

OUTROS TRABALHOS EM:  
[www.projetoederedes.com.br](http://www.projetoederedes.com.br)



Universidade Federal do Ceará  
Centro de Ciências  
Departamento de Computação  
Mestrado e Doutorado em Ciência da Computação

## **SOFTWARE COMO UM SERVIÇO**

Flávio Rubens de Carvalho Sousa

MONOGRAFIA DE QUALIFICAÇÃO

Fortaleza  
Maio - 2010

Universidade Federal do Ceará  
Centro de Ciências  
Departamento de Computação

Flávio Rubens de Carvalho Sousa

## **SOFTWARE COMO UM SERVIÇO**

Monografia submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará como requisito parcial para a Qualificação de Doutorado em Ciência da Computação.

Avaliador: Prof. José Antônio Macedo, DSc

Fortaleza  
Maio - 2010

## RESUMO

Com o crescimento da Internet, cada vez mais desenvolvedores utilizam o modelo de Software como um Serviço (SaaS) para disponibilizar suas aplicações. SaaS é um modelo de entrega de software onde o software não é comprado e executado pelos usuários em infra-estrutura própria, mas executado na infra-estrutura de TI de um provedor de serviços. No modelo de SaaS, um provedor de serviços possui e opera uma aplicação que é acessada por muitos usuários através da Internet.

Os usuários geralmente não precisam comprar licenças de software e instalar pacotes de software nos seus locais ambiente de computação. Eles usam as credenciais emitida pelo provedor de SaaS para se autenticar e consumir o software como um serviço web através da Internet.

Como o software está na Web, ele pode ser acessado pelos usuários de qualquer lugar e a qualquer momento, permitindo mais integração entre unidades de uma mesma empresa ou outros serviços de software. Assim, novos recursos podem ser incorporados automaticamente aos softwares sem que os usuários percebam estas ações, tornando transparente a evolução e atualização dos sistemas.

Este trabalho tem como objetivo apresentar os principais conceitos relacionados a Software como um Serviço. O desenvolvimento de SaaS é destacada, assim como a arquitetura orientada a serviço e o gerenciamento de dados como um serviço. Por fim, desafios e oportunidades de pesquisa em software como um serviço são apresentados.

**Palavras-chave:** Software como um serviço, SaaS, serviço, SOA.

# SUMÁRIO

<b>Capítulo 1—Introdução</b>	<b>1</b>
1.1 Motivação . . . . .	1
1.2 Objetivo e Contribuição . . . . .	2
1.3 Estrutura da Monografia . . . . .	2
<b>Capítulo 2—Software como um Serviço</b>	<b>3</b>
2.1 Introdução . . . . .	3
2.2 Conceitos . . . . .	4
2.3 Desenvolvimento de SaaS . . . . .	9
2.4 Arquitetura Orientada a Serviços . . . . .	11
2.5 Desafios . . . . .	15
2.5.1 Construção de SaaS . . . . .	15
2.5.2 Integração de Serviços . . . . .	15
2.5.3 Modelo de Negócio . . . . .	16
2.6 Conclusão . . . . .	16
<b>Capítulo 3—Banco de Dados como um Serviço</b>	<b>17</b>
3.1 Introdução . . . . .	17
3.2 Requisitos . . . . .	21

3.3	Sistemas de Gerenciamento de Dados como um Serviços . . . . .	22
3.4	Discussão . . . . .	24
3.5	Problemas em Aberto . . . . .	25
3.5.1	Implantação e Provisionamento . . . . .	25
3.5.2	Qualidade do Serviço de Dados . . . . .	26
3.5.3	Segurança . . . . .	26
3.5.4	Modelo de Banco de Dados Multi-Inquilino . . . . .	26
3.6	Conclusão . . . . .	27
	<b>Capítulo 4—Conclusão</b>	<b>28</b>
4.1	Conclusão e Trabalho Futuros . . . . .	28

# CAPÍTULO 1

## INTRODUÇÃO

Esta monografia apresenta um estudo sobre software como um serviço. Este estudo aborda diversos do SaaS, destacando principalmente aspectos relacionados a arquitetura orientada a serviços.

Neste capítulo serão apresentadas a justificativa e a motivação para o desenvolvimento deste trabalho, assim como os objetivos e as contribuições que se pretende alcançar. Ao final do capítulo, será descrito como está organizada o restante desta monografia.

### 1.1 MOTIVAÇÃO

Com o crescimento da Internet, cada vez mais desenvolvedores utilizam o modelo de Software como um Serviço (SaaS) para disponibilizar suas aplicações [1]. SaaS é um modelo de entrega de software onde o software não é comprado e executado pelos usuários em infra-estrutura própria, mas executado na infra-estrutura de TI de um provedor de serviços. No modelo de SaaS, um provedor de serviços possui e opera uma aplicação que é acessada por muitos usuários através da Internet [2].

O modelo de SaaS proporciona softwares com propósitos específicos que são disponíveis para os usuários através da Internet. Os softwares são acessíveis a partir de vários dispositivos do usuário por meio de uma interface *thin client* como um navegador Web. No SaaS, o usuário não administra ou controla a infra-estrutura subjacente, incluindo rede, servidores, sistemas operacionais, armazenamento, ou mesmo as características individuais da aplicação, exceto configurações específicas. Com isso, os desenvolvedores se concentram em inovação e não na infra-estrutura, levando ao desenvolvimento rápido de softwares.

