Capítulo

2

Software as a Service: Desenvolvendo Aplicações Multi-tenancy com Alto Grau de Reuso

Josino Rodrigues Neto, Vinicius Cardoso Garcia, Andrêza Leite de Alencar, Júlio César Damasceno, Rodrigo Elia Assad e Fernando Trinta

Abstract

Software as a Service (SaaS) represents a new paradigm and business model with witch companies don't need buy and maintain their own IT infrastructure. Instead, they acquire a software as service of third party, obtaining significant benefits, especially regarding the reduction of infrastructure maintenance costs. The aim of this work is to present the main concepts related to multi-tenancy architecture, and an approach to implementation of Software as a Service. During this work the key technologies will be presented associated with the subject and a practical example of implementing a multi-tenancy application using Grails Framework and reusable components.

Resumo

Software como serviço (SaaS) representa um novo paradigma e um modelo de negócios onde as empresas não precisam comprar e manter sua própria infraestrutura de TI. Ao invés disso, elas adquirem um serviço de software de terceiros, obtendo assim consideráveis benefícios, principalmente no que tange a redução de custos de manutenção dessa infraestrutura. O objetivo desse minicurso é apresentar os principais conceitos relacionados à arquitetura multi-tenancy, uma das abordagem para implementação de Software como Serviço. Durante esse trabalho serão apresentadas as principais tecnologias associadas ao assunto e um exemplo prático da implementação de um aplicativo multi-tenancy utilizando o framework Grails e componentes reutilizáveis.

2.1. Introdução

Em 1969, Leonard Keinrock, um dos cientistas chefe da ARPANET, precursora da internet, disse: “A partir de agora, redes de computadores estão ainda na sua infância, mas a medida que crescem e tornam-se sofisticadas, nós iremos provavelmente ver a disseminação do ‘computador utilitário’ que, como telefones e energia elétrica, irão servir casas e escritórios por todo país” [30]. Essa visão de computação utilitária baseada no modelo de fornecimento de serviços antecipa a transformação massiva de toda a indústria de computação nos últimos anos. Serviços de computação estarão prontamente disponíveis sob demanda, como os outros serviços utilitários disponíveis atualmente como água, energia e telefone. Similarmente, os usuários (consumidores) precisam pagar os provedores somente quando eles acessam os serviços de computação. Além disso, consumidores não precisam mais realizar altos investimentos ou enfrentar a dificuldade de construir e manter complexas infra-estruturas de TI para instalar um aplicativo em suas dependências.

Os profissionais de desenvolvimento de software estão enfrentando inúmeros novos desafios com o objetivo de criar serviços que atendam a milhões de consumidores ao invés de prover um software que seja executado em computadores pessoais. E ao longo dos anos, o surgimento de avanços tecnológicos como processadores multicore e ambientes de computação em rede como Cluster Computing [62], Grid Computing [25], computação P2P [57] e o mais recente Cloud Computing [42], tornam esse objetivo cada vez mais factível.

Os serviços de computação citados anteriormente precisam ser altamente confiáveis, escaláveis e dinâmicos para suportar acesso ubíquo e fornecer a possibilidade de composição de outros serviços [15]. Além disso, consumidores devem ter a capacidade de determinar o nível de serviço requerido através de parâmetros de QoS (Quality of Service) e SLA (Service Level Agreement) [7].

Cloud Computing foi o último desses paradigmas a emergir e promete entregar serviços confiáveis através da nova geração de datacenters que são construídos utilizando tecnologias de virtualização de processamento e armazenamento. Consumidores estarão habilitados a acessar aplicações e dados da “cloud” em qualquer lugar do mundo e sob demanda, como é o caso do Gmail[5], Google Docs[6], Office Web Apps[7], Dropbox[8], dentre outros.

[5] http://www.gmail.com
[6] http://docs.google.com
[7] http://office.microsoft.com/web-apps
[8] http://dropbox.com

2.2. Conceitos Fundamentais

2.2.1. Cloud Computing

Atualmente não existe uma definição oficial do que seja *Cloud Computing*. Nesse trabalho usaremos a definição proposta pelo National Institute of Standards and Technology(NIST), que define Cloud Computing como um modelo que permite, de forma conveniente, o acesso a um pool de recursos computacionais compartilhados (rede, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados e liberados com o mínimo de esforço de gerenciamento ou interação do provedor de serviço.

Ainda segundo o NIST, *Cloud Computing* é composta por cinco características essenciais [42]:

- **Alocação de recursos sob demanda:** O usuário pode adquirir unilateralmente recursos computacionais, como tempo de processamento do servidor ou armazenamento, através da rede na medida em que necessite e sem precisar de interação humana com os provedores de cada serviço;

- **Amplo acesso a rede:** recursos estão disponíveis através da rede e podem ser acessados por meio de mecanismos que funcionem em plataformas heterogêneas (por exemplo, telefones celulares, laptops e PDA);

- ***Pooling* de recursos:** os recursos do provedor de computação são agrupados para atender vários consumidores através de um modelo *multi-tenancy*, com diferentes recursos físicos e virtuais atribuídos dinamicamente e novamente de acordo com a demanda do consumidor. Há um senso de independência local em que o cliente geralmente não tem nenhum controle ou conhecimento sobre a localização exata dos recursos disponibilizados, mas pode ser capaz de especificar o local em um nível maior de abstração (por exemplo, país, estado ou data center). Exemplos de recursos incluem o armazenamento, processamento, memória, largura de banda de rede e máquinas virtuais;

- **Elasticidade rápida:** recursos podem ser adquiridos de forma rápida e elástica, em alguns casos automaticamente, caso haja a necessidade de escalar com o aumento da demanda, e podem também ser liberados, na retração dessa demanda. Para os usuários, os recursos disponíveis para uso parecem ser ilimitados e podem ser adquiridos em qualquer quantidade e a qualquer momento; e

- **Serviço medido:** sistemas em nuvem automaticamente controlam e otimizam a utilização dos recursos, alavancando a capacidade de medição em algum nível de abstração adequado para o tipo de serviço (por exemplo, armazenamento,

processamento, largura de banda, e contas de usuários ativos). Uso de recursos pode ser monitorado, controlado e relatado a existência de transparência para o fornecedor e o consumidor do serviço utilizado.

Ainda segundo o NIST [42], Cloud Computing é dividido em três modelos de serviço (Figura 2.1):

• **Software como Serviço (SaaS):** A capacidade fornecida ao consumidor é a de usar as aplicações do fornecedor em uma infraestrutura da nuvem. As aplicações são acessíveis de vários dispositivos cliente através de uma interface thin client, como por exemplo um browser web. O consumidor não administra ou controla a infraestrutura básica, incluindo rede, servidores, sistemas operacionais, armazenamento, ou mesmo capacidades individuais da aplicação. Mesmo assim ainda é possível definir algumas configurações específicas para o usuário na aplicação;

• **Plataforma como Serviço (PaaS):** A capacidade fornecida ao consumidor é a de realizar deploy de uma aplicação em uma infraestrutura pré-definida ou adquirir aplicações criadas usando linguagens de programação e as ferramentas suportadas pelo provedor de PaaS. O consumidor não administra ou controla a infraestrutura básica como rede, servidores, sistemas operacionais, ou armazenamento, mas tem controle sobre os aplicativos utilizados e, eventualmente, hospedagem de aplicativos e configurações de ambiente; e

• **Infraestrutura como Serviço (IaaS) :** A capacidade prevista para o consumidor é a de processamento, armazenamento, redes e outros recursos computacionais fundamentais para que o consumidor seja capaz de implantar e executar programas arbitrários. Esses recursos podem incluir sistemas operacionais e aplicativos. O consumidor não administra ou controla a infraestrutura de nuvem subjacente, mas tem controle sobre os sistemas operacionais, armazenamento, aplicativos implantados, e, eventualmente, o controle limitado de componentes de rede (por exemplo, firewall).

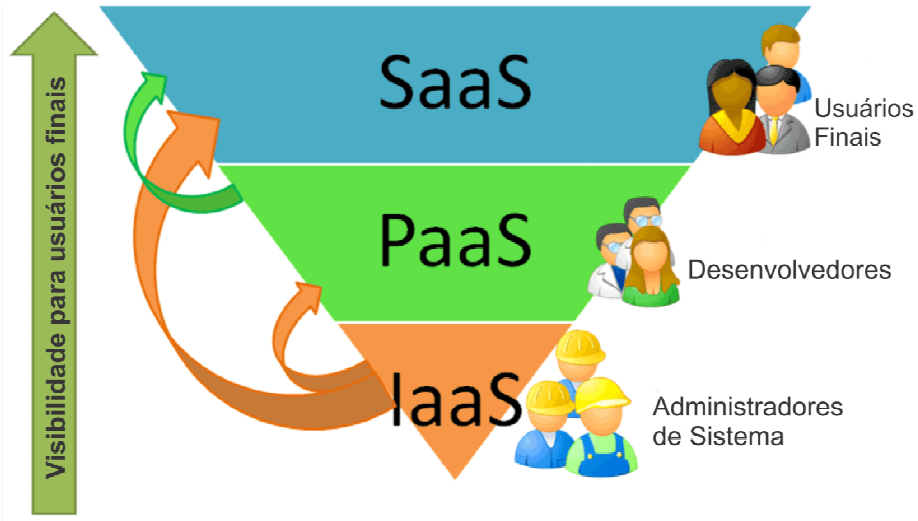


Figura 2.1. Modelos de serviço de *Cloud Computing*

Nesse trabalho teremos como foco principal os tópicos relacionados a SaaS, embora sejam mencionados alguns conceitos associados à PaaS e IaaS. O motivo disso é que multitenancy é um conceito existente dentro do contexto de Software como serviço.

2.2.2. Software como Serviço (SaaS)

Nos últimos anos muitas empresas têm saído do modelo de entrega de software empacotado para o modelo de fornecimento de software na web [23]. Essas aplicações entregues através da web vão desde emails a calendários, sistemas colaborativos, publicações online, processadores de texto simples, aplicações para negócios e aplicações para uso pessoal.

Salesforce.com [35], por exemplo, criou um aplicativo para CRM (*Customer Relationship Management*) e o configurou não como um software empacotado, mas como software rodando sobre servidores acessível através do browser. Quando fez isso, criou a própria plataforma in-house para entregar o software como serviço a seus consumidores.

Logo em seguida Salesforce.com criou o *AppExchange*, uma plataforma de integração aberta para outras empresas de software construírem produtos utilizando algumas funcionalidades do CRM do *salesforce*. Pouco tempo depois *Salesforce* estendeu o conceito de plataforma aberta como force.com, um ambiente de desenvolvimento e deployment usando a infraestrutura de SaaS do *Salesforce*. Posteriormente, algumas outras empresas ingressaram nesse mercado, a Amazon com o EC2 [2] e Google com o Google AppEngine[9], abrindo suas infraestruturas de cloud para hospedarem aplicações de terceiros, além de produzirem serviços online.

Amazon, em especial, vem se tornando bastante atraente porque possui uma rica infraestrutura para suportar operações de varejo online e tem oferecido vários serviços para usuários de cloud como: *data storage*, processamento, fila de mensagens, *billing* e etc. O artigo *"Amazon S3's Amazing Growth"* [24] menciona o crescimento vertiginoso no uso do serviço S3 da Amazon [2] nos últimos anos, é possível ver o gráfico de crescimento desse serviço na Figura 2.2.

Cloud Computing e SaaS também parecem ser eficientes tanto para usuários quanto para fornecedores de software. Vários consumidores podem usar a mesma instalação do software e consequentemente isso melhora as taxas de utilização de hardware e rede. Por exemplo, Amazon[10] e Google[11] têm enormes *datacenters* que eles não utilizam completamente o tempo todo. Eles podem executar seus próprios produtos enquanto hospedam aplicações de outras empresas. Empresas como Amazon, Google e

[9] https://appengine.google.com
[10] http://aws.amazon.com
[11] http://google.com

Salesforce geralmente garantem a qualidade de serviço para todos os seus consumidores de cloud através de SLAs (*Service Level Agreements*) detalhadas.

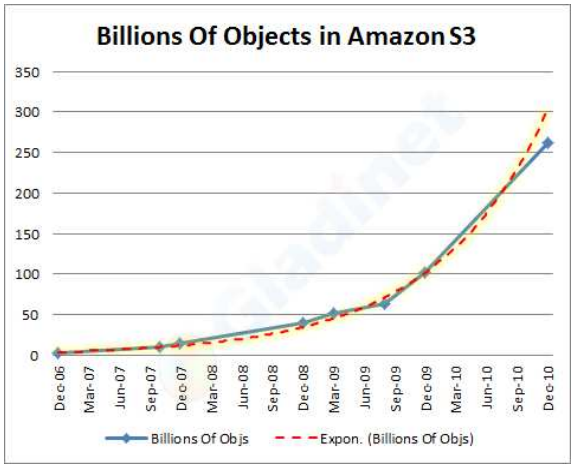


Figura 2.2. Amazon S3's Growth (Fonte: [24])

Os serviços providos por esses grandes players do mercado permitem que empresas possam desenvolver software no modelo SaaS e hospedar em algum de seus datacenters pagando apenas pelo que usarem. Isso torna-se um atrativo principalmente para Startups e empresas com poucos recursos financeiros.

