IJCSI International Journal of Computer Science Issues, Volume 13, Issue 5, September 2016
ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784
www.IJCSI.org https://doi.org/10.20943/01201605.146152

Need of Autonomic Management SaaS Application

Nadir K.Salih[1], Tianyi Zang[2]
Department of Electrical and Computer Engineering, Engineering Collage, Karary University, Sudan[1]
School of Computer Science and Engineering, Harbin Institute of Technology, China[2]

ABSTRACT*-Technology Building a SaaS with existing technology is hard new software technology, useful for both business and education purposes Business can be easily adopted in several domains, such as Healthcare, education and OA (Office Automation). SaaS application gave many aspects of business management. For that it became available and using in many domains. It will be needed to realize customer's requirements from design to runtime. Wherefore the modeling issue is very important for SaaS application. We will follow model driven architecture for mapping from source of model to the target of the model. In this paper we described new model for SaaS application to simplify management. By benefit from meta-model and type graph we dynamically generated instances to our model. And showed two cases study first bank system to show the need of autonomic management for SaaS application and the other SaaSHER to describe the new model for SaaS application.*
Keywords, meta-model, type graph, autonomic management, SaaS application

I. INTRODUCTION

To develop any application we should use model driven development technique. That it defined many concepts. Like abstract class is a class that cannot be instantiated, it exists extensively for inheritance and it must be inherited [1][2]. Meta-modelling, is the analysis, construction and development of the frames, rules, constraints, models and theories applicable and useful for modelling a predefined class of problems [3][4]. According to the Meta-Object Facility (MOF) standard, a meta-model is a model that defines the language for expressing a model [5][6][7]. Meta-model is model's model that serves for explanation and definition of relationships among the various components of the applied model itself [8][10].

Multi-graph is a graph with multiple edges between the same vertices. Formally G(V,E,F): a multi-graph is a set of vertices V along a set of edges E , and a function F mapping from E to V. The function F shows which vertices are connected by which edge [9].The SaaS Application needs to develop for satisfy users. The important contribution of this paper is to show autonomic management is very important to SaaS application. And we have defined the benefit from meta-model and type graph to dynamically generate instances for SaaS application.

In the next section, we will discuss related work. Afterwards, in Section III we describe the general description for SaaS Model, and how it can be applied to cloud applications. In Section IV we formally describe the formal problem description. This is followed by Section V, where we described different approached to Solution Techniques. Subsequently, in Section VI we highlight to autonomic management. Finally, Section VII contains our conclusions.

II. RELATED WORK

As we understand SaaS application is new model for business process. In [11] they defined Multi-layered customization framework supporting continuous testing and recoverability. In [12][29] author they Executing of configurable and multitenant SaaS application. In [13] they depend on Multi-tenancy is increased utilization of hardware resources and improved ease of maintenance. In [14] they solved the problem of orchestrating SaaS business processes based on BPEL. The authors in [15][16] they focused on tenant-aware meta-data management and Open multi-tenant architectural blueprint based on a real world scenario. In [17][18] by Hybrid approach they solved placement of tenants and Innovative multi-layered customization framework. Authors in [19] they used COP achieves a higher customization flexibility. In[20][21][22][28] they Support SaaS providers in managing the variability of SaaS applications and their requirements, Calculations of resource requirements for multi-tenants with applied constraints in a shared application instance, and reduce its complexity by decoupling its management through different application layers. SaaS reference architecture must support at design time as well as at runtime this opinion defined in [23]. By three architectural patterns that support variability in multi-tenant SaaS environments in [24] they Customizable SaaS. In [25][26][30] the authors they depicts the design space and represents the common and variant parts of SaaS architectures, and Templates and derived fixed and tenant-specific parts of a solution. That is almost recent researches in SaaS application they didn't mention or define methods to autonomically management SaaS. For that we show the necessity of autonomic management for SaaS application to improve the business process.