Como o software está na Web, ele pode ser acessado pelos usuários de qualquer lugar e a qualquer momento, permitindo mais integração entre unidades de uma mesma

empresa ou outros serviços de software. Assim, novos recursos podem ser incorporados automaticamente aos softwares sem que os usuários percebam estas ações, tornando transparente a evolução e atualização dos sistemas. O SaaS reduz os custos, pois é dispensada a aquisição de licenças de softwares. Como exemplos de SaaS podemos destacar os serviços de CRM (*Customer Relationship Management*) *on-line* da Salesforce [3] e o Google Docs [4].

## 1.2 OBJETIVO E CONTRIBUIÇÃO

Este trabalho apresenta um estudo sobre software como um serviço. O objetivo é compreender como o SaaS são construídos, utilizados e elencar desafios que devem ser superados para a ampla utilização deste modelo. Também são apresentados aspectos de banco de dados como um serviço (DaaS) e desafios neste contexto específico.

A principal contribuição desta monografia é fazer um levantamento do estado da arte sobre SaaS e DaaS. Outra contribuição é apresentar desafios na construção e utilização de SaaS e DaaS.

## 1.3 ESTRUTURA DA MONOGRAFIA

Os próximos capítulos desta monografia estão estruturados da seguinte forma:

- Capítulo 2: introduz o Software como um Serviço, destacando as principais características e sua arquitetura.
- Capítulo 3: descreve o banco de dados com um serviço.
- Capítulo 4: apresenta as conclusões e os possíveis trabalhos futuros.

## CAPÍTULO 2

# SOFTWARE COMO UM SERVIÇO

Este capítulo discute as principais características de Software como um Serviço, destacando a arquitetura orientada a serviços. Nele também são abordados os papéis dos componentes relacionados a SaaS, assim como aspectos do desenvolvimento de SaaS. Por fim, apresentamos alguns desafios em SaaS.

### 2.1 INTRODUÇÃO

SaaS está ganhando força com o aumento significativo do número de fornecedores movendo-se neste espaço e do recente sucesso de um grupo de empresas líderes no mercado. Projetado para alavancar os benefícios trazidos por economia de escala, SaaS trata de como produzir softwares para um grande grupo de clientes através da web com uma única instância da aplicação sendo executados sob uma plataforma *multi-tenant* ou multi-inquilino [5].

Os usuários geralmente não precisam comprar licenças de software e instalar pacotes de software nos seus locais ambiente de computação. Eles usam as credenciais emitida pelo provedor de SaaS para se autenticar e consumir o software como um serviço web através da Internet.

A idéia de ter Software como um Serviço fornece mudança no modo em que o aplicativo é disponibilizado aos seus usuários. Neste contexto, os usuários utilizam aplicativos que não estão residindo em suas máquinas locais, e sim, em um provedor de serviços. Um usuário, consumidor de serviço, necessita somente de um navegador e uma conexão de internet para utilizar o serviço. O fato de ter um software como serviço oferece uma série de vantagens quanto a atualização do mesmo [6].

Sendo assim, os usuários não necessitam se preocupar com as atualizações destes softwares, pois estes são atualizados em seus servidores, o que automaticamente refletem essas atualizações para os usuários. Uma outra vantagem, desta abordagem centralizada de software, é que os usuários podem utilizar software com poucos recursos de hardware,

pois a maior parte do processamento é feita no servidor. Portanto, as máquinas dos usuários servem apenas como interface para submissão de requisições e visualização dos resultados.

Na abordagem de SaaS é necessário ter um componente que gerencie estes serviços. Um provedor de SaaS hospedam os aplicativos, os dados e aplicam as atualizações de modo centralizado e transparente para os usuários. No modelo SaaS não é só o processamento das requisições que ficam no lado servidor, mas também, todos os recursos que promovem características tais como a segurança e disponibilidade. A segurança pode ser aplicada em diversos contextos como o acesso restrito a recursos, alterações indevidas e confidencialidade das informações. Já a disponibilidade diz respeito às garantias que os usuários necessitam para utilizarem os serviços.

Neste sentido, o serviço tem que estar estruturado para atender as necessidades acordadas entre os usuários e os provedores de serviços. Um das vantagens fornecidas em um modelo SaaS é integração com outros sistemas. Normalmente, um serviço utiliza entradas e saídas padronizadas, conhecidas pelos desenvolvedores de sistemas. Neste sentido, de posse das interfaces do serviço, qualquer sistema poderá utilizar esse serviço para aproveitar, recursos de processamento e software já desenvolvidos.

Portanto, a utilização de serviços para compor um sistema favorece a um rápido desenvolvimento do mesmo, pois o sistema é montado como se duas partes fossem peças de um quebra-cabeça. Um outro ponto de vista interessante, deste tipo de software, é que isso favorece a uma maior disponibilidade pelo fato da distribuição dos recursos. Com isso, se alguns serviços pararem por algum motivo, o sistema como um todo não fica indisponível.

## 2.2 CONCEITOS

O termo multi-inquilino é um dos principais conceitos relacionados a SaaS. Um inquilino em um cenário de SaaS é um usuário que utiliza SaaS. Ele refere-se ao uso do mesmo software e instâncias por vários usuários e empresas de forma simultânea. O objetivo dessa abordagem é disponibilizar os mesmos recursos de software para um número maior de usuários.

Esta abordagem está baseada no conceito da “Cauda Longa” [7], utilizada por diversas empresas de internet que conseguem obter renda tanto com produtos de nicho

como produtos tradicionais.

Com o seu artigo “The Long Tail”, o escritor Chris Anderson popularizou a idéia da “longa cauda” ao explicar por que os varejistas on-line