Segundo Harris et al.[21] existem três tipos diferentes de aplicações SaaS:

- **Single-instance:** as aplicações proveem serviços para um único cliente e executam em um servidor exclusivo. Nesse caso cada servidor web possui uma única instância da aplicação. Essa pode considerada a abordagem com maior desperdício de recursos, dado que um servidor pode, por exemplo, executar com apenas 10% de sua capacidade;

- **Multi-instance:** essas aplicações executam em ambientes onde o servidor web é compartilhado. Nessa abordagem uma cópia da aplicação é inicialmente configurada para cada cliente, e então cada cópia é implantada como um contexto no mesmo servidor web; e

- **Multi-tenancy:** provê uma única aplicação compartilhada por vários clientes. Nessa abordagem várias aplicações "virtuais" são criadas na mesma instância.

Implementar o conceito de SaaS nem sempre é tão simples como parece. Chong[12] propõe 4 níveis de maturidade para aplicações que utilizam o modelo de SaaS:

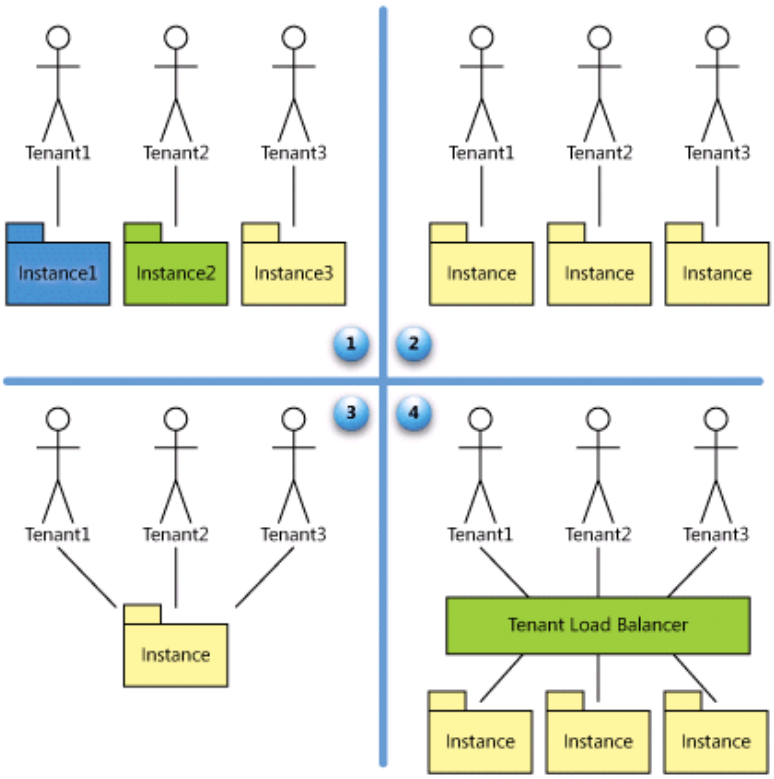


Figura 2.3. Níveis de Maturidade SaaS (Fonte: [12])

- **Nível 1 Ad-Hoc/Personalizado:** O primeiro nível de maturidade é semelhante ao modelo de entrega de software do provedor de serviços de aplicativos (ASP Application Service Provider) tradicional, que data da década de 1990. Nesse nível, cada cliente tem a sua própria versão personalizada do aplicativo hospedado e executando nos servidores do provedor. Pensando em arquitetura, software nesse nível de maturidade é muito semelhante aos softwares corporativos vendidos tradicionalmente, em que diferentes usuários de uma organização conectam a uma instância quaisquer outras instâncias ou processos que o host esteja executando para os seus outros clientes;

- **Nível 2 Configurável:** No segundo nível de maturidade, o fornecedor hospeda uma instância separada do aplicativo para cada tenant. Enquanto no primeiro nível cada instância é personalizada individualmente para o tenant, neste nível todas as instâncias utilizam a mesma implementação de código e o fornecedor atende as necessidades dos clientes fornecendo opções de configuração detalhadas que permitem ao cliente alterar a aparência e o comportamento do aplicativo para os seus usuários. Apesar de serem idênticas a nível do código, cada instância permanece totalmente isolada de todas as demais;

- **Nível 3 Configurável e eficiente para vários tenants:** No terceiro nível de maturidade, o fornecedor executa uma única instância que serve a todos os clientes. Metadados configuráveis são usados para fornecer uma experiência de usuário e um conjunto de recursos exclusivos para cada instância. Políticas de

autorização e de segurança garantem que os dados de cada cliente sejam mantidos separados dos dados de outros clientes e que, da perspectiva do usuário final, não exista qualquer indicação de que a instância do aplicativo esteja sendo compartilhada entre vários tenants; e

- **Nível 4 Escalonável, configurável e eficiente para vários tenants:** No quarto e último nível de maturidade, o fornecedor hospeda vários clientes em um ambiente com balanceamento de carga. Os dados de cada cliente são mantidos separados e com metadados configuráveis fornecendo uma experiência do usuário e um conjunto de recursos exclusivos para cada cliente. Um sistema de SaaS é escalonável para um número de clientes arbitrariamente grande, uma vez que a quantidade de servidores e instâncias no lado do fornecedor pode ser aumentada ou diminuída conforme necessário para corresponder à demanda sem a necessidade de remodelar a arquitetura aplicativo, além disso as alterações ou correções podem ser transmitidas para milhares de *tenants* tão facilmente quanto para um único *tenant*.

Normalmente se esperaria que o quarto nível fosse a meta definitiva para qualquer aplicativo de SaaS, mas não é sempre assim. É necessário verificar as necessidades operacionais, arquiteturais e de negócio relacionadas à aplicação. Uma abordagem *singletenant* faz sentido financeiramente? O seu aplicativo pode ser feito para executar em uma única instância lógica? Você pode garantir que a qualidade de serviço desejada por cada cliente seja atendida? Essas são questões que devem ser respondidas quando se pretende adotar o modelo SaaS.

2.2.3. Multi-tenancy

Multi-tenancy é uma abordagem organizacional para aplicações SaaS. Bezemer e Zaidman [8] definem multi-tenancy como aplicações que permitem a otimização no uso dos recursos de hardware, através do compartilhamento de instâncias da aplicação e da instância do banco de dados, enquanto permite configurar a aplicação para atender às necessidades do cliente como se estivesse executando em um ambiente dedicado. Tenant é uma entidade organizacional que aluga uma aplicação multi-tenancy. Normalmente, um tenant agrupa um número de usuários que são os stakeholders da organização.

A definição anterior foca no que nós consideramos aspectos importantes em aplicações multi-tenancy:

- Possibilidade de compartilhamento de recursos de hardware, permitindo a redução de custos [58];

- Alto grau de configurabilidade, permitindo que cada consumidor customize sua própria interface e seu workflow na aplicação [47, 26]; e

- Uma abordagem arquitetural na qual os tenants fazem uso de uma única aplicação e banco de dados [34].

É possível que algumas pessoas confundam multi-tenancy com o conceito de multi-usuário. Multi-tenancy não é multi-usuário. Em uma aplicação multi-usuário nós assumimos que os usuários estão usando a mesma aplicação com opções de acesso limitadas e que essa instância da aplicação é utilizada por apenas um único cliente. Nesse caso podemos ter vários usuários com o perfil "gerente", com o perfil "vendedor", ou com qualquer perfil de usuário necessário para o funcionamento da aplicação. Em uma aplicação multi-tenancy nós assumimos que existe uma instância da aplicação que atende a vários clientes (tenants) e possui um alto grau de configuração. Dependendo da definição dessas configurações, dois tenants podem possuir aparência e workflows diferentes. Um argumento adicional para essa distinção é que o SLA para cada tenant pode ser diferente [39].

Uma abordagem mais profunda sobre o tema multi-tenancy será apresentado na seção seguinte.

2.3. Propostas de Arquitetura Multi-tenancy

Um ponto importante durante o desenvolvimento de software em qualquer segmento é conhecer implementações semelhantes já realizadas por outros desenvolvedores. Partindo desse princípio essa seção apresenta propostas de arquitetura de software que auxiliem no desenvolvimento de aplicações multi-tenancy. Foram encontradas propostas de arquitetura para aplicações multi-tenancy [31, 9, 8, 49, 56, 63, 28, 11, 32] e para plataformas de suporte a multi-tenancy [60, 48, 5].

Dentre essas propostas algumas já foram aplicadas à indústria, como é o caso do Force.com [60] e EXACT [9]. Force.com é uma plataforma de desenvolvimento de software que utiliza arquitetura dirigida à metadados para construção de aplicações multitenancy. Nessa arquitetura tudo que é exposto para os desenvolvedores e usuários da aplicação é internamente representado como metadado. Formulários, relatórios, workflows, priviégios de acesso, customizações específicas do tenant e lógica de negócio, tudo é armazenado como metadados. Outro exemplo de arquitetura dirigida à metadados pode ser encontrada em [48]. Bezemer et al. [9] apresentam uma arquitetura utilizada no desenvolvimento de aplicações na empresa EXACT, uma empresa de desenvolvimento de software especializada em ERP, CRM e aplicações financeiras. Em sua arquitetura os autores apresentam 3 componentes básicos para a implementação de aplicações multitenancy: componente de autenticação, customização e banco de dados. Além disso os autores apresentam um estudo de caso e uma lista de lições aprendidas.

Outros autores apresentam Arquiteturas Orientadas a Serviço (SOA ServiceOriented Architecture) que implementam requisitos de multi-tenancy como é o caso de Jing and Zhang [28],Azeez et al. [5] e Pervez et al. [49]. Jing and Zhang [28] apresentam OSaaS, uma arquitetura que utiliza tecnologias SOA para desenvolvimento de aplicações SaaS. Azeez et al. [5] apresentam uma arquitetura para implementar

multi-tenancy a nível de SOA, que permite a usuários executar seus serviços e outros artefatos SOA em um framework multi-tenancy SOA. Já Pervez et al. [49] apresentam uma arquitetura para SaaS que foca nos aspectos de segurança e balanceamento de carga em ambiente de cloud computing.

Além das propostas de arquitetura citadas anteriormente, foi encontrado também uma proposta de estilo arquitetural, o SPOSAD (*Shared, Polymorphic, Scalable Application and Data*), que é descrito em [31] e [32]. Um estilo arquitetural define os tipos de elementos que podem aparecer em uma arquitetura e as regras que regem a sua interconexão. O SPOSAD descreve componentes, conectores e elementos de dados de uma arquitetura multi-tenancy, bem como restrições impostas nesses elementos. Os autores também apresentam os benefícios do uso dessa arquitetura e informações que podem auxiliar em decisões de projeto.

Tsai et al. [56] apresentam uma arquitetura de duas camadas que foca em escalabilidade, que trabalha a nível de serviço e aplicação para economizar o uso de recursos, e a idéia chave é aumentar os recursos somente onde houver gargalos. Várias técnicas de duplicação de recursos computacionais são propostas, incluindo duplicação preguiçosa e duplicação pro-ativa para alcançar a melhor performance do sistema. Além da arquitetura esse trabalho apresenta um algoritmo para alocação de recursos para cluster em ambientes de cloud. Já Yuanyuan et al. [63] apresentam uma arquitetura multi-tenancy para sistemas de suporte a negócios (BSS Business Support System) e discutem brevemente uma abordagem para alcançar configurabilidade, seguranca e escalabilidade na arquitetura. Focando em escalabilidade de dados, os autores propõem um particionamento horizontal baseado em grupos de entidades pela análise das relações entre as entidade de negócio do sistema.

Calero et al. [10] apresentam a arquitetura de um sistema de autorização multitenancy apropriado para um serviço de middleware para PaaS. Cada empresa pode provê vários serviços de cloud que podem colaborar com outros serviços tanto da mesma organização quanto de organizações diferentes. Esse sistema de autorização suporta acordos de colaboração entre empresas (também conhecidos como federações).

2.4. Componentes Básicos de uma Aplicação Multi-tenancy

Multi-tenancy afeta quase todas as camadas de uma aplicação típica e possui um grande potencial para ser implementado como interesse transversal [43]. Para reduzir a complexidade do código, a implementação de requisitos multi-tenancy deve ser separada da lógica de negócio o máximo possível. Caso contrário, a manutenção pode se tornar um pesadelo, porque:

- Implementar código de requisitos da arquitetura multi-tenancy juntamente com a lógica de negócio dos tenants, exige que todos os desenvolvedores sejam reeducados para entenderem os conceitos de multi-tenancy; e

- Misturar multi-tenancy com código de lógica de negócio dos tenants leva ao aumento da complexidade da implementação, pois é mais difícil manter o controle de onde o código multi-tenancy é introduzido.