III. GENERAL DESCRIBE MODEL

As depict in figure1 our model for SaaS application we have three levels Provider-Model, Tenant-Model, and User-Model. Tenant-Model looks as instance of meta-model and User-Model as instance of Tenant-

Model. Any level has the same layers application but it is different perspective from level to level.

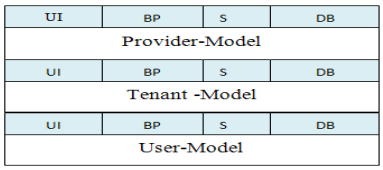


Fig1 SaaS application model

User Interface(UI): it is layer can be found in every model level begin from Provider-model that show the provider information about all tenants like define the style of User Interface is difference from tenant to tenant. In tenant level will be responsible from User Interface of user model how it look like how controlling it. In User-Model can put some features in User Interface and change by user.

Business Process (BP): can category the activity process is difference from level to level. For example workflow process in provider-model will be difference from tenant to tenant according on the requirements changing. The same workflow process in tenant-model will different from user to user if they have not typical demands. In user model can put some optional activity process for user for customization his process.

Services(S): service layer in provider-model will describe services that can be introduced from provider for tenants. Like analysis events and make reporting. In tenant-model will deliver some services for user. For user-model we can encourage user by setup some services when he want.

Database (DB): this layer can define the access of data and storage it in memory. We can put the access of data for three levels depend on security issues. In addition the data storage for any tenants can be managed by provider-model. Tenant-model can tune and mining user data. For user-model can put different organized for various users type.

IV.FORMAL PROBLEM DESCRIPTION

To explain the Challenges in SaaS Systems Management we use example of bank system as SaaS application from IBM [27]. We will define our model through it and show why we need autonomic management for SaaS application. From the case diagram we have three models Provider-model (Administrator), tenant-model (bank), and user-model (customer) as depict in figure2 below.

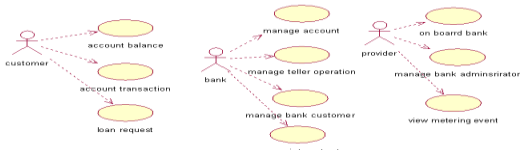


Fig2 Case diagram of bank system

Provider-model:

We have to describe: administrator for all tenants On-board bank: include information for every bank (ID, Name, and State (register, provision)) can delete or update and show the details of any bank. In addition provider administrator can add new bank. Manage bank administrators: can change (delete, update) information of bank administrator or add new administrator of bank. View metering events: the system filter metering event according to date and agent and subscriber to display the details of operation like (add account, add customer) In this level of Provider-model because we would manage many tenants for that we need to Self-configuration for GUI according to tenants requirements it will help system to change dynamically in runtime. For example: say we have two kinds of tenants normal and VIP if the tenant have exceed limit number of customers and transactions will be VIP tenant for that the system will be monitor and analyze for plan execution to show different GUI for every tenant. Self-optimization need in bank system for optimize QoS and tuning of resources can do self servicing to delete or update tenant or self reporting to display the operation for every tenant. For example can monitor and analyze data show the result of risk that can occur from debit or credit services in any time and so monitor workflow (BP) it can be different from tenant to tenant according to some policy. Self-healing to monitoring the bank system for any fail or error can occur to prevent it or solve problem. For example monitoring the size of data memory for every tenant if it will be reach critical case the system will make indicator to show what will be happen. Self-protecting is very important in SaaS application exactly if we use multi-tenancy to share database and schema in bank system. The system ensures every tenant could have access to his data.

