Dynamic metadata for customized deployment in a multi-tenant B2B setup

Aasawaree Deshmukh*
Divya Awatramani*
Shilpa Agrawal*
Varun Gadia*

Abstract

Software as a service (SAAS) provides enterprises with a medium to deploy scalable and extensible web services to their clients based on their contracts. In a multitenant business to business setup often there is an anomaly as the tenants have varied needs in terms of their deployment and interfaces. In statically compiled applications the needs of runtime requests and polymorphism in terms of the deployed package cannot be met. For these reasons in this paper we propose a solution to this problem by considering a framework that supports dynamic metadata-data about the application itself. The hosting enterprise does not have to deploy a separate service for every tenant and each tenant in the setup can customize his application and independently execute the service In well-defined metadata driven architecture the compiled runtime engine (kernel) and the metadata clearly differ from each other. Due to this clear separation tenants can individually customize their contracts and make any relevant changes dynamically without affecting the overall performance or working of the setup.

Keywords: Software as a service, metadata, polymorphism, multitenant setup.

1. Introduction

Traditionally, in order to use any application or service enterprises had to purchase voluminous licenses to deploy the software instances for every single machine. However, with every company using varied platforms, operating systems and hardware it is nearly impossible to provide data access with a tool that is hardware and software independent. Also it could turn out to be cost ineffective and inconvenient in case of large enterprises. Limitations were further felt when the need to upgrade any deployed service came into existence. SAAS[1], Software as a service is a platform that provides a solution to these needs by providing a cost effective way to access or deploy any web based service without involvement of any hardware, software and operating system constraints.

In a SAAS based architecture the services are maintained and hosted by a single deployment server. The users can access any specific service deployed by the hosting environment through the web. SAAS also provides a number of advantages in terms of convenience to use, painless upgradation and seamless integration. Since the services are deployed remotely by a hosting environment, SAAS finds its importance in enterprises that spawn across continents. Multitenancy is the key for the efficiency of SAAS. General single tenant applications require a

*Department of Computer Science, Pune Institute of Computer Technology, Pune, Maharashtra

dedicated set of resources to manage the requirements of a single enterprise. However with multitenancy we can pay heed to the needs of multiple enterprises through resources meant for a single enterprise.

In multitenant setup each tenant has its own schema that is used to depict the components of the deployed service that it is accessing.

In this paper we discuss the need for dynamic metadata so that the tenants can customize their schema the way they want. In the general scenario all the tenants have to make use of the service as it is and the deployed service and its components are identical for all. However certain tenants may have varied requirements and to address them they may need to make certain changes in their contract. In such a case, we propose the model where clients will be able to describe the schema of their business objects and the metadata system will expose the core storage model of the infrastructure in the client domain model so that each client gets their data in the schema that they have described.

2. Metadata Driven Architectures

Multitenancy requires that each tenant be able to customize his contract individually to provide support for applications that are reliable, upgradeable, secure and fast. It should also be made sure that each tenant's data is kept secure in the shared database so that all tenants act in a mutually exclusive manner. The tenant should be able to customize the application's interface and other objects in real time. Also the application should be upgraded without compromising on the tenants' customizations.

All these requirements cannot be made in a statically compiled application as they do not support runtime modifications in their schema. For multitenancy the application should support polymorphism to meet the requirements of the different users which is not possible in case of statically compiled applications.

The solution we propose for this problem is to consider a framework that exposes client\tenant specific "views" on top of a singular core service deployment. The tenant specific "view" is defined by

i. Any tenant specific data contracts over and above the core data contract.
ii. Any tenant specific business rules.
iii. A dedicated client service endpoint.

In a service oriented contract based system, such logical views over a single core infrastructure require that that metadata (WSDL incase of SOAP web services) that is generated at the client endpoint is specific to the tenant "view". This metadata would be the super set of the core service data contract as well as the client specific data contract. This would imply that the said runtime infrastructure should support:

I. Tenants would have the ability to publish their own data contract schema.
II. Dynamic translation to and from the core data contract to the tenant specific data contract.
III. Generation of dynamic metadata (WSDL) for a given tenant endpoint.