Estes problemas podem ser superados integrando cuidadosamente multi-tenancy na arquitetura. No restante desta seção, descrevemos os componentes da arquitetura abordada por Bezemer et al. [9] para implementação de multi-tenancy como um interesse transversal. A Figura 2.4 apresenta a descrição dessa aplicação e as subseções seguintes descrevem cada componente da aplicação.

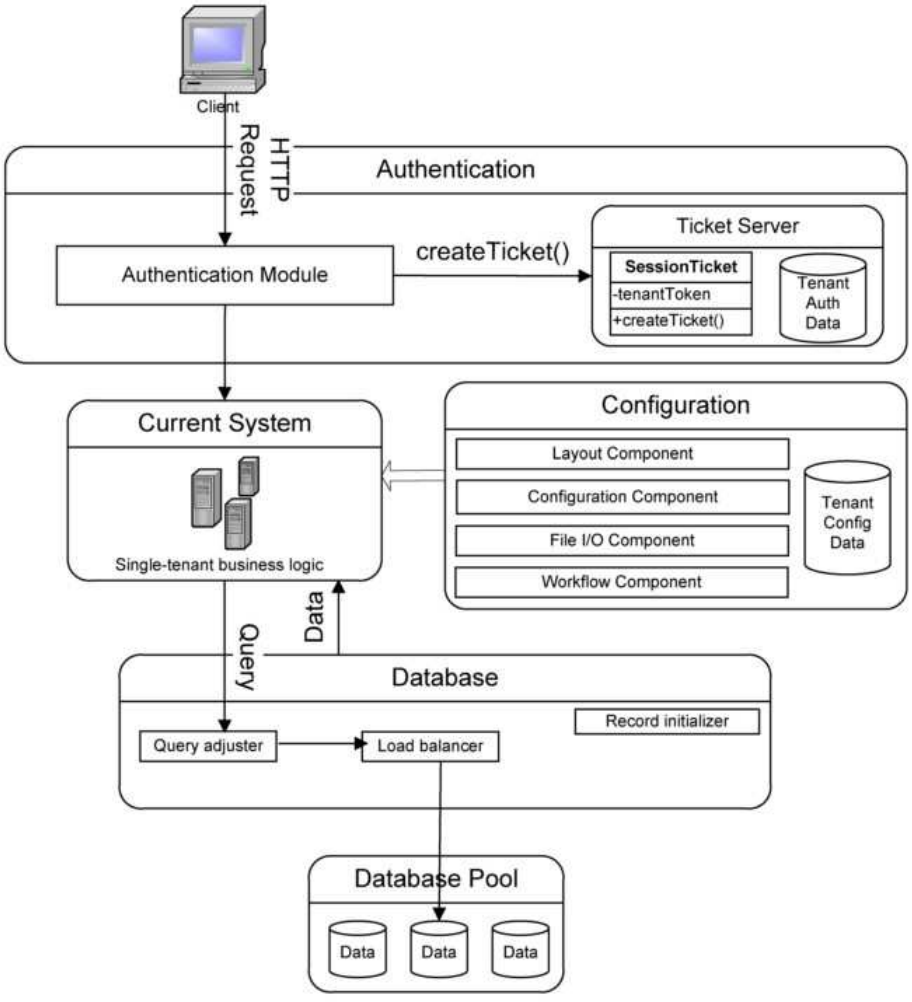


Figura 2.4 Arquitetura de Referência adotada (Fonte: [9])

2.4.1. Autenticação

Devido a uma aplicação multi-tenancy ter apenas uma instância da aplicação e do banco de dados, todos os tenants usam o mesmo ambiente físico. A fim de oferecer a customização do ambiente e ter certeza de que os tenants podem acessar somente os seus próprios dados, tenants devem ser autenticados. Enquanto autenticação de usuário é, possivelmente, já presente na aplicação de destino, um mecanismo separado de

autenticação de tenants específicos pode ser necessário, por duas razões: (1) geralmente é muito mais fácil introduzir um mecanismo de autenticação adicional do que mudar um já existente, e (2) autenticação de tenants permite que um único usuário faça parte de mais do que uma organização lógica, o que estende a idéia de autenticação de usuários com “grupos”.

2.4.2. Configuração

Em uma aplicação multi-tenancy a customização deve ser possível através de configuração. A fim de permitir que o usuário tenha uma experiência como se ele estivesse trabalhando em um ambiente dedicado, é necessário permitir pelo menos os seguintes tipos de configuração:

- **Estilo de Layout (Layout Style):** O componente de configuração de estilo de layout permite o uso de temas e estilos específicos;
- **Configuração Geral (General Configuration):** O componente de configuração geral permite a especificação de configurações específicas, como configurações de chave de criptografia e detalhes do perfil pessoal;
- **Entrada e saída de arquivo (File I/O):** O componente de configuração de I/O de arquivo permite a especificação de caminhos de arquivos, que podem ser usados para, por exemplo, geração de relatório; e
- **Fluxo de trabalho (Workflow):** O componente de configuração de fluxo de trabalho permite a configuração de fluxos específicos. Por exemplo, configuração de fluxos é necessária em uma aplicação de planejamento de recursos empresariais ERP, em que os passos para se realizar uma mesma tarefa pode variar significativamente entre diferentes companhias.

2.4.3. Banco de dados (Database)

Em uma aplicação multi-tenancy há uma grande exigência pelo isolamento dos dados. Pelo fato de os tenants usarem a mesma instância de um banco de dados é necessário ter certeza de que eles podem acessar somente seus próprios dados. Atualmente sistemas de gerenciamento de dados (Data Base Management Systems DBMS) de prateleira não são capazes de lidar com multi-tenancy de forma nativa, isso deve ser feito em uma camada entre a camada lógica de negócios e o pool de banco de dados da aplicação. As principais tarefas dessa camada são as seguintes:

- **Criação de novos tenants no banco de dados:** Se a aplicação armazena ou recupera dados que podem ser de tenants específicos, é tarefa da camada de banco de dados criar os registros do banco de dados correspondente quando um novo tenant se inscreveu para a aplicação;

- **Adaptação de consulta:** A fim de prover um isolamento de dados adequado, a camada de banco de dados deve ter certeza que consultas são ajustadas de forma que cada tenant possa acessar somente seus próprios registros; e

- **Balanceamento de carga:** Para melhorar o desempenho de uma aplicação multi-tenancy é necessário um balanceamento de carga eficiente para o pool de banco de dados. Note que qualquer acordo feito no SLA de um tenant e quaisquer restrições impostas pela legislação do país onde o tenant está localizado deve ser satisfeita. Por outro lado, nossa expectativa é a de que é possível criar algoritmos de balanceamento de carga mais eficientes usando as informações coletadas sobre as características de funcionamento dos tenants.

2.5. Implementando um Protótipo de Aplicação Multi-tenancy

2.5.1. Tecnologias

Após a escolha de uma arquitetura de referência para ser adotada por nossa aplicação, tem-se a necessidade de escolher as tecnologias que serão adotadas para a implementação. Durante a escolha dessas tecnologias avaliou-se a possibilidade do uso de JSF (Java Server Faces)[12], Struts 2[13], Spring Framework[14] e Grails[15]. Para a implementação do protótipo descrito nesse trabalho optou-se por utilizar o Grails Framework, pois apresenta um conjunto de recursos que podem dar produtividade para a equipe de desenvolvimento como geração de CRUD (Create, Read, Update e Delete); arquitetura baseada em plugins, que permite a criação e reuso de componentes; e total integração com frameworks e APIs Java. É possível desenvolver aplicações multi-tenancy em outras linguagens de programação como PHP, C#, Python, Ruby, etc. O que pode variar de uma linguagem para outra é o esforço necessário para desenvolver esse tipo de aplicação e algumas variações de performance.

Grails é um framework web open-source que utiliza a linguagem Groovy[16], e outros frameworks consagrados como Hibernate[17], Spring Framework e Sitemesh[18]. Uma descrição visual da arquitetura do Grails é apresentada na Figura 2.5. Grails foi projetado para desenvolver aplicações CRUD de forma simples e ágil, utilizando o modelo de “escrever código por convenção” introduzido pelo Ruby on Rails. O Grails propõe trazer a produtividade do Ruby on Rails para a plataforma Java, porém ele possui uma grande vantagem, já que é baseado na linguagem Groovy. Groovy (padronizado pela JSR-241) é uma linguagem dinâmica e ágil para a plataforma Java, que possui muitas características de linguagens de script como Ruby, Python e Smalltalk; e, além disso, aplicações Groovy podem utilizar classes Java facilmente.

[12] http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html
[13] http://struts.apache.org/2.3.1.2/index.html
[14] http://www.springsource.org/spring-framework
[15] http://grails.org
[16] http://groovy.codehaus.org
[17] http://hibernate.org
[18] http://sitemesh.org

Linguagens de script estão ganhando cada vez mais popularidade, devido a quantidade reduzida de código fonte necessário para implementar determinadas funcionalidades, se comparado com uma implementação em Java.

O protótipo multi-tenancy descrito nesse trabalho, em particular, se beneficiará da capacidade de definir métodos e propriedades em tempo de execução, disponibilizado pela linguagem Groovy. Essa característica da linguagem Groovy é chamada de metaprogramação. Em uma linguagem estática como Java, o acesso a uma propriedade ou invocação de um método é resolvido em tempo de compilação. Em comparação, Groovy não resolve o acessos à propriedades ou invocação de métodos até que a aplicação seja executada [50]. Uma aplicação que utiliza essa linguagem pode dinamicamente definir métodos ou propriedades em tempo de execução, isso vai de encontro à necessidade de customização das aplicações, dado que, com a utilização desse recurso, pode-se adicionar atributos e customizar comportamentos de um tenant específico futuramente.

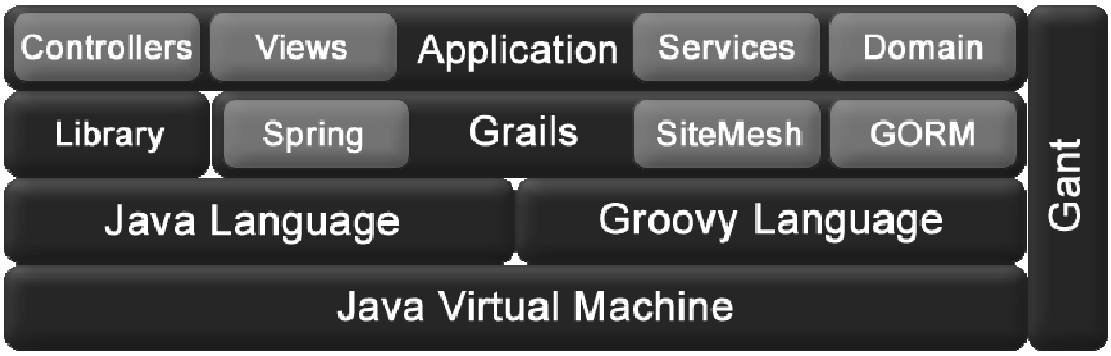


Figura 2.5. Arquitetura do Grails Framework (Fonte: [29])

Outro ponto que influenciou bastante na escolha de Grails foi sua arquitetura baseada em plugins. Grails não se propõe a ter todas as respostas e soluções para o desenvolvimento de aplicações web. Ao invés disso, ele provê uma arquitetura baseada em plugins e um repositório mantido pela comunidade de desenvolvedores onde é possível encontrar plugins que implementam as mais diversas funcionalidades como segurança, teste, busca, geração de relatórios, REST, web services, etc. Essa abordagem proporciona o reuso e permite que funcionalidades de difícil implementação possam ser adicionadas na aplicação de forma bastante simples.

Além das tecnologias relacionadas à programação, foi necessário escolher a tecnologia utilizada para armazenamento de dados. Como já existiam dados legados cadastrados em um banco de dados relacional, decidiu-se adotar o SGBD (Sistema de Gerenciamento de Banco de Dados) PostgreSQL. A escolha desse SGBD deu-se pelo fato do mesmo ser open-source, bastante consolidado no mercado e possuir integração com várias ferramentas de relatórios, o que poderia facilitar futuras extrações de dados para os clientes.

2.5.2. Prototipagem

Para exemplificar uma aplicação real que implementa o conceito de multi-tenancy será descrita a implementação de uma aplicação desenvolvida utilizando Grails Framework. A Arquitetura dessa aplicação é apresentada na Figura 2.6. Em nossa proposta os módulos associado à lógica de negócio da aplicação são implementados o mais independente possível dos requisitos multi-tenancy. Juntamente com o Framework Grails foram adotados alguns plugins existentes em seu repositório. A implementação do protótipo tem o objetivo de apresentar como esses conceitos podem ser aplicados na prática.