Tenant-Model

Services can do by tenant (bank administrator for all customers) Manage Account: can add or modification customer's account, have ID, type (checking, savings), balance. Manage teller operation: by account ID get all transaction (Check cashing, depositing, transfers, wire transfers, Payment collecting…etc). Manage bank customers: query customers information views all customers, add new customer Manage interest rate: find all rate, add new rate. We need in Tenant-Model Self-configuration for GUI to categorize customers in the same bank. The style of GUI can be change dynamically according to customer activity. For example the customer who is having a lot of transaction through the bank will show him additional services can give through the bank according to his balance. We need Self-optimizing in bank system in tenant-model need to optimize services to serve a lot of customers by less costing from provider. For example we can make self-service in customers transaction and self-reporting to measure and qualify the efficiency and resources using. We need Self-

healing for any business process error in work flow if the customer changes the ordering of steps. For example the steps of Checking transaction apply by steps of Savings this transaction will be add to Saving transaction in total balance. Self-protecting any customer need full security to his data

User-Model

Users have to do a lot of process like Account balance: can view account information (account-ID, type, and total-balance) Account transaction: display all transaction for Account by date and type like debit, credit process. Loan requests: see the current interest rate for bank define the product number and type (cheap, state). And show the last loan request by date and amount, status (approved, reject). In this level we need to Self-configuration the system let the customer to customize his style that he wants. Self-optimization can make self-searching to see all option for transactions that can appear for customers. Self-healing the system can self correct the error of customer like enter error data, make error in process credit or debit. Self-protecting the system need to protect this level of user model not let any unauthorized user can access the system. From this description of this running example we briefly obtain these Challenges in SaaS Systems Management:

- Large-scale, heterogeneous distributed systems with highly dynamic, complex multi-component interactions.
- Large volumes of real-time high-dimensional data, but also lots of missing information and uncertainty.
- Too much complexity, too few (skilled) administrators.

For these three challenges we trusted SaaS application need for self-managing for development and evolution systems. In addition autonomic management will lead to realize the feature of SaaS application by minimizing costing and increasing performance.

Derivation model

According to Meta-Object Facility the model will be have three sequences Instance (Inst), Model (M), and Meta-Model (MM). As we depict in figure3 we have three models Provider Model (PM), Tenant Model(TM), and User Model they corresponding SaaS application. User model look like instance, tenant-model stand like a model and provider model show the meta-model.

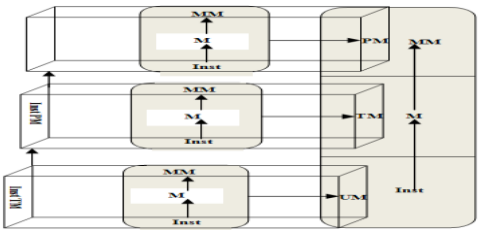


Fig3 MOF for SaaS application

To explain the sequence of MOF in our SaaS application model we can take SaaSEHR application as example see figure4.

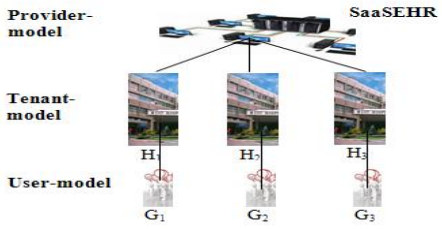


Fig4 SaaSEHR application

From provider we have the application and all resources management, the application has functions will achieve by tenant-model this level will be look as administrator for all patient belong to unique hospital, and in user-model have groups of patients every group has the same services and typical characteristics for application. We can describe the SaaSHER model in hierarchy sequence as depict in figure5.

Provider model is UML meta-model Models the language UML, i.e., defines concepts like classes, attributes, associations, contains descriptions of elements that can be used to describe the models on the tenant model layer.

Tenant model is UML-model by using any UML diagram, we instantiate the UML meta-model and obtain a UML model it contains application-specific models.

User model Run-time Instances Real instances of the models, contains concrete run-time instances. Note the difference between instance specification and real instance!