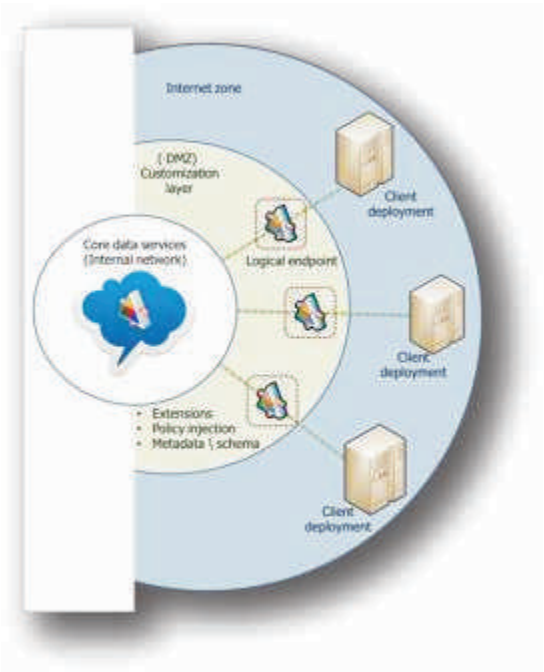


Fig 1: Typical metadata driven architecture in multitenat cloud based setup.

3. Mathematic Model

S= {DBS, USER, APPLICATION, REQUEST, OBJ, FIELD, FUNC, METADATA, OUTPUT | Φ}

Φ=Φ1+Φ2+Φ3+Φ4.

Φ1=rules such that the DBS server contains the application base.

Φ2=rules for no duplicate objects created for same application.

Φ3=no duplicate fields generated.

Φ4=metadata should not be empty.

DBS = The server side database that creates the application.

USER = The set of all the user_id for all users requesting for the application.

APPLICATION = {app_id1…}: set of application request dynamically created.

REQUEST = {r1, r2,……, r_n | Φ }

REQUEST = Set of all the user requests for the application.

Φ = $r_i \neq r_j$ where i ≠ j.

As if the request type's are similar then no replication for the application will be done.

OBJ= {{user_id, r, app_id, {f_i, v_i}, func1}, {user_id, r, app_id, {f_i, v_i},func2},…….} is a set holding intermediate data for each input application.

Field = {f1, f2,……, f_n} is a set of fields which will be dynamically created and added to the database.

Φ = $f_i \neq f_j$ where i ≠ j.

FUNC ={func1,func2,……funcn} Set of functions associated with respective Application(OBJ).

METADATA = {{ki,vi},{ki+1,ni+1}……} Set of key value pairs for various fields.

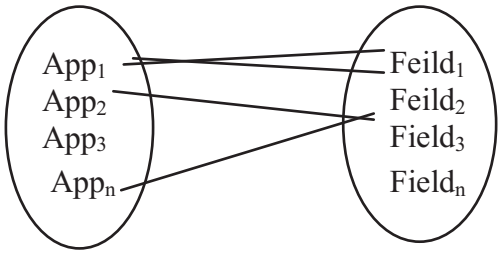
OUTPUT = {o1,o2,…….on} Set of GUI created for every application(OBJ).
F1 => (user_id, r_i) ------ >app_id.

Every client selects specific fields(fieldi) for his or her application which will be linked with application Id(app_id | f1) using F2

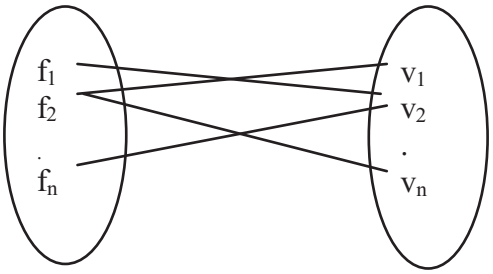
$$F_2 = \sum_{k=1}^{N1} \sum_{i=1}^{N2} (F1\ (user\_id_k, r_k), field_i) \sum_{j=1}^{N1} W$$

Where W is a set that represents how app_id and various fields are related.
N1 represents the number of application requests.
N2 represnets the number of fields linked with its respective application.

R1 is a many to one and onto relation between Applications (OBJ) and fields



R1 => APP⟶FIELD
R2 is a many to many and onto relation between fields and validations

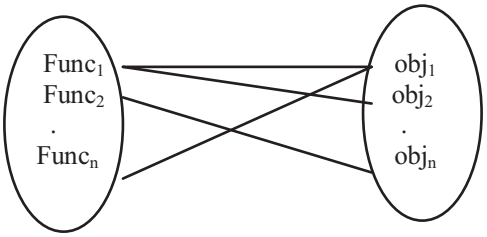


R2 => field⟶validation
W_k obtained through F2, relations such as appid->field, but each field represented here has a value, this relation between each field in W_k and its subsequent values have been linked using F3 and represented using the set IN_OBJ.

$$F_3 => \sum_{K=1}^{N1} \sum_{i=1}^{N2} (W_k, v_i) \longrightarrow \sum_{j=1}^{N1} IN\_OBJ_j$$

Where IN_OBJ = (user_id,app_id, {f_i, v_i})

R3 => functions ⟶IN_OBJ



R3 is many to many and onto relationship between functions and each IN_OBJ.
IN_OBJ obtained through F3 conatins fields such as Wk->Value, however each IN_OBJ here performs a particular function (operation) the linking between each IN_OBJ and its associated functions is given by F4 and is represented by the set OBJ.

$$F_4 => \sum_{K=1}^{N1} \sum_{i=1}^{n2} (IN\_OBJ_k, func_i) \longrightarrow \sum_{j=1}^{n1} OBJ_j$$

Where OBJ = (user_id,r, app_id, {f_i, v_i}, func)

OBJ obtained from F4 needs to be converted to the Intermediate data format, ie the {key,value) format which is done using F6.