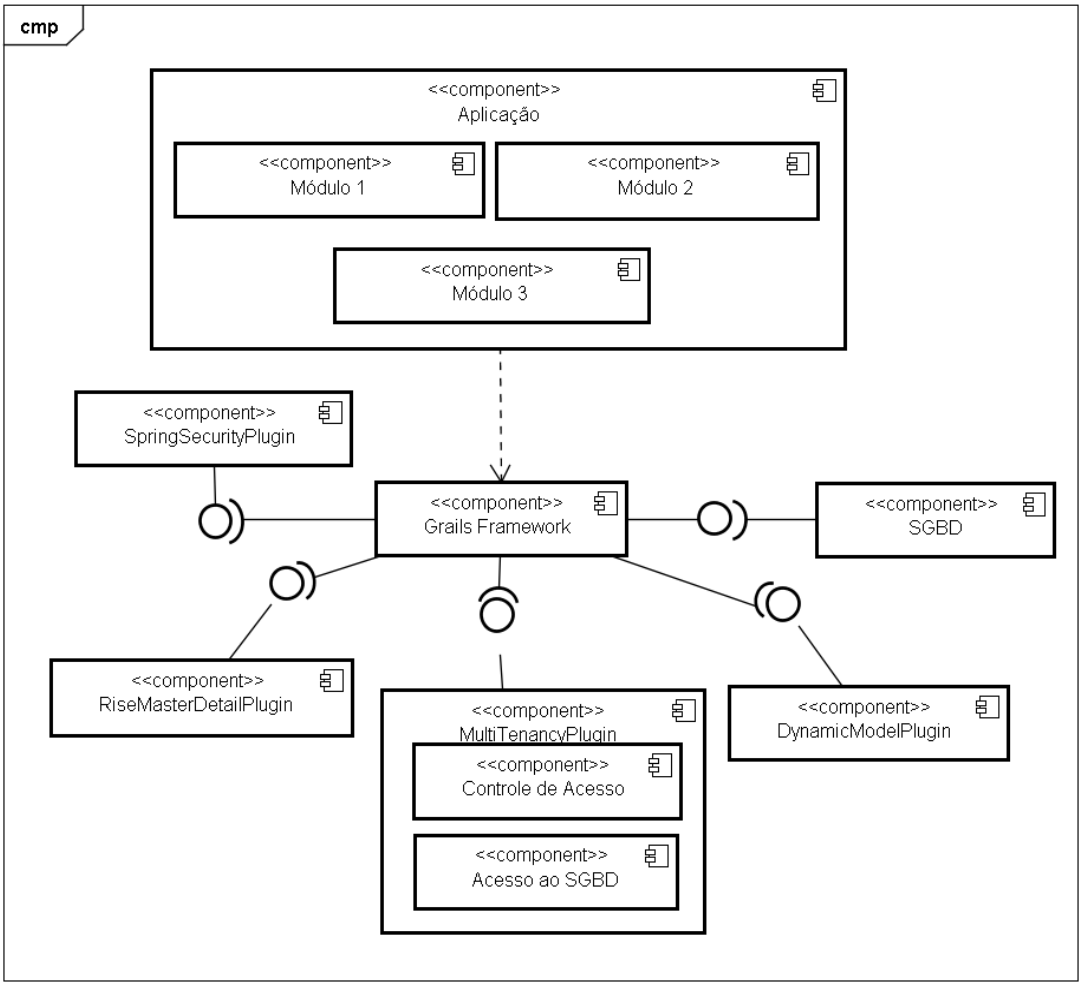


Figura 2.6. Arquitetura da Aplicação

Como mencionado anteriormente, existe uma grande necessidade de separar o código de lógica de negócio do código relacionado aos requisitos multi-tenancy. Com o objetivo de implementar os requisitos multi-tenancy de forma reutilizável, a solução foi implementar esses requisitos como um plugin do Grails, dessa forma aplicações futuras poderiam se beneficiar do código implementado. Uma vantagem do uso dessa abordagem é que dentro de um plugin grails é possível adicionar não só ter classes

implementadas em Groovy ou Java, mas também páginas da camada de visão e outros tipos de arquivo. Antes de implementar os requisitos multi-tenancy como plugin, foi realizado uma pesquisa no repositório de plugins do Grails para verificar se já existia algo semelhante ao que se pretendia implementar.

Durante a pesquisa foi encontrado o plugin *Multi-tenancy Core*[19] que implementa as funcionalidades de dois dos componentes mencionados em nossa arquitetura de referência[9]: o componente de autenticação e o componente de acesso ao banco de dados. Esse plugin implementa a funcionalidade de associar um usuário da aplicação a um tenant específico e além disso realiza de forma dinâmica o filtro dos dados durante uma consulta ao banco de dados. Dessa forma um usuário da aplicação tem acesso apenas aos dados vinculados ao seu tenant. Para utilizar esse plugin é necessário apenas instalá-lo na aplicação e adicionar uma anotação *@MultiTenant* nas classes de domínio da aplicação. Essa anotação indica que todos os objetos dessa classe deverão ser gerenciados pelo plugin e sempre que um registro for salvo no banco deverá ser associado ao tenant do usuário que estiver logado no momento. Um exemplo simplificado de uma classe Groovy com essa anotação é apresentada na Figura 2.7.

```groovy
package br.com.rise.alexandria.fur

import grails.plugin.multitenant.core.groovy.compiler.MultiTenant;

/**
* The Product entity.
*
* @author Josino Rodrigues
*/
@MultiTenant
class Product {

    String name
    String description

    static hasMany = [modules : Module]


    static constraints = {
        project(blank:false)
        name(size:1..100, blank:false, unique:['tenantId','project'])
        description()
    }

    //MÉTODOS ...

}
```

Figura 2.7. Exemplo de classe escrita em linguagem Groovy que utiliza o plugin Multi-Tenancy Core

A autenticação e controle de acesso pode ser realizado através da integração desse plugin com o plugin Spring Security[20], um framework de controle de acesso bastante utilizado em aplicação Java. A Figura 2.8 apresenta a tela de autenticação gerada pelo próprio plugin do Spring Security. De forma integrada com o plugin Multi-tenancy Core, ele associa cada usuário logado ao tenant ao qual ele pertence.

[19] http://multi-tenancy.github.com/grails-multi-tenancy-core
[20] http://static.springsource.org/spring-security/site/index.html

O terceiro componente mencionado em nossa arquitetura de referência é o componente de configuração. O objetivo desse componente é gerenciar as configurações relacionadas a cada tenant e prover funcionalidades de customização da aplicação para estes.

Figura 2.8. Tela de autenticação gerada pelo Spring Security Plugin

Essa customização pode ir desde alterações em interface com o usuário até alteração nas classes de negócio, tudo isso realizado de forma dinâmica. Durante a pesquisa foi encontrado o plugin *Dynamic Domain Class*[21] que proporciona a criação de classes de domínio de forma dinâmica, esse plugin ajudou a validar a idéia de usar Groovy e Grails para customizar as classes de domínio em tempo de execução. A tela apresentada na Figura 2.9 apresenta o formulário utilizado na aplicação para criação de classes dinâmicas. Para cada classe dinâmica criada o Grails Framework cria automaticamente o CRUD para essa entidade.

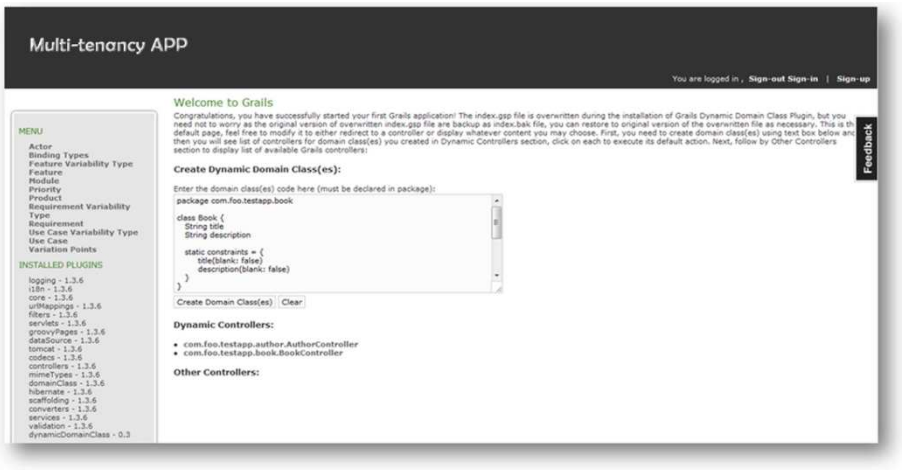


Figura 2.9. Tela com Formulário para criação de classes dinâmicas

Durante a implementação de um projeto real, detectamos que o padrão de geração de telas do Grails não satisfazia aos anseios dos usuários quanto à forma de interação com o aplicativo. Apartir dessa necessidade foi desenvolvido um plugin para implementar o

[21] http://code.google.com/p/grails-dynamic-domain-class-plugin/

padrão de tela Mestre/Detalhe [45], não implementado nativamente pelo gerador de telas padrão do Grails. Durante a implementação desse plugin identificou-se que poderia ser viável implementar módulos completos da aplicação como plugins Grails, desde classes de negócio à camada de visão.

Após a implementação do protótipo foi realizado a avaliação do protótipo para verificar alguns atributos de qualidade atendidos pela proposta. A Tabela 2.1 apresenta para cada atributo de qualidade uma breve descrição de como ele foi atendido.

Tabela 2.1. Resultado da aplicação dos critérios de exclusão (Fonte: Elaboração própria)

ATRIBUTO	AVALIAÇÃO
Disponibilidade	Como foi adotado um SGBD *PostgreSQL*, é possivel agendar tarefas automáticas no servidor para executar *backups* periódicos.
Integridade Conceitual	Tanto as entidades de negócio quanto as classes utilitárias da aplicação foram modularizadas em *plugins* de forma que os interesses da aplicação foram bem definidos e separados.
Flexibilidade	A flexibilidade pode ser alcançada através do uso das característica de linguagem dinâmica existente na linguagem *Groovy*. Dessa forma é possivel criar Entidade de domínio e alterar entidades já existentes em tempo de execução, caso seja necessário.
Interoperabilidade	O uso de *plugins Grails* permite que os dados da aplicação possam ser expostos através de REST ou *Web Services* para os usuários.
Manutenibilidade	A arquitetura de *plugins* do *Grails* facilita a manutenção dos código já que cada plugin pode ser mantido separadamente, reduzindo as dependências entre os componentes da aplicação.
Reusabilidade	As partes da aplicação implementadas como *plugins* podem ser reutilizadas em aplicações futuras facilmente, sem a necessidade de qualquer adaptação no *plugin*.
Segurança	A autenticação e controle de acesso puderam ser garantidos pelo uso do *Framework Spring Security*, que pôde ser facilmente integrado com o *plugin* de *multi-tenancy* no atendimento do requisito de isolamento de acesso entre dados dos *tenants*.
Testabilidade	O *Grails Framework* provê uma infraestrutura para auxiliar na implementação de testes unitários que utiliza *JUnit*, um *framework* de teste para aplicações *Java*. Durante a criação de uma classe o *Grails Framework* já cria uma classe de testes correspondente para otimizar o tempo de desenvolvimento.
Usabilidade	*Grails* trabalha com templates para geração de telas. Embora já possua um template padrão, o desenvolvedor pode implementar um template próprio que atenda à necessidades específicas de interação com o usuário. O template padrão do *Grails Framework* já implementa alguns padrões de usabilidade bem estabelecidos no mercado.

Esse protótipo foi baseado em uma experiência adquirida durante o desenvolvimento de uma aplicação real. A aplicação desenvolvida entrou em produção e atendeu satisfatoriamente aos requisitos do cliente. Durante o desenvolvimento do software em questão foi utilizado Grails porque era, na época, a opção open source mais madura no tocante a multi-tenancy. Atualmente já existem outros frameworks que

dão suporte ao desenvolvimento de aplicações multi-tenancy como Atena Framework[22], Hibernate 4[23], Rails[24], etc.

2.6. Vantagens e desvantagens

Durante uma vasta pesquisa bibliográfica foram identificados 12 trabalhos que citam vantagens ou desvantagens da adoção de multi-tenancy. Apartir da leitura dos mesmos identificamos as seguintes vantagens:

- O provedor de serviço pode usar a mesma versão da aplicação (com o único código base) para prover serviços para várias organizações [55];
- Consumidores obtêm acesso à capacidade de processamento elástico que pode ser aumentada ou diminuída de acordo com a demanda sem a necessidade de realizar manutenção em servidores [55];
- Dois benefícios de uma abordagem baseada em uma plataforma multi-tenancy são colaboração e integração. Pelo fato de todas as aplicações executarem em um único espaço, torna-se mais fácil permitir a um usuário de qualquer aplicação acesso à uma coleção de dados de outra aplicação semelhante. Essa capacidade simplifica o esforço necessário para integrar aplicações relacionadas e os dados que elas gerenciam [60];
- Atualização do software de uma só vez para todos os tenants [44];
- Economia nos custos com infra-estrutura de hardware [53, 33, 4];
- Economia nos custos de gerenciamento de infra-estrutura [53, 33, 4];
- Aumento da margem de lucro para o provedor de serviço através da redução dos custos de entrega e diminuição dos custos de assinatura do serviço para os clientes [36];
- Possibilidade de reusar regras de negócio com o mínimo de adaptação [9];
- Organização dos usuários em vários níveis de acordo com suas necessidades e melhor gerenciamento de recursos [27];
- O usuário pode customizar sob-demanda os serviços providos pelo fornecedor do software [67]; e
- Redução dos custos de venda e manutenção do software por porte do provedor do software [67].