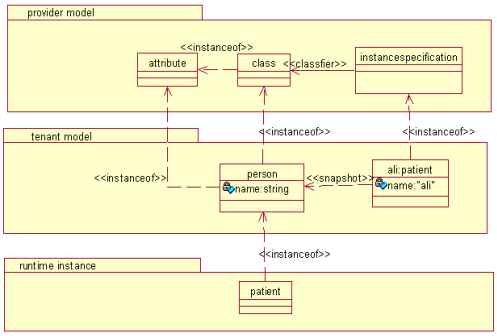


Fig5 hierarchy sequence of model

V. SOLUTION TECHNIQUES

In system workflow we look for our model from bottom to top. For example we have G_1 is group of patients belong to hospital H_1 is high level care, G_2 is group of patients belong to hospital H_2 is middle level care, and G_3 is group of patients belong to hospital H_3 low level care. The activity processes for patient in

SaaSEHR application can be depict in figure6 from start to end.

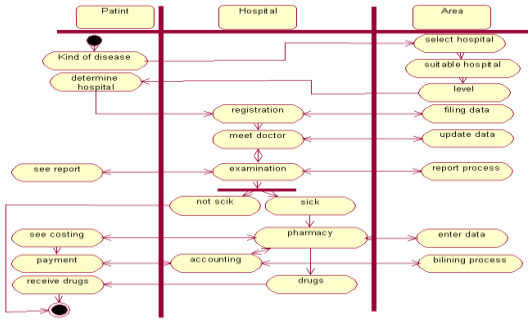


Fig6 Workflow model of SaaSEHR

If we begin with User-model we can take three processes: Select hospital, the report of result, Payment.

Meta-model of SaaSEHR define Domain Specific Languages (DSLs) see the figure8. From our system workflow we obtain meta-model include the class and independencies of SaaSEHR application as depict in figure7 bellow. It contains meta-classes, meta-associations and cardinality constraints.

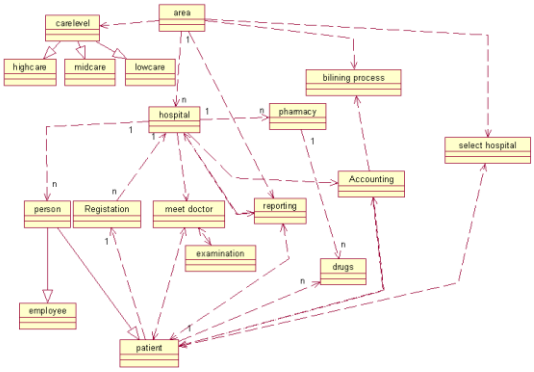


Fig7 Meta-model of SaaSEHR

There is a need for a systematic derivation of instances of meta-models. Each model must be an instance of a "meta-model", a meta-model being the specification of a set of models. The instance of the meta-model must conform to the cardinality constraints. In addition, instances of meta-models may further be restricted by the use of additional constraints specified in the Object Constraint Language (OCL). The instance of the meta-model must conform to the cardinality constraints.

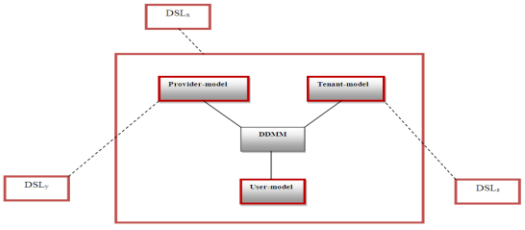
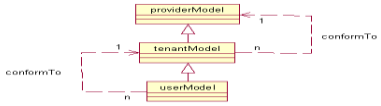


Fig8 Meta-model and DSL

Meta-models define Domain Specific Languages (DSLs). A DSL is a coordinated set of models. DDMM is Domain Definition Meta-Model .Each model in our system will be reference to other model. Tenant-model will reference to Provider-model and user-model will be reference to tenant-model as depict in figure9.



