CLOUDIO: A Cloud Computing-oriented Multi-Tenant Architecture for Business Information Systems

Enrique Jiménez Domingo
Javier Torres Niño
Angel Lagares Lemos
Computer Science Department
Universidad Carlos III de Madrid
Leganés, Madrid, Spain
enrique.jimenez@uc3m.es
javier.torres@uc3m.es
angel.lagares@uc3m.es

Miguel Lagares Lemos
Ricardo Colomo Palacios
Juan Miguel Gómez Berbís
Computer Science Department
Universidad Carlos III de Madrid
Leganés, Madrid, Spain
miguel.lagares@uc3m.es
ricardo.colomo@uc3m.es
juanmiguel.gomez@uc3m.es

***Abstract*— Cloud Computing is evolving from a mere "storage" technology to a new vehicle for Business Information Systems (BIS) to manage, organize and provide added-value strategies to current business models. However, the underlying infrastructure for Software-as-a-Service (SaaS) to become a new platform for trading partners and transactions must rely on intelligent, flexible, context-aware Multi-Tenant Architectures.**

In this paper, we present Cloudio, a Cloud Computing-based metadata-powered Multi-Tenant Architecture, backed with a proof-of-concept J2EE implementation.

Keywords: Cloud, Multi-Tenancy, business, architecture

I. INTRODUCTION

Cloud Computing has provided a reliable, cost-efficient, cutting-edge infrastructure for advanced software delivery models such as Software-as-a-Service (SaaS). Most SaaS platforms are built on Multi-Tenant Architectures (MTA), which allow expanding and modeling efficiently large data management structures, such as databases.

Nevertheless, from a software engineering standpoint, the need for flexible, context-aware and dynamic Multi-Tenant Architectures is gaining momentum, particularly because when dealing with Business Information Systems BIS) scenario requirements, such as Customer Relationship Management (CRM) or Enterprise Resource Planning (ERP) systems, which require the capability for extending Data Models inside the MTA and also provide more knowledge-oriented metadata-driven approaches such as inference, where data relationships can be entailed.

In this paper, we present CLOUDIO, a Cloud Computing-based flexible, Data Model extended and dynamic Multi-Tenant Architecture which addresses the aforementioned topics by supporting it with a very pragmatic and implementation-driven details. For that, we have implemented a proof-of-concept of CLOUDIO using both J2EE and JDBC technologies.

II. CONCEPTUAL MODEL

A multi-tenant architecture allows managing different kind of users of a system in a very flexible way due to the great amounts of configurations that can be adopted easily. For this reason, it is necessary to establish the difference between what is a tenant and a user.

A tenant is considered the owner or the supplier of a SaaS application. He is responsible of the management, maintenance and update of the application and should ensure its availability and security.

The other aspect that provides the advantages and power to multi-tenant architectures is the database model. An appropriate model for the database can take an important increase in the scalability of the system. For achieving that it is possible to find different configurations, each one with its domain of applications, powers and limitations: Separate databases, Shared database, separate schema, Shared database, shared schema. To choose the best option in each case it is necessary to study the case in which the configurations are going to be used and try to make the most of them in terms of efficiency and cost.

Now we are going to provide a brief description of each one to get a better understanding of the matter that it has been treating.

In our approach we have chosen shared database, shared schema. This allows using only the stored space that is necessary in each case and avoiding the problems explained before with lower costs

Here, a metadata table stores important information about every custom field defined by every tenant, including the field´s name (label) and data type.

Firstly, the particular ID data of the associated record in the primary data table is vital for the approach. Secondly, the extension ID associated with the correct custom field definition and finally, the value of the custom field in the record which is being saved and eventually transformed into a string. This approach allows each tenant to create as many custom fields as necessary to meet its business needs.

III. ARCHITECTURE

The architecture of this system has been designed to offer the best flexibility and usability for the multi-tenancy database. The major contribution of this paper is the two levels of multi-tenancy that this architecture provides. The first level of multi-tenancy is due to different tenants are assigned to different databases (Different Instances), and the second one is provoked for the ability of the tenants to share the tables of the database (Shared Schema). It has been possible adding two innovative elements to the typical architecture: the "JDBC Driver Manager" and the "Multi-Tenant DB Index", which together make the transition from single-tenant to multi-tenant almost automatic in a very flexible way. These two artifacts make this architecture different from a regular multi-tenant application working in the Amazon or another cloud.