As desvantagens da adoção de multi-tenancy foram pouco mencionadas nos trabalhos encontrados. Em geral os trabalhos mencionavam mais vantagens do que

[22] http://athenasource.org
[23] http://docs.jboss.org/hibernate/orm/4.1/devguide/en-US/html/ch16.html
[24] http://rubyonrails.org

desvantagens. A seguir são listados alguns pontos considerados desvantagens por alguns autores:

- É difícil calcular os recursos requeridos por cada novo tenant e ao mesmo tempo garantir que as restrições de todos os outros tenants da mesma instância sejam atendidas[33];
- Fatores limitantes e gargalos nos recursos computacionais exigidos pelas várias instâncias devem ser identificados [33], e isso não é trivial nesse tipo de ambiente;
- Dificuldade de comparar e otimizar a redução de custos das diferentes formas de distribuição dos tenants entre os servidores, pelo fato de existirem várias variáveis envolvidas [33];
- Preocupação das empresas com o custo inicial de reestruturas suas aplicações legadas para multi-tenancy, [8]; e
- Preocupação dos mantenedores de software com a possibilidade de multi-tenancy introduzir problemas adicionais de manutenção decorrentes do fato desses novos sistemas serem altamente customizáveis [8].

2.7. Desafios da área

2.7.1. Alocação de Recursos

Para que se possa desfrutar dos benefícios que uma aplicação multi-tenancy traz, um conjunto de desafios devem ser solucionados [33]:

1. Calcular os recursos computacionais necessários para o funcionamento de cada novo tenant, e ao mesmo tempo atender às restrições de todos os tenants em instância da aplicação compartilhada;
2. Identificar fatores limitantes ou gargalos nos recursos computacionais requeridos para as várias instâncias, cada uma com vários tenants contendo diferentes restrições que devem ser atendidas;
3. Durante a distribuição dos tenants é necessário indicar a melhor localização para que nenhuma restrição de SLA seja violada;
4. A distribuição dos tenants e instâncias em um ambiente de computação distribuído deve ser automatizada; e
5. A economia alcançada entre as diferentes formas de distribuições de tenants e instâncias deve ser comparada e otimizada, de forma que seja encontrada a melhor distribuição possível.

O cálculo do número máximo de usuários e tenants em uma instância compartilhada em um servidor, sem que haja violação de restrições definidas no

contrato de SLA de cada tenant é um desafio não trivial. Kwok e Mohindra [33] propõem uma abordagem para o cálculo de recursos requeridos para o bom funcionamento de vários tenants em uma instância de aplicação compartilhada. Nesse trabalho também é descrito um método para otimizar a distribuição de tenants e instâncias de uma aplicação em um conjunto de servidores sem violar qualquer requisito de SLA dos tenants. Por fim os autores apresentam uma ferramenta que tem o objetivo de auxiliar o deployment de aplicações multi-tenancy usando o número mínimo de servidores, fornecendo portanto o máximo de economia em um ambiente de computação distribuído.

Fehling et al. [18] também realizam um estudo sobre as oportunidades de otimização do uso de recursos, mas nesse caso o cenário considerado é um ambiente de computação heterogêneo onde os recursos utilizados são oriundos de clouds públicas e privadas. Nesse trabalhos os autores utilizam um algoritmo Smarter Simulated Annealing para auxiliar na busca de uma distribuição otimizada dos recursos computacionais.

Outra questão associada à alocação de recursos é a priorização das requisições recebidas por uma instância de uma aplicação multi-tenancy. Em um cenário multi-tenancy é comum que cada tenant necessite priorizar as requisições de seus consumidores de tal maneira que um consumidor com prioridade alta poderá ser atendido mais rapidamente que outro, e que a instância da aplicação também priorize os tenants, de forma que as requisições de um tenant tenham maior prioridade de atendimento que as de outro. Diante desse cenário, Tsai et al.e [55] propõem um modelo para priorizar requisições de vários tenants enquanto preserva as prioridades locais das requisições de um tenant específico. Os autores propõem um algoritmo chamado Crystalline Mapping que mapeia prioridades internas de um tenant específico para prioridades globais.

2.7.2. Banco de Dados

Uma das preocupações quando pretende-se implementar uma aplicação multi-tenancy é planejar como os dados da aplicação serão armazenados. Atualmente, boa parte das aplicações existentes no mercado utilizam bancos de dados relacionais. Existem várias estratégias para implementar um esquema de banco de dados relacional de forma que ele permita o armazenamento de dados de vários tenants. Aulbach et al. [3] apresenta várias dessas técnicas, que são mostradas na Figura 2.10 e brevemente descritas a seguir:

- ***Basic Layout*** **-** a técnica mais básica para implementar multi-tenancy é adicionar uma coluna TENANTID em cada tabela para armazenar um valor que identifica a qual tenant um registro pertence. Essa abordagem provê uma boa consolidação mas não provê uma boa extensibilidade, já que não possui nenhum mecanismo para customização de armazenamento de dados para um tenant específico;

- ***Private Table*** **-** é a forma mais básica para suportar extensibilidade, dado que cada tenant possui sua próprias tabelas privadas. Nessa abordagem, é necessário uma camada de transformação de queries para ajustar o nome das tabelas nas queries e substituir pelo nome da tabela privada;
- ***Extension Table*** **-** essa técnica é a combinação das duas técnicas anteriores, as tabelas de extensão bem como as tabelas base devem possuir uma coluna para armazenar os dados de identificação do tenant. Outra coluna também precisa ser adicionada para armazenar a tabela lógica para que os dados possam ser reconstruídos. Essa abordagem é utilizada para mapear esquemas orientados a objeto com herança nos modelos relacionais atualmente;
- ***Universal Table*** **-** estruturas genéricas permitem a criação de um número arbitrário de tabelas com formas arbitrárias. Universal Table é uma estrutura genérica com uma coluna Tenant, uma coluna Table e um grande número de colunas de dados genéricas. A coluna de dados tem um tipo flexível, como VARCHAR, no qual outro tipo pode ser convertido;
- ***Pivot Table*** **-** nessa técnica as entidades de domínio são representadas por tabelas lógicas, cujas informações são montadas dinâmicamente. Uma Pivot Table possui as seguintes colunas: Tenant (identifica o tenant), Table (identifica a tabela à qual o dado está associado), Row (identificar a linha à qual o dado está associado) Col (identifica o campo que a linha representa) e uma coluna para armazenar o dado em si, que normalmente é de um tipo flexível como VARCHAR. Essa abordagem proporciona uma alta flexibilidade mas possui muitas colunas de metadados que podem impactar negativamente na performance da aplicação, durante a manipulação dos dados;
- ***Chunk Table*** **-** essa técnica é uma extensão da técnica anterior e trabalha com o conceito de Chunk Table. Uma Chunk Table é semelhante a uma Pivot Table exceto pelo fato de possuir um conjunto de colunas de dados de vários tipos, e a coluna Col é substituida pela coluna Chunk. Em comparação com a técnica Pivot Table, essa técnica armazena uma quantidade menor de metadados, o que diminui o tempo de reconstrução da tabela lógica; e
- ***Chunk Folding*** **-** as tabelas lógicas são verticalmente particionadas em chunks, os quais permitem junção quando necessário.

Essas são as técnicas básicas para a implementação de modelos de dados para aplicações multi-tenancy, outras abordagens foram criadas a partir delas [19, 61, 59]. Aulbach et al. [4] realizam um estudo experimental para comparar cinco técnicas de implementação de aplicações multi-tenancy a nível de banco de dados. Os autores concluíram que ainda não existe um sistema de gerenciamento de banco de dados (SGBD) ideal para esse tipo de aplicação e que um SGBD para SaaS deveria ser baseado na técnica Private Table.

Schiller et al. [52] apresentam em seu trabalho as primeiras funcionalidades para que um SGBD relacional possa suportar múltiplos tenants nativamente. Em sua

proposta tenants são introduzidos como objetos de primeira classe e é proposto o conceito de “contexto” para isolar um tenant de outro. Além disso, o conceito de herança permite compartilhar o esquema da aplicação entre os tenants, ao mesmo tempo em que permite que o esquema seja extendido com estruturas de dados adicionais. Ao final do trabalho o autor realiza uma avaliação da implementação de sua proposta através de experimentos.

2.7.3. Customização

Um ponto importante em uma aplicação multi-tenancy é a customização. Em aplicações web customizáveis o código torna-se mais complexo, problemas de performance impactam todos os tenants da aplicação e planejar segurança e robustez tornam-se pontos importantes. Segundo Jansen et al. [26], existem três áreas de pesquisa que são

$Account_{17}$

Aid	Name	Hospital	Beds
1	*Acme*	*St. Mary*	135
2	*Gump*	*State*	1042

$Account_{42}$

Aid	Name	Dealers
1	*Big*	65

diretamente relacionadas a Técnicas de Realização de Customização (Customization Realization Techniques CRT) em aplicações multi-tenancy: variabilidade em linhas de produtos de software, personalização do usuário final em aplicações web e arquiteturas multi-tenancy. Pesquisadores que pretendem atacar esse problema devem direcionar seus estudos nessas três áreas.

a) Private Table Layout

$Account_{Ext}$

Tenant	Row	Aid	Name
17	0	1	*Acme*
17	1	2	*Gump*
35	0	1	*Ball*
42	0	1	*Big*

$Healthcare_{Account}$

Tenant	Row	Hospital	Beds
17	0	*St. Mary*	135
17	1	*State*	1042

$Automotive_{Account}$

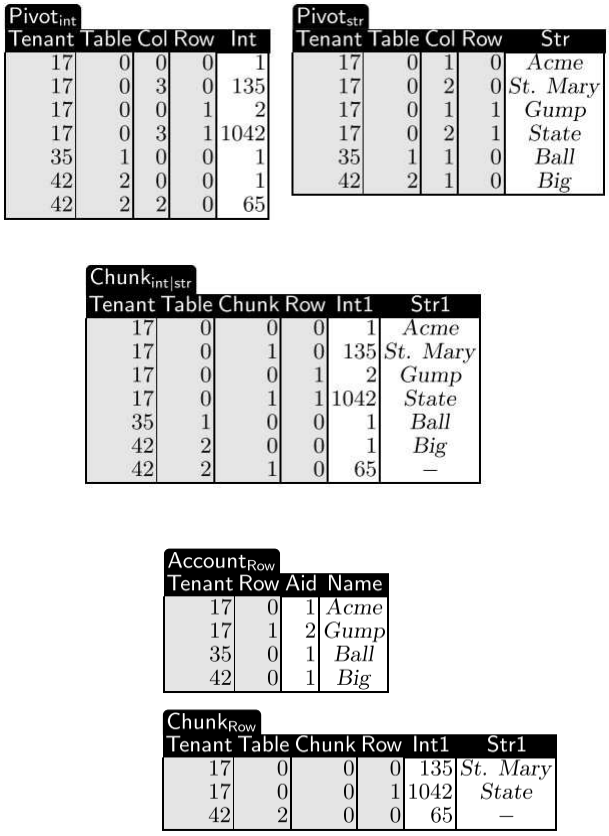
Tenant	Row	Dealers
42	0	65

b) Extension Table Layout

Universal

Tenant	Table	Col1	Col2	Col3	Col4	Col5	Col6
17	0	*1*	*Acme*	*St. Mary*	*135*	–	–
17	0	*2*	*Gump*	*State*	*1042*	–	–
35	1	*1*	*Ball*	–	–	–	–
42	2	*1*	*Big*	*65*	–	–	–

c) Universal Table Layout



Pivot$_{int}$

Tenant	Table	Col	Row	Int
17	0	0	0	1
17	0	3	0	135
17	0	0	1	2
17	0	3	1	1042
35	1	0	0	1
42	2	0	0	1
42	2	2	0	65

Pivot$_{str}$

Tenant	Table	Col	Row	Str
17	0	1	0	*Acme*
17	0	2	0	*St. Mary*
17	0	1	1	*Gump*
17	0	2	1	*State*
35	1	1	0	*Ball*
42	2	1	0	*Big*

Chunk$_{int|str}$

Tenant	Table	Chunk	Row	Int1	Str1
17	0	0	0	1	*Acme*
17	0	1	0	135	*St. Mary*
17	0	0	1	2	*Gump*
17	0	1	1	1042	*State*
35	1	0	0	1	*Ball*
42	2	0	0	1	*Big*
42	2	1	0	65	–

Account$_{Row}$

Tenant	Row	Aid	Name
17	0	1	*Acme*
17	1	2	*Gump*
35	0	1	*Ball*
42	0	1	*Big*

Chunk$_{Row}$

Tenant	Table	Chunk	Row	Int1	Str1
17	0	0	0	135	*St. Mary*
17	0	0	1	1042	*State*
42	2	0	0	65	–

d) Pivot Table Layout

e) Chunk Table Layout

f) Chunk Folding

Figura 2.10. Tecnicas de mapeamento de esquema (Fonte: [3])

Outro ponto importante é que os tenants de uma aplicação multi-tenancy não somente têm diferentes requisitos com respeito a propriedades funcionais, mas também podem exigir diferentes propriedades de qualidade de serviço como privacidade e performance. Alguns tenants exigem que a aplicação possua alta disponibilidade e estão dispostos a pagar um alto preço pelo uso desse serviço. Outros tenants não estão interessados em alta disponibilidade mas preocupam-se mais com um baixo preço. Em casos como esse é necessário que exista na aplicação algum mecanismo para ajustar a aplicação às reais necessidades do usuário, no tocante aos casos citados anteriormente.