Formal Description of Generation

In our model of SaaS application as depict in figure10 we generate tenant-model instance from provider model according to class, association from source to target to show mandatory, optional, XOR, and OR rules, and constraints to remark the associations is require or exclude. A meta-model can be considered as a class diagram on the meta-level, i.e. it contains meta-classes, meta-associations and cardinality constraints. Instances of meta-models may further be restricted by the use of additional constraints specified in the Object Constraint Language (OCL).Typed graph transformations with inheritance will be the basis for the formal background for instance generating graph grammars. From object oriented modeling we have concert and abstract type that depend on inheritance, like we can describe type graph have node and edge from source node the edge will reach the target. For that type graph with inheritance will be show asset of nodes belong to node source. An instance of a meta-model is a concrete model that conforms to its meta-model.

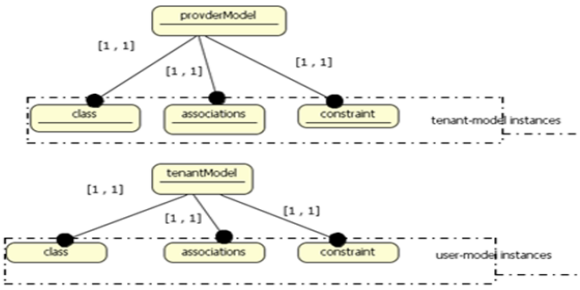


Fig10 Generating Instances

Definition 1 (Type graph with inheritance) A type graph with inheritance is a triple TGI = (TG, I, A) define a type graph TG = (TG_V ,TG_E , S_{TG}, T_{TG}) (with a set TG_V of nodes, a set TG_E of edges, source and target functions S_{TG}, T_{TG} : $TG_E \rightarrow TG_V$), an acyclic inheritance relation $I \subseteq TG_V \times TG_V$, and a set $A \subseteq TG_V$, called abstract nodes. For each $x \in TG_V$, the inheritance clan is defined by clan I (x) = $\{y \in TG_V \mid (y, x) \in I^* *\}$, where $I^* *$ is the reflexive-transitive closure of I .In type graph have inheritance in node type and edge type depend on source and target to show the

function this called as clan morphism have the same feature.

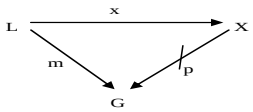
Definition 2 (Clan morphism figure 12) LetTGI = (TG, I, A) with TG = (TG_V ,TG_E , S_{TG}, T_{TG}) be a type graph with inheritance. A clan-morphism ctp : G → TGI from a graph G = (G_V , G_E , S_G, T_G) to TGI is a pair ctp = (ctp_V : G_V → TG_V , ctp_E: G_E → TG_E) such that for all e ∈ G_E the following holds:

- $ctp_V \circ S_{G(e)} \in clanI\ (S_{TG} \circ ctp_{E\,(e)})$ and
- $ctp_V \circ T_{G(e)} \in clanI\ (T_{TG} \circ ctp_{E\,(e)})$.

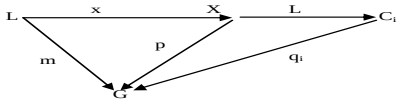
(G, ctp) is called a clan-typed graph.

In graph grammars it have rule for transformation in application this rule have left-hand side and right side left is called as source graph and copy of replace left hand by right hand side lead to the target. For controlling transformation used negative application condition NAC(x).

Definition 3 (Application condition) A negative application condition is of the form NAC(x), where x: L → X is an Injective morphism. A morphism m: L→G satisfies NAC(x) if there does not exist an injective morphism p : X → G with p ◦ x = m:



An atomic application condition is of the form P(x, ∨i ∈I xi) where x: L → X and xi : X → C_i with i ∈ I are injective morphisms. A morphism m: L → G satisfies P(x,∨i∈I xi) if for all injective morphisms p: X → G with p◦x = m there does exist an i ∈ I and an injective morphism qi :Ci → G with qi ◦xi = p:

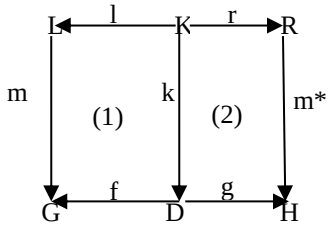


Definition 4 (Rules) A rule typed over a type graph TGI = (TG, I, Abs) with inheritance is given by p = (L l ←K r→ R, A_p), where L, K, R are clan-typed graphs, l and r are type-preserving injective graph morphisms, ctp−1 R (Abs) ⊆ r (KV), and Ap is a set of application conditions of the form NAC(x) or P(x,∨i∈I xi) as defined in Def. 3.