F_6 => F_6 (OBJ) ⟶ {key, value} = METADATA

This Intermediate data format <<METADATA>> , {key,value} is converted to OUTPUT.
F_7 => METADATA ⟶ OUTPUT

4. Probability

p->1 when r->1 i.e. probability of creating of the application without any failure or delay is 1 when there is only one application request.
Similarly p ⟶ 0 when r ⟶8 .

5. Morphism

It includes morphism as the same DBS is replicated for various applications.
DBS----> {app_1, app_2, ….., app_n}
Same fields are used whith diff. validations for various applications.
$Field_i$ € App_i (where i=k) some constant and j varies from 1 to n.

6. NP-HARD

As the no. of requests and the applications id's goes on increasing the load on the server increases and the probability of running all applications successfully tends to zero. So some of the requests for application will be dropped at the server side only.

A condition on the no. of requests to be served at a particular instance t will transform this problem into NP-COMPLETE problem.

7. Success

count(OUTPUT)=count(REQUEST)
All the request for applications have been served
For an particular app_id:
D(fi,vi)/dt=0 as the fields requested by the user for an application should not change with time.

8. Failure

1. count(OUTPUT) ≠ count (REQUEST)
2. count(REQUEST) > n

*count :- a function that returns the total no. of elements in the set.

9. Future Enhancements

A dynamic metadata stack is just one of the core facets of a customizable multi-tenant system. It would allow for a contract based metadata exchange between remote systems on a data contract that is specific to each tenant. The other supplementary infrastructure of a multi-tenant system that would be required along with this is:

i. A policy injection driven business rules framework to plug-in client customization without changing the core functionality.
ii. A document-oriented core business model which allows for addition of client specific data attributes without conflicts with the core model data.
iii. A tenant configuration management system which allows for pulling of client specific business rules and runtime configuration.
iv. A user friendly GUI to allow for clients to "build their own" data objects for consumption.

10. Conclusion

In this paper we have discussed the metadata model for generating application requests runtime from clients in a B2B setup. The main advantage of this model is that clients will be able to customize their contracts the way they want and they do not need to compromise on their demands and requirements. Also, the security, reliability and scalability will be high. New clients can keep getting added to the existing setup without affecting the overall performance of the system.

11. References

1. *Zhengxiong Hou, Xingshe Zhou, Jianhua Gu,Yunlan Wang,Tianhai Zhao. ASAAS: Application Software as a Service for High Performance Cloud Computing, Center for High Performance Computing Northwestern Polytechnical University,Xi'an, China, 2010.*
2. *Thomas Kwok, Thao Nguyen and Linh Lam, A Software as a Service with Multi-tenancy Support for an Electronic Contract Management Application, IBM Research Division,Thomas J. Watson Research Center,19 Skyline Drive, Hawthorne, NY 10532,2008.*
3. *Pu Liu1 and Michael J. Lewis2, Uniform Dynamic Deployment of Web and Grid Services, Systems and Technology Group, IBM, Endicott, NY, 2007.*
4. *Francisco Curbera and Nirmal Mukhi, Metadata-Driven Middleware for Web Services, IBM Research, 2003.*
5. *Eric Konieczny, Ryan Ashcraft, David Cunningham, Sandeep Maripuri, Booz Allen Hamilton, Establishing Presence within the Service-Oriented Environment, 8283 Greensboro Drive, McLean, VA 22102, 2009.*
6. *Sungjoo Kang1, Sungwon Kang2, Sungjin Hur1, A Design of the Conceptual Architecture for a Multitenant SaaS Application Platform, Electronics and Telecommunications Research Institute, Department of Computer Science, Korea Advanced Institute of Science and Technology, 2011.*
7. *Florian Rosenberg, Philipp Leitner, Anton Michlmayr and Schahram Dustdar, Integrated Metadata Support for Web Service Runtimes Distributed Systems Group, Technical University Vienna Argentinier Vienna, Austria.*
8. *Francisco Curbera and Nirmal Mukhi, Metadata-Driven Middleware for Web Services,IBM Research,2003.*