2.7.4. Escalabilidade

Esse é um dos problemas mais críticos para serem resolvidos em um cenário real. Dado um número fixo de servidores, é necessário otimizar a distribuição dos tenants de forma a maximizar o número total de tenants possíveis sem violar seus requisitos de SLA e ainda estar preparado para o crescimento do volume de dados e de requisições.

Yuanyuan et al. [63] apresentam uma arquitetura focada na escalabilidade de dados. Eles propõem uma entidade com base em grupos de particionamento horizontal, através da análise das relações entre as entidades de uma aplicação multi-tenancy. Nessa abordagem, entidades de negócio altamente coesas formam um grupo de entidades e o particionamento ocorre com base nesse grupo. Dessa forma cada operação acessa um único grupo de instâncias, podendo obter os dados necessários através do acesso a um único banco de dados e evitando a necessidade de transações distribuídas.

Koziolek [31, 32] apresenta um estilo arquitetural focado em escalabilidade de dados e ilustra como os conceitos apresentados nesses trabalhos podem ajudar a fazer as Plataformas como Serviço (PaaS) atuais, como Force.com, Windows Azure, e Google App Engine, escaláveis e customizáveis. Já Zhang et al. [66] focam no problema de localização de tenants, propõem um algoritmo heurístico chamado Tenant Dispatch Heuristic (TDH) e realizam um conjunto de simulações e comparações onde é avaliado como o uso desse algoritmo impacta na escalabilidade e economia de recursos.

2.7.5. Migração

Migrar aplicações web legadas que trabalham com um único tenant (Isolated Tenancy Hosted Applications ITHA) para multi-tenancy (multi-tenancy Enabled Application MTEA) não é uma tarefa trivial, devido a quantidade de ferramentas e técnicas de migração apropriadas. De acordo com Zhang et al. [65] os métodos existentes para migração são muito abstratos ou muito específicos quando tentamos aplicá-los como guias para migrar uma aplicação que trabalha com um tenant isolado para uma aplicação multi-tenancy. Nesse mesmo trabalho os autores propõem um método sistemático que provê diretrizes para migrar aplicações ITHA para MTEA, levando em conta custo, risco e efetividade do método de migração.

Dado a característica de demanda sazonal existente em vários tipos de aplicação, uma funcionalidade essencial para um ambiente multi-tenancy é a funcionalidade que permite a migração de tenants entre hosts, uma técnica chamada live migration [13, 40] é apresentada por alguns autores como uma possível solução.

Elmore et al. [17] apresentam Zephyr, uma técnica para live migration que tem o objetivo de minimizar a interrupção de serviço do tenant migrado. Através da introdução de uma sincronização dupla que permite a ambos os nós, o nó origem dos dados e o nó destino, executarem simultaneamente transações para o tenant. A migração inicia-se com a transferência dos metadados do tenant para o nó destino, que pode realizar novas transações enquanto o nó origem completa as transações que foram iniciadas quando a migração se iniciou. De acordo com os autores, live migration é uma importante característica para habilitar elasticidade em bancos de dados multi-tenancy para plataformas de cloud.

2.7.6. Monitoramento

Para obter uma visão geral do funcionamento de seus serviços, provedores precisam de informações sobre todas as camada de seu sistema, incluindo a aplicação, máquinas virtuais e uso de rede. Por razões de simplicidade, todas essas informações devem idealmente ser entregues através de uma única interface de monitoramento. Hasselmeyer e D'Heureuse [22] apresentam alguns requisitos básicos de uma infraestrutura de monitoramento para ambientes multi-tenancy:

- ***Multi-tenancy:*** a infraestrutura de monitoramento precisa naturalmente ser capaz de lidar com informações de monitoramento provinientes de diferentes clientes (tenants);
- ***Escalabilidade:*** uma solução de monitoramento precisa ser escalável em vários aspectos. Ela precisa escalar para um grande número de agentes de monitoramento, de notificação de eventos, de tenants, de recursos (virtuais e físicos, servidores, elementos de rede e aplicações), e de tipos de informação de monitoramento;

- ***Dinamismo:*** sistemas de monitoramento multi-tenancy precisam suportar o dinamismo que é inerente a um ambiente multi-tenancy. Esse dinamismo decorre da rápida e frequente adição e remoção de tenants no datacenter. Além disso, a alocação de recursos para os tenants também está em constante mudança e o conjunto de recursos que está sendo monitorado também pode mudar;

- ***Simplicidade:*** o sistema de monitoramento deveria ser simples em dois aspectos: primeiro, a interface para monitoramento do sistema precisa ser fácil de entender e usar. Segundo, o sistema precisar ser fácil de instalar e manter, tanto para quem gerencia o datacenter quanto para os consumidores. Uma API simples é desejável, uma vez que facilita o trabalho de desenvolvedores que criam soluções de controle e monitoramento; e

- ***Abrangência:*** esse requisito aborda a necessidade de uma infraestrutura única de monitoramento que deve ser utilizável para todos os tipos de informação de monitoramento, não importa o que está relacionado ao recurso ou qual o significado que ele transmite. Essa abrangência aplica-se a múltiplos aspectos: tipos de dados, fontes de notificação e tenants.

Já Cheng et al. [11] propõem um mecanismo de controle que monitora a qualidade de serviço por tenant, detecta situações anormais e dinamicamente ajusta o uso de recursos baseado nos parâmetros de SLA definidos para o tenant. Nesse trabalho os autores apresentam sua proposta e realizam um estudo experimental para avaliar os resultados.

2.7.7. Performance

Esse assunto está diretamente ligado à monitoramento de aplicações multi-tenancy, isso porque aplicações são monitoradas para que se possa verificar se a mesma está executando em um nível de performance compatível com a SLA definida para o cliente. Desde 2008 temos um bom número de trabalhos que estudam esse problema, dentre eles estão propostas de arquitetura, frameworks, métodos, técnicas, ferramentas e experimentos, que utilizam as mais variadas abordagens. O primeiro trabalho encontrado relacionado à performance foi o de Wang et al. [58], onde os autores apresentam os principais padrões de implementação de aplicações multi-tenancy que tratam dos aspectos de isolamento e segurança, e avaliam a performance desse padrões através de uma série de experimentos e simulações.

Para permitir que um provedor de serviço ofereça diferentes níveis de performance baseado no nível de SLA definido para um tenant específico, Lin et al.[38] propõe uma solução que utiliza um regulador de performance baseado no controle de feedback. O regulador possui uma estrutura hierárquica, na qual um controlador de alto nível gerencia a taxa de admissão de requisições para prevenir sobrecarga e um controlador de baixo nível que gerencia os recursos alocados pelas requisições admitidas para alcançar um nível específico de diferenciação de serviço entre os tenants que compartilham os mesmos recursos. Um protótipo dessa abordagem é implementado utilizando Tomcat e MySQL, e ao final são realizados alguns experimentos para validar a proposta.

Outro ponto importante é o isolamento de performance para prevenir potenciais anomalias de comportamento de um tenant, que possam afetar a performance de outros tenants que compartilham os mesmos recursos. Li et al. [36] propõem o SPIN (Service Performance Isolation Infrastructure) que visa permitir um abrangente compartilhamento de recursos em ambientes multi-tenancy. Uma vez que alguns tenants agressivos podem interferir na performance de outros, SPIN fornece um relatório de anomalias, identifica os tenants agressivos, e fornece um mecanismo de moderação para eliminar o impacto negativo em outros tenants. Durante sua pesquisa os autores implementaram um protótipo do SPIN e realizaram alguns experimentos para demonstrar sua eficiência.

Um desafio interessante na área de gerenciamento de performance é entender e predizer performance de sistemas. Durante a realização desse mapeamento foram encontradas duas abordagens relacionadas a esse assunto. Schaffner et al. [51] apresentam um estudo sobre a automação de tarefas operacionais em clusters multi-tenancy de bancos de dados em memória orientados a coluna. Foi desenvolvido um modelo para predizer se a alocação de um tenant em um servidor no cluster, levaria a uma violação no tempo de resposta esperado. Já Ahmad e Bowman [1] realizaram um estudo experimental para entender e predizer a performance de sistemas, nesse trabalho

os autores sugerem uma abordagem de máquina de aprendizado que usa dados monitorados para entender a performance do sistema.

Guo et al. [20] apresentam um framework que possui um componente específico para isolamento de performance. Segundo os autores os objetivos principais do isolamento de performance incluem dois aspectos. Primeiro, evitar que o comportamento (potencialmente ruim) de um tenant afete o comportamento de outro, de maneira imprevista. Segundo, evitar a injustiça entre tenants em termos de performance de uso. Para alcançar esse objetivo os autores sugerem um padrão de isolamento de performance híbrido, baseado em várias técnicas apresentadas em seu trabalho.

2.7.8. Segurança

Confiança é um dos maiores desafios que influenciam a ampla aceitação de SaaS. Na ausência de confiança em SaaS, segurança e privacidade de dados figuram entre os principais e mais importantes assuntos relacionados a arquitetura multi-tenancy. Foram encontrados durante esta pesquisa alguns trabalhos publicados nos últimos 2 anos que estão relacionados a esse assunto.

Um assunto bastante relevante é a avaliação de credibilidade de tenants, que indica quais tenants têm um comportamento que possa prejudicar o funcionamento dos demais. Zhiqiang [46] apresenta um algoritmo de avaliação de credibilidade de tenants baseado na experiência. Essa abordagem visa realizar a detecção e gerenciamento de tenants maliciosos a um baixo custo. De acordo com o histórico de acesso dos tenants, ele pode calcular a credibilidade de tenants, atribuir privilégios de acesso e determinar estratégias de detecção e monitoramento.

De acordo com Li et al. [37], separação de responsabilidade de segurança entre provedores SaaS e consumidores precisa ser suportada em um ambiente de cloud. Arquitetura multi-tenancy baseada em modelo de controle de acesso (MTACM multi-tenancy based access control model) foi projetada para inserir o princípio de separação de responsabilidade de segurança em cloud; essa arquitetura define dois níveis de granularidade no mecanismo de controle de acesso. O primeiro nível está relacionado ao provedor de serviço, que compartilha sua infraestrutura entre vários clientes. O segundo nível é o nível da aplicação onde uma mesma aplicação hospeda informações de vários tenants.

Zhang et al. [64], Calero et al. [10] e Tsai et al. [54] apresentam três abordagens diferentes para implementação de mecanismos de segurança e controle de acesso para aplicações multi-tenancy. Zhang et al. [64] apresentam uma abordagem de controle de acesso baseado em restrições de privacidade customizáveis. Essa abordagem combina criptografia de dados e separação de informação e define três tipos de restrições de privacidade baseado na funcionalidade de customização em aplicações SaaS. Calero et al. [10] descrevem um sistema de autorização para um serviço de middleware em uma PaaS, que suporta controle de acesso baseado em hierarquia de

funções, hierarquia de objetos baseada em caminho e federações. Os autores apresentam também uma arquitetura de sistema de autorização para implementação do modelo.

De acordo com Tsai et al. [54] as abordagens atuais para controle de acesso em cloud não escalam bem para requisitos multi-tenancy porque eles são baseados principalmente em identificadores (IDs) de usuário individual em diferentes níveis de granularidade. No entanto, o número de usuários pode ser enorme e causar um overread significativo no gerenciamento de segurança. RBAC (Role-Based Access Control) tornase atrativo nesse caso pelo fato do número de papéis de usuário em uma aplicação ser significativamento menor, e usuários podem ser classificados de acordo com seus papéis. Os autores propõem um RBAC usando uma ontologia de papéis para arquitetura multitenancy em clouds.

2.7.9. Integração com outros sistemas

De acordo com o cenário apresentado até aqui, é possível perceber que seria muito difícil e caro criar uma aplicação multi-tenancy que fosse tão configurável a ponto de permitir que todos os requisitos do usuário pudessem ser atendidos através de configurações. Mesmo nesse cenário é possível permitir que os próprios usuários implementem parte de suas soluções e as integrem à plataforma multi-tenancy utilizada. Para solucionar esse problema, alguns autores propuseram abordagens para implementar arquitetura multi-tenancy com SOA.

Azeez et al. [5] apresentam uma arquitetura para implementar multi-tenancy no nível de SOA, que habilita usuários a executar seus serviços e outros artefatos SOA em um framework multi-tenancy SOA. Em seu trabalho os autores discutem arquitetura, decisões de projeto, e problemas encontrados, juntamente com potenciais soluções. Diferentes aspectos de arquitetura de sistemas no que se refere a multi-tenancy para SOA também são considerados como: deployment de serviços, envio de mensagens, segurança, execução de serviços e finalmente acesso a dados.