Definition 5 (Rule matching and application) Given a rule p as in Definition 4 and a clan-typed graph (G, ctpG), then m is a match of p in G if

- m is an injective morphism of the left-hand side L of the rule p = (L l← K r→ R, Ap) as defined in Definition 4 in the graph G;
- $t_K\ (x_1) = t_K\ (x_2)$ for $t_K = ctp_G \circ m \circ l$ and x1, $x_2 \in K_V$ with r (x_1) = r (x_2);
- m satisfies all simple negative application conditions and all atomic application conditions in Ap.

Given a match m, a direct derivation (G, ctp_G) p⇒,m (H, ctpH) exists if there is a span of



graph morphisms G←D→H and a co-match m* : R→H of p in H where (1) and (2) are pushouts in the category of Graphs TG .

Given a rule set *R*, *(G,* ctp*G)*⋆⇒ *R (H,* ctp*H)* is a finite sequence of an arbitrary number of direct derivations by rules of *R*. A derivation *(G,* ctp*G)* ⋆⇒ *R (H,* ctp*H)* terminates, if ∃*r* ∈ *R* : *(H,* ctp*H)* ⇒*r (H ,* ctp*H)*.

As we mention tenant-model is instance of provider-model and user-model is instance of tenant-model. We want to generate instance from meta-model like generate tenant-model from provider-model and generate user-model instance from tenant-model in a systematic way. We can generating instances by three layers layer1 classes have some abstract and other is concrete class have inheritance feature.layer2 see instance drive from associations from source to target to show mandatory, optional, XOR, and OR rules. Layer3 constraints to remark the associations are require or exclude. From our model if we take the provider-model in layer1 we must determine all concrete classes' inheritance as we see in SaaSEHR model the carelevel class has inheritance to lowcare, midcare, and highcare as depict bellow in figure13 here we can generate new instances because we have option classes this feature will generate different instance for different a kind of hospitals according to carelevel. For tenant-model can generate instances for different persons see figure11

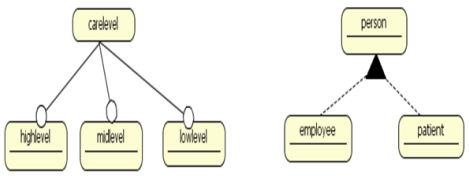


Fig11 Inheritance class

Layer2 instances can generate from association that have rules lead to various instances from meta-model show multiplicity value $mv(E_i)=(min,max) \in M$ to each edge E_i such that $min;max \in Z \wedge min \geq 0;max > 0$.

- Mandatory relationship: A f-arc $E = (T_E,H_E)$ such that $|H(E)|= 1 \wedge label(E) = [1\ldots1]$
- Optional relationship: A f-arc $E = (T_E,H_E)$ such that $|H(E)| = 1 \wedge label(E) = [0\ldots1]$
- Alternative relationship (XOR): A f-arc $E = (T_E,H_E)$ such that $|H(E)| = q = q > 1 \wedge label(E) = [1\ldots1]$
- "Addition" relationship (OR)A f-arc $E = (T_E,H_E)$ such that $|H(E)| = q = q > 1 \wedge label(E) = [1\ldots q]$.

For Association we can show in type graph which edge is mandatory will be include in every instance like edge between area and hospital because every area must have hospital. By option edge can create different instance like the patient can select hospital according on levelcare. In alternative edge can generate various instances because in a time must use only way from multi like the payment of patient can be from bank, cash, and insurance. In addition OR edge instance can be variable because two choice can select in time or can select only one like in tenant-model if we want to talk about process for employees just the instance will describe processes of a employee person work in hospital, however if we have processes joint between employees and patients here the instance in tenant-model will describe processes for all persons in hospital.