The SaaS application is in the top of the architecture and it is a J2EE application that needs to store information in a database. Actually, it does not need to be a "SaaS" application but this paper is focused on SaaS applications because multi-tenancy is a key feature in the Cloud Computing world.

The JDBC Driver Manager is just above the predefined JDBC driver and allows, as it was explained, a multi-tenant database to be created automatically with no effort in the tenant side. In the same layer there is the Multi-Tenant DB Index that contains the data of which database is the correspondent for each tenant, therefore it is implemented as a hash map of key->value pairs, being the key the tenant id, and the value the connection parameters and the subjacent JDBC driver to use.

Under the JDBC Driver Manager it is located the JDBC Driver that really connects to the database, any regular driver works. The cloud infrastructure is in charge of balancing the amount of work of every database to keep them working in a high-performance status.

In the bottom down there are the databases. The data of a tenant exists only in one database, although that database can be replicated as many times as needed. This means that the queries against the database thankful to the Multi-Tenant DB Index will consult in just one database.

IV. IMPLEMENTATION

The implementation consists of two parts: the necessary modifications to the database schema, and the driver implementation.

A. Database Modification

First, a new table is created to store tenant data, with a single field, the tenant id. This table is not strictly needed, but it allows the user to associate data with tenants (e.g: tenant name) and to maintain data consistency in the database, forcing every data to be owned by one of the tenants on this table.

The other bit of data that is added to the database is a new column, storing the identifier of the data owner, in every existing table. This new column allows to yielding loosely decoupled virtual tables of each tenant, so they can only access the data rows that are really owned by them. By adding a reference from these columns to the tenants table, we achieve consistency, in the sense that invalid tenants cannot be specified as owners of database rows. Application developers are encouraged to add any information related to the tenant to this table, and the driver will not be affected.

These modifications to the database schema are automated with a Java program that connects to the specified database, create a tenant table with a single tenant_id field, and then list all the original tables and add the owner column

B. Driver Implementation

The actual driver is implemented as a JDBC driver, so any Java application can use it without significant modifications.

When a Java program requests a new connection, the driver starts a real connection to the database using a second database driver. The back-end driver to use, as well as its connection parameters, depends on the tenant that is requesting the connection to the database, which is passed from the application to the front-end JDBC driver. This approach allows the application developer to use any method he sees appropriate to determine the tenant that is connecting to the database.

The tenant identifier is looked-up in a hash table, which associates it with a parameter set that allows to opening a JDBC connection. Once a connection is established, the application can begin sending queries. Each query will then be analyzed, and then rewritten to include the extra data needed to handle the multi-tenancy schema.

This re-implementation of the queries implements the second level of Multi-Tenancy, where the data is separated into virtual tables while sharing the same tables of the same database instance. This query is then send to the database through the back-end driver, and the results are returned verbatim to the application, which can use them as usual.

V. CONCLUSIONS

In this paper, we have presented the CLOUDIO software platform a new approach for Multi-Tenant Architectures supporting the SaaS advanced software delivery approach, which is rooted on the ever-growing use of Cloud Computing. CLOUDIO builds on a set of Business Information Systems scenario requirements, mostly concerned with the flexibility of the Data Model and managing large datasets, optimizing the underlying architecture. We have focused on providing a result-oriented evaluation where we can proof the forthcomings of using our J2EE-based implemented solution.

ACKNOWLEDGMENT

This work is supported by the Spanish Ministry of Industry, Tourism, and Commerce under the project GODO2 (TSI- 020100-2008-564), SONAR2 (TSI-020100-2008-665), and SITIO (TSI-0204000-2009-148), under the PIBES (TEC2006-12365-C02-01) and MID-CBR (TIN2006-15140-C03-02) projects of the Spanish Committee of Education & Science.