2.8. Considerações Finais

Grandes empresas de TI (Tecnologia da Informação), como HP e IBM têm investido bastante nessa área, isso é comprovado com o fato de vários artigos encontrados durante esse estudo serem escritos por membros dessas empresas. Além disso, algumas empresas também vendem aplicativos que podem ser configurados para executar como SaaS em nuvens privadas; alguns sistemas da SAP por exemplo, podem ser usados como um SaaS oferecido dentro das empresas.

A primeira dificuldade encontrada durante as pesquisas foi identificar quais os requisitos de uma aplicação multi-tenancy que são essenciais para sua implementação. Durante a tentativa de identificá-los, foi possível perceber que duas características chave de multi-tenancy são o auto grau de automação nas atividades de manutenção da aplicação e uma interface amigável para que o usuário possa realizar suas

customizações, reduzindo ao máximo a necessidade de intervenções por parte de desenvolvedores e administradores de sistema.

Durante a elaboração desse trabalho foi possível notar também a existência de muitos trabalhos associados à multi-tenancy e armazenamento de dados. Propostas de métodos, técnicas e até uma proposta de SGBD multi-tenancy foi encontrada. Embora esse seja o campo de multi-tenancy onde mais se encontrou publicações, ainda é um campo onde existem muitos pontos de melhoria, principalmente pelo fato do armazenamento de dados influenciar diretamente no tempo de resposta de todas as operações que manipulam dados neste tipo de ambiente.

Embora vários experimentos já tenham sido realizados com as abordagens existentes na literatura para armazenamento de dados em aplicações multi-tenancy, não foi encontrado um consenso sobre qual abordagem é a melhor. Migração de dados, modelagem dos dados, tempo de resposta, armazenamento distribuído, integração com ferramentas de BI (Business Inteligence), todos esses fatores podem influenciar na escolha de uma abordagem para armazenamento de dados.

Atualmente, novas formas de armazenamento de dados estão surgindo como: HBase[25], Cassandra[26], Hipertable[27], MongoDB[28], CouchDB[29], etc. Esses sistemas armazenam dados de uma forma diferente do usado nos modelos relacionais e compõem um movimento chamado NoSQL. Uma lacuna pouco explorada foi a utilização dessas novas tecnologias no desenvolvimento de aplicações multi-tenancy, dado que bancos de dados NoSQL surgiram como uma solução para a questão da escalabilidade no armazenamento e processamento de grandes volumes de dados[14].

A necessidade de uma nova tecnologia de banco de dados surgiu como consequência da ineficiência dos bancos de dados relacionais em lidar com a tarefa de manipulação de grandes volumes de dados. Vale ressaltar que o modelo relacional foi proposto na década de 70, quando as aplicações de banco de dados caracterizavam-se por lidar com dados estruturados, ou seja, dados que possuem uma estrutura fixa e bem definida, e esse não é o caso de aplicações multi-tenancy que podem ser customizadas [41].

O crescente surgimento de APIs [16] web mostra que a utilização de APIs para permitir que usuários possam criar extensões de aplicações faz muito sentido no contexto de aplicações web e SaaS. Um exemplo de sucesso é o Tweetdeck, uma aplicação cliente do twitter que foi desenvolvida utilizando a API disponibilizada pelo twitter aos desenvolvedores e teve grande aceitação pelos usuários do twitter.com. O valor real da aquisição não foi declarado, mas especula-se algo em torno de 40 milhões de dólares [6]. Uma API web bem definida pode permitir que usuários de uma

[25] http://hbase.apache.org
[26] http://cassandra.apache.org
[27] http://hypertable.com
[28] http://www.mongodb.org
[29] http://couchdb.apache.org

aplicação multi-tenancy possam, eles mesmos, criarem extensões para a aplicação e até fazer disso um negócio.

Outro exemplo de sucesso é o marketplace desenvolvido pela Salesforce, o AppExchange, que possui mais de 1 milhão de aplicações cadastradas, desde extensões da aplicação de CRM do Salesforce até aplicações para outros propósitos. Exemplos como esse mostram quão vantajoso é trazer desenvolvedores independentes para o mercado de SaaS.

A realidade é que para entender o campo de customização de aplicações multitenancy é necessário entender o contexto no qual essas aplicações estão surgindo. Dado que não é possível atender com uma única aplicação a necessidade de todos os clientes, como já foi mencionado anteriormente, enfrenta-se em aplicações multi-tenancy o desafio de permitir de alguma forma que o cliente adicione novas regras e customize a aplicação ou faça extensões que auxiliem na solução de seus problemas específicos. Durante esse trabalho foram encontrados vários estudos relacionados ao tema, muitos deles utilizando metadados, orientação a aspectos, SOA, etc.

Customização de aplicações já é um tema bastante explorado pelos pesquisadores da área de linha de produtos de software. Basicamente podemos considerar que a customização de aplicações multi-tenancy é equivalente a variabilidade de linha de produtos de software em tempo de execução. Uma iniciativa de customização de aplicações multi-tenancy tomando como base os conhecimentos de linha de produtos de software é apresentado por Jansen et al. [26].

Dado que SaaS tem como objetivo prover software para uma grande massa de consumidores, o atendimento aos requisitos de alocação de recursos, monitoramento, escalabilidade e performance são requisitos não funcionais que podem até ser ignorados em uma aplicação comum, mas que são de vital importância em aplicações multi-tenancy, principalmente se a aplicação estiver hospedada em um ambiente de cloud onde a tarifação (pagamento) é realizada por consumo. Nesses ambientes uma aplicação mal projetada pode levar a custos operacionais tão altos que podem inviabilizar o funcionamento da aplicação, dado o valor que deverá ser pago para manter a infraestrutura de servidores funcionando.

Por outro lado, em uma contexto onde a alocação de recursos é feita de forma inteligente, o monitoramento indica quais recursos estão ociosos e quais os gargalos da aplicação. Nesse caso, se a elasticidade (capacidade de aumentar e diminuir os recursos de hardware) é realizada de forma automática e inteligente, os custos operacionais tendem a ser diminuidos drasticamente se a aplicação consumir somente o que é estritamente necessário para atender aos requisitos de SLA definidos aos clientes.

Nos últimos anos muitas empresas têm saído do modelo de entrega de software empacotado para o modelo de fornecimento de software na web. Essas aplicações entregues através da web vão desde emails a calendários, sistemas colaborativos, publicações online, processadores de texto simples, aplicações para negócios e

aplicações para uso pessoal. Com o aumento da popularidade de marketing baseado em web, e-commerce e muitos outros serviços baseados em web, mesmo os pequenos negócios têm sido obrigados oferecer seus produtos e serviços na internet para serem competitivos no mercado. A característica de compatilhamento de recursos e diminuição de custos trazidos por multitenancy, têm feito dessa abordagem uma excelente alternativa para prover software de qualidade e com um baixo custo para pequenas e médias empresas.

Por fim, consideramos que a implementação de multi-tenancy não só é viável do ponto de vista de negócios como do ponto de vista técnico, embora ainda existam problemas e pontos de melhoria que devem ser tratados.

2.9. Referências

[1] M. Ahmad and I. T. Bowman. Predicting system performance for multi-tenant database workloads. In Proceedings of the Fourth International Workshop on Testing Database Systems, DBTest '11, pages 6:1–6:6, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0655-3. doi: http://doi.acm.org/10.1145/1988842.1988848. URL http://doi.acm.org/10.1145/1988842.1988848.

[2] Amazon. What is AWS? Amazon Company, 2011. URL http://aws.amazon.com/pt/what-is-aws. Disponível em: <http://aws.amazon.com/pt/what-is-aws>. Acesso em: 10 dez. 2011.

[3] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger. Multi-tenant databases for software as a service: schema-mapping techniques. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, SIGMOD '08, pages 1195–1206, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-102-6.doi: http://doi.acm.org/10.1145/1376616.1376736. URL http://doi.acm.org/10.1145/1376616.1376736.

[4] S. Aulbach, D. Jacobs, A. Kemper, and M. Seibold. A comparison of flexible schemas for software as a service. In Proceedings of the 35th SIGMOD international conference on Management of data, SIGMOD '09, pages 881–888, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-551-2. doi: http://doi.acm.org/10.1145/1559845.1559941. URL http://doi.acm.org/10.1145/1559845.1559941.

[5] A. Azeez, S. Perera, D. Gamage, R. Linton, P. Siriwardana, D. Leelaratne, S. Weerawarana, and P. Fremantle. Multi-tenant soa middleware for cloud computing. In Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, pages 458 – 465, july 2010. doi: 10.1109/CLOUD.2010.50.

[6] BBC News. Twitter acquires Tweetdeck app for undisclosed fee. BBC News, 2011. URL http://www.bbc.co.uk/news/technology-13518382. Disponível em: <http://www.bbc.co.uk/news/technology-13518382>. Acesso em: 10 dez. 2011.

[7] D. Berberova and B. Bontchev. Design of service level agreements for software services. In Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing, CompSysTech'09, pages 26:1–26:6, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-986-2. doi: http://doi.acm.org/10.1145/1731740.1731769. URL http://doi.acm.org/10.1145/1731740.1731769.

[8] C.-P. Bezemer and A. Zaidman. Multi-tenant saas applications: maintenance dream or nightmare? In Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), IWPSE-EVOL '10, pages 88–92, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0128-2. doi: http://doi.acm.org/10.1145/1862372.1862393. URL http://doi.acm.org/10.1145/1862372.1862393.

[9] C.-P. Bezemer, A. Zaidman, B. Platzbeecker, T. Hurkmans, and A. 't Hart. Enabling multi-tenancy: An industrial experience report. In Software Maintenance (ICSM), 2010 IEEE International Conference on, pages 1 –8, sept. 2010. doi: 10.1109/ICSM.2010.5609735.

[10] J. Calero, N. Edwards, J. Kirschnick, L. Wilcock, and M. Wray. Toward a multitenancy authorization system for cloud services. Security Privacy, IEEE, 8(6):48–55, nov.-dec. 2010. ISSN 1540-7993. doi: 10.1109/MSP.2010.194.

[11] X. Cheng, Y. Shi, and Q. Li. A multi-tenant oriented performance monitoring, detecting and scheduling architecture based on sla. In Pervasive Computing (JCPC), 2009 Joint Conferences on, pages 599 –604, dec. 2009. doi: 10.1109/JCPC.2009.5420114.

[12] F. Chong and G. Carraro. Architecture strategies for catching the long tail. 2006. URL http://msdn2.microsoft.com/en-us/library/aa479069.aspx. Disponível em: <http://msdn2.microsoft.com/en-us/library/aa479069.aspx>. Acesso em: 10 dez. 2011.

[13] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation Volume 2, NSDI'05, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1251203.1251223.

[14] D. Diana, Mauricio, Gerosa, and M. Aurélio. Nosql na web 2.0: Um estudo comparativo de bancos não-relacionais para armazenamento de dados na web 2.0. Communications, 2010.

[15] S. Dustdar and W. Schreiner. A survey on web services composition. Int. J. Web Grid Serv., 1:1–30, August 2005. ISSN 1741-1106. doi: 10.1504/IJWGS.2005.007545. URL http://dl.acm.org/citation.cfm?id=1358537.1358538.

[16] A. DuVander. Api growth doubles in 2010, social and mobile are trends. 2011. URL http://blog.programmableweb.com/2011/01/03/api-growth-doubles-in-2010-social-and-mobile-are-trends/.

[17] A. J. Elmore, S. Das, D. Agrawal, and A. El Abbadi. Zephyr: live migration in shared nothing databases for elastic cloud platforms. In Proceedings of the 2011 international conference on Management of data, SIGMOD '11, pages 301–312, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0661-4. doi: http://doi.acm.org/10.1145/1989323.1989356.

[18] C. Fehling, F. Leymann, and R. Mietzner. A framework for optimized distribution of tenants in cloud applications. In Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing, CLOUD '10, pages 252–259, Washington, DC, USA, 2010. IEEE Computer Society. ISBN 978-0-7695-4130-3. URL http://dx.doi.org/10.1109/CLOUD.2010.33.

[19] F. Foping, I. Dokas, J. Feehan, and S. Imran. A new hybrid schema-sharing technique for multitenant applications. In Digital Information Management, 2009. ICDIM 2009. Fourth International Conference on, pages 1 –6, nov. 2009. doi:10.1109/ICDIM.2009.5356775.

[20] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao. A framework for native multitenancy application development and management. In E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on, pages 551 –558, july 2007. doi: 10.1109/CEC-EEE.2007.4.

[21] I. S. Harris and Z. Ahmed. An open multi-tenant architecture to leverage smes. European Journal of Scientific Research, 65(4):601–610, 2011.

[22] P. Hasselmeyer and N. d'Heureuse. Towards holistic multi-tenant monitoring for virtual data centers. In Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP, pages 350 –356, april 2010. doi: 10.1109/NOMSW.2010.5486528.