Layer3 the constraint for nodes first is require constraint A f-arc $E = (T_E,H_E)$ such that $|H(E)| = q = q \geq 1 \wedge label(E) = [q\ldots q]$ The semantic meaning of a require edge is that the node in the tail set imposes the constraint of selecting all nodes in the head set. The semantic meaning of a mutex edge is that it is not possible to select more than one of the edge in the tail set at the same time A f-arc $E = (T_E;H_E)$ such that $T(E) = Root \wedge |H(E)|= q / q > 1 \wedge label(E) = [0\ldots1]$.

In this layer require constraint can define instance by require node In example in tenant-model if the hospital just used payment from bank that lead to require account in bank for patient. For mutex constraint generates instance will different because the selecting will be for only one edge from two or many. Like in patient payment workflow we have two kind's workflow with insurance or without it. You couldn't take both workflows in same time.

VII. CONCLUSIONS

The track record of success is accelerating the rate of adoption and expanding the range of applications that are being converted to the SaaS delivery model. It is also dramatically changing the competitive landscape of viable SaaS providers. In this paper our novelty is show new describe for SaaS application model. And by used meta-model and type graph autnomicaly generated instances. In addition the classification of SaaS model in three levels it will be easy in future to management SaaS application in autonomic way.

ACKNOWLEDGEMENT

This work has been developed with the support under the project with number: 2012AA02A604, 863 Program key projects in China: The Technology and the System Development for Smart Acquirement of Personal Healthcare Information. And so the Key Project of NSF in China: Methodology of Value-oriented Software Services: Theory, Method and Application with number: 61033005.

.

REFERENCES

[1] T. Zang, R Calinescu, M. Kwiatkowska. Metamodel-driven SOA forcollaborative e-science application. International Journal of Computer Systems Science & Engineering,2011.

[2] D. Fogli, L.P.Provenza. A meta-design approach to the development of e-government services. Journal of Visual Languages and Computing,2012,pp. 47–62.

[3] C. Atkinson, T. Kühne. Model-Driven Development: A Metamodeling Foundation. IEEE SOFTWARE,2003.

[4] T. Lemattre, B. Denis, J-M. Faure. Using a meta-model to build operational architectures of automation systems for critical processes. IEEE ,ETFA. 2011.

[5] X. Zhang, K. He, J Wang, J. Liu, C. Wang, H. Lu. On-Demand Service-Oriented MDA Approach for SaaS and Enterprise Mashup Application Development. International Conference on Cloud Computing and Service Computing.IEEE,2012.

[6] D. Berardi , D. Calvanese, G. De Giacomo. Reasoning on UML class diagrams. Artificial Intelligence. Elsevier. 2005.

[7] E. Brottier, F. Fleurey, J. Steel, B.Baudry, Y. Le Traon. Metamodel-based Test Generation for Model Transformations: an Algorithm and a Tool. International Symposium on Software Reliability Engineering, IEEE, 2006.

[8] L.Pedro, L. Lucio, D.Buchs . Modeling Languages System Prototype and Verification Using Metamodel-Based Transformations. Published by the IEEE Computer Society,2007.

[9] H. Wu, R. Monahan, J. F. Power. Exploiting attributed type graphs to generate meta-model instances using an SMT solver. IEEE, 2013.

[10] A. Cicchetti, D. Di Ruscio, D.S. Kolovos, and A. Pierantonio. A test-driven approach for metamodel development. European Community's 7th Framework Programme. 2007.

[11] Wei. Tsai, Q.Shao, Y. Huang, X. Bai. Towards a Scalable and Robust Multi-tenancy SaaS. Second Asia-Pacific Symposium on Internetware. ACM, 2010.