[23] J. Huang. Gartner says worldwide software as a service revenue is forecast to grow 21 percent in 2011. 2011. URL http://www.gartner.com/it/page.jsp?id=1739214. Disponível em: <http://www.gartner.com/it/page.jsp?id=1739214>. Acesso em: 10 dez. 2011.

[24] J. Huang. Amazon s3's amazing growth. Cloud Computing Journal, 2011. URL http://cloudcomputing.sys-con.com/node/1699373. Disponível em: <http://cloudcomputing.sys-con.com/node/1699373>. Acesso em: 10 dez. 2011.

[25] B. Jacob. Introduction to Grid Computing. IBM redbooks. IBM, International Technical Support Organization, 2005. ISBN 9780738494005. URL http://books.google.com.br/books?id=4GSUtgAACAAJ.

[26] S. Jansen, G.-J. Houben, and S. Brinkkemper. Customization realization in multitenant web applications: case studies from the library sector. In Proceedings of the 10th international conference on Web engineering, ICWE'10, pages 445–459, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-13910-8, 978-3-642-13910-9. URL http://dl.acm.org/citation.cfm?id=1884110.1884147.

[27] A. Jasti, P. Shah, R. Nagaraj, and R. Pendse. Security in multi-tenancy cloud. In Security Technology (ICCST), 2010 IEEE International Carnahan Conference on, pages 35 –41, oct. 2010. doi: 10.1109/CCST.2010.5678682.

[28] J. Jing and J. Zhang. Research on open saas software architecture based on soa. In Computational Intelligence and Design (ISCID), 2010 International Symposium on, volume 2, pages 144 –147, oct. 2010. doi: 10.1109/ISCID.2010.125.

[29] C. M. Judd, J. F. Nusairat, J. Shingler, M. Moodie, J. Ottinger, J. Pepper, F. Pohlmann, B. Renow-clarke, D. Shakeshaft, M. Wade, and et al. Beginning groovy and grails from novice to professional. Specialist, page 440, 2008. URL http://books.google.com/books?hl=en&lr=&id=9AxFrcvawvAC&oi=fnd&pg=PR15&dq=Beginning+Groovy+and+Grails:+From+Novice+to+Professional&ots=AWQ-JVaL8&sig= BzgKgJMF9jI7E27n9UwfxSRT4OY.

[30] L. Kleinrock. An internet vision: the invisible global infrastructure. Ad Hoc Networks, 1(1):3–11, 2003. URL http://linkinghub.elsevier.com/retrieve/pii/S157087050300012X.

[31] H. Koziolek. Towards an architectural style for multi-tenant software applications. Proc Software Engineering 2010 SE10 Fachtagung des GIFachbereichs Softwaretechnik, To appear:81–92, 2010.nURL http://www.koziolek.de/docs/Koziolek2010-SE.pdf.

[32] H. Koziolek. The sposad architectural style for multi-tenant software applications. In Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on, pages 320 –327, june 2011. doi: 10.1109/WICSA.2011.50.

[33] T. Kwok and A. Mohindra. Resource calculations with constraints, and placement of tenants and instances for multi-tenant saas applications. In Proceedings of the 6th International Conference on Service-Oriented Computing, ICSOC '08, pages 633–648, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-89647-0. URL http://dx.doi.org/10.1007/978-3-540-89652-4_57.

[34] T. Kwok, T. Nguyen, and L. Lam. A software as a service with multi-tenancy support for an electronic contract management application. In Services Computing,

2008. SCC ’08. IEEE International Conference on, volume 2, pages 179 –186, july 2008. doi: 10.1109/SCC.2008.138.

[35] S. Lehnen. Understanding the basic building blocks of sales force crm. 2011. URL http://www.salesforce.com/customer-resources/learning-center/details/best-practices/understanding-the-basic-building.jsp. Disponível em: <http://www.salesforce.com/customer-resources/learning-center/details/best-practices/understanding-the-basic-building.jsp>. Acesso em: 28 dez. 2011.

[36] X. H. Li, T. C. Liu, Y. Li, and Y. Chen. Spin: Service performance isolation infrastructure in multi-tenancy environment. In Proceedings of the 6th International Conference on Service-Oriented Computing, ICSOC ’08, pages 649–663, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-89647-0. doi: http://dx.doi.org/10.1007/978-3-540-89652-4_58. URL http://dx.doi.org/10.1007/978-3-540-89652-4_58.

[37] X.-Y. Li, Y. Shi, Y. Guo, and W. Ma. Multi-tenancy based access control in cloud. In Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on, pages 1 –4, dec. 2010. doi: 10.1109/CISE.2010.5677061.

[38] H. Lin, K. Sun, S. Zhao, and Y. Han. Feedback-control-based performance regulation for multi-tenant applications. In Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on, pages 134 –141, dec. 2009. doi: 10.1109/ICPADS.2009.22.

[39] H. Lin, K. Sun, S. Zhao, and Y. Han. Feedback-control-based performance regulation for multi-tenant applications. In Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on, pages 134 –141, dec. 2009. doi: 10.1109/ICPADS.2009.22.

[40] H. Liu, H. Jin, X. Liao, L. Hu, and C. Yu. Live migration of virtual machine based on full system trace and replay. In Proceedings of the 18th ACM international symposium on High performance distributed computing, HPDC ’09, pages 101–110, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-587-1. URL http://doi.acm.org/10.1145/1551609.1551630.

[41] B. F. Lóscio, H. R. d. Oliveira, and J. C. d. S. Pontes. Nosql no desenvolvimento de aplicações web colaborativas. VIII Simpósio Brasileiro de Sistemas Colaborativos, 2011.

[42] P. Mell and T. Grance. The nist definition of cloud computing. National Institute of Standards and Technology, 53(6):50, 2009. URL http://www.nist.gov/ itl/cloud/upload/cloud-def-v15.pdf.

[43] A. Mesbah and A. van Deursen. Crosscutting concerns in j2ee applications. In Web Site Evolution, 2005. (WSE 2005). Seventh IEEE International Symposium on, pages 14 – 21, sept. 2005. doi: 10.1109/WSE.2005.4.

[44] R. Mietzner, T. Unger, R. Titze, and F. Leymann. Combining different multitenancy patterns in service-oriented applications. In Proceedings of the 2009 IEEE International Enterprise Distributed Object Computing Conference (edoc 2009), EDOC '09, pages 131–140, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3785-6. doi: http://dx.doi.org/10.1109/EDOC.2009.13. URL http://dx.doi.org/10.1109/EDOC.2009.13.

[45] T. Neil. 12 standard screen patterns, 2010. URL http://designingwebinterfaces.com/designing-web-interfaces-12-screen-patterns. Disponível em:<http://designingwebinterfaces.com/designing-web-interfaces-12-screen-patterns>. Acesso em: 10 dez. 2011.

[46] Z. Nie. Credibility evaluation of saas tenants. In Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on, volume 4, pages V4–488 –V4–491, aug. 2010. doi: 10.1109/ICACTE.2010.5579322.

[47] Nitu. Configurability in saas (software as a service) applications. In Proceedings of the 2nd India software engineering conference, ISEC '09, pages 19–26, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-426-3. doi: http://doi.acm.org/10.1145/1506216.1506221. URL http://doi.acm.org/10.1145/1506216.1506221.

[48] B.-T. Oh, H.-S. Won, and S.-J. Hur. Multi-tenant supporting online application service system based on metadata model. In Advanced Communication Technology (ICACT), 2011 13th International Conference on, pages 1173 –1176, feb. 2011.

[49] Z. Pervez, S. Lee, and Y.-K. Lee. Multi-tenant, secure, load disseminated saas architecture. In Advanced Communication Technology (ICACT), 2010 The 12th International Conference on, volume 1, pages 214 –219, feb. 2010.

[50] C. Richardson. Orm in dynamic languages. Commun. ACM, 52(4):48–55, Apr.2009. ISSN 0001-0782. doi: 10.1145/1498765.1498783. URL http://doi.acm.org/10.1145/1498765.1498783.

[51] J. Schaffner, B. Eckart, D. Jacobs, C. Schwarz, H. Plattner, and A. Zeier. Predicting in-memory database performance for automating cluster management tasks. In Data Engineering (ICDE), 2011 IEEE 27th International Conference on, pages 1264–1275, april 2011. doi: 10.1109/ICDE.2011.5767936.

[52] O. Schiller, B. Schiller, A. Brodt, and B. Mitschang. Native support of multitenancy in rdbms for software as a service. In Proceedings of the 14th International Conference on Extending Database Technology, EDBT/ICDT '11, pages

117–128, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0528-0. doi: http://doi.acm.org/10.1145/1951365.1951382. URL: http://doi.acm.org/10.1145/1951365.1951382.

[53] L. Shwartz, Y. Diao, and G. Grabarnik. Multi-tenant solution for it service management: A quantitative study of benefits. In Integrated Network Management, 2009. IM '09. IFIP/IEEE International Symposium on, pages 721 –731, june 2009. doi: 10.1109/INM.2009.5188879.

[54] W.-T. Tsai and Q. Shao. Role-based access-control using reference ontology in clouds. In Autonomous Decentralized Systems (ISADS), 2011 10th International Symposium on, pages 121 –128, march 2011. doi: 10.1109/ISADS.2011.21.

[55] W.-T. Tsai, Q. Shao, and J. Elston. Prioritizing service requests on cloud with multitenancy. In e-Business Engineering (ICEBE), 2010 IEEE 7th International Conference on, pages 117 –124, nov. 2010. doi: 10.1109/ICEBE.2010.38.

[56] W.-T. Tsai, X. Sun, Q. Shao, and G. Qi. Two-tier multi-tenancy scaling and load balancing. In e-Business Engineering (ICEBE), 2010 IEEE 7th International Conference on, pages 484 –489, nov. 2010. doi: 10.1109/ICEBE.2010.103.

[57] Q. H. Vu, M. Lupu, and B. C. Ooi. Peer-to-peer computing. Media, 2010. URL http://www.springerlink.com/index/10.1007/978-3-642-03514-2.

[58] Z. H. Wang, C. J. Guo, B. Gao, W. Sun, Z. Zhang, and W. H. An. A study and performance evaluation of the multi-tenant data tier design patterns for service oriented computing. In Proceedings of the 2008 IEEE International Conference on e-Business Engineering, pages 94–101, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3395-7. doi: 10.1109/ICEBE.2008.60. URL http://dl.acm.org/citation.cfm?id=1471605.1472222.

[59] C. Weiliang, Z. Shidong, and K. Lanju. A multiple sparse tables approach for multitenant data storage in saas. In Industrial and Information Systems (IIS), 2010 2nd International Conference on, volume 1, pages 413 –416, july 2010. doi: 10.1109/ INDUSIS.2010.5565824.

[60] C. D. Weissman and S. Bobrowski. The design of the force.com multitenant internet application development platform. In Proceedings of the 35th SIGMOD international conference on Management of data, SIGMOD '09, pages 889–896, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-551-2. URL http://doi.acm.org/10.1145/1559845.1559942.

[61] W. Xue, L. Qingzhong, and K. Lanju. Multiple sparse tables based on pivot table for multi-tenant data storage in saas. In Information and Automation (ICIA), 2011 IEEE International Conference on, pages 634 –637, june 2011. doi: 10.1109/ICINFA.2011.5949071.

[62] C. S. Yeo, R. Buyya, H. Pourreza, R. Eskicioglu, P. Graham, F. Sommers, and G. Computing. Cluster computing: High-performance, high-availability, and highthroughput processing on a network of computers. Communication, pages 1–24, 2006. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.66.1453.

[63] D. Yuanyuan, N. Hong, W. Bingfei, and L. Lei. Scaling the data in multi-tenant business support system. In Knowledge Engineering and Software Engineering, 2009. KESE '09. Pacific-Asia Conference on, pages 43 –46, dec. 2009. doi: 10.1109/KESE.2009.20.

[64] K. Zhang, Y. Shi, Q. Li, and J. Bian. Data privacy preserving mechanism based on tenant customization for saas. In Multimedia Information Networking and Security, 2009. MINES '09. International Conference on, volume 1, pages 599 –603, nov. 2009. doi: 10.1109/MINES.2009.256.

[65] X. Zhang, B. Shen, X. Tang, and W. Chen. From isolated tenancy hosted application to multi-tenancy: Toward a systematic migration method for web application. In Software Engineering and Service Sciences (ICSESS), 2010 IEEE International Conference on, pages 209 –212, july 2010. doi: 10.1109/ICSESS.2010.5552407.

[66] Y. Zhang, Z. Wang, B. Gao, C. Guo, W. Sun, and X. Li. An effective heuristic for on-line tenant placement problem in saas. In Web Services (ICWS), 2010 IEEE International Conference on, pages 425 –432, july 2010. doi: 10.1109/ICWS.2010. 65.

[67] J. Zheng, X. Li, X. Zhao, X. Zhang, and S. Hu. The research and application of a saas-based teaching resources management platform. In Computer Science and Education (ICCSE), 2010 5th International Conference on, pages 765 –770, aug. 2010. doi: 10.1109/ICCSE.2010.5593500.