[12] S. Kang, Sungwon Kang, S. Hur. A Design of the Conceptual Architecture for a Multitenant SaaS Application Platform.IEEE, 2011.
[13] C. Bezemer, A. Zaidman. Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare, Report TUD-SERG,2010.
[14] Y. Shi, S. Luan, Q. Li, H. Wang. A Flexible Business Process Customization Framework for SaaS.IEEE, 2009.
[15] O. Schiller, B. Schiller, A. Brodt, B. Mitschang. Native Support of Multi-tenancy in RDBMS for Software as a Service. Proceedings of the 14th International Conference on Extending Database Technology ACM,2011, pp.117-128.
[16] Irene S. Harris, Z. Ahmed. An Open Multi-Tenant Architecture to Leverage SMEs. European Journal of Scientific Research. Vol.65 No.4, pp. 601-610, 2011.
[17] E. Yang, Y. Zhang, L. Wu, Y. Liu, S. Liu. A Hybrid Approach to Placement of Tenants for Service-based Multi-tenant SaaS Application. Asia -Pacific Services Computing Conference. IEEE, 2011.
[18] W.Tek Tsai, Q. Shao, W. Li. OIC: Ontology-based Intelligent Customization Framework for SaaS. International Conference on Service-Oriented Computing and Applications (SOCA) IEEE,2010.
[19] E. Truyen, N. Cardozo,S. Walraven, J. Vallejos, Bainomugisha, S. Gunther, T. Hondt, W. Joosen. Context-oriented Programming for Customizable SaaS Applications. Proceedings of the 27th Annual Symposium on Applied Computing ACM, 2011, pp. 418-425.
[20] Ralph Mietzner, Andreas Metzger, Frank Leymann, Klaus Pohl. Variability Modeling to Support Customization and Deployment of Multi-Tenant-Aware Software as a Service Applications. ICSE'09 Workshop, IEEE, 2009, 18-25
[21] T. Kwok, A. Mohindra. Resource Calculations with Constraints and Placement of Tenants and Instances for Multi-tenant SaaS Applications. Springer, 2008, pp. 633–648.
[22] Ali Ghaddar, Dalila Tamzalit, Ali Assaf. Decoupling variability management in multi-tenant SaaS applications. International Symposium on Service Oriented System Engineering, IEEE.2011.
[23] Julia Schroeter, Sebastian Cech, Sebastian Götz, Claas Wilke, Uwe Aßmann. Towards Modeling a Variable Architecture for Multi-Tenant SaaS-Applications. Sixth International Workshop on Variability Modeling of Software-Intensive Systems.ACM, 2012.
[24] Jaap Kabbedijk, Slinger Jansen. Variability in Multi-tenant Environments: Architectural Design Patterns from Industry, Springer, 2011.
[25] K. Öztürk, B. Tekinerdogan, Feature Modeling of Software as a Service Domain to Support Application Architecture Design. International Conference on Software Engineering Advances. IARIA, 2011.
[26] Hyun Jung La, Soo Dong Kim. A Systematic Process for Developing High Quality SaaS Cloud Services. Springer, 2009.
[27]IBM, bank system. https://www6.software.ibm.com/developerworks/offers/techbriefings/cc4dreplays/session2_dcarew.pdf.
[28] Nadir K Salih, Tianyi Zang. Autonomic Management for Applicability and Peformence in SaaS Model. International conference on parallel and distributed processing techniques and applications (PDPTA'14), held in July 21-24 Las Vegas,USA.- 2014
[29] Nadir K Salih, Tianyi Zang. Modeling and Self-Configuring SaaS Application. International conference on software engineering research and practice (SERP14), held in July 21-24 Las Vegas,USA.- 2014.
[30] Nadir K Salih, Tianyi Zang. Variable service process by feature meta-model for SaaS Application. IEEE International Conference in Green and Ubiquitous Technology, IEEE, 2012, pp102 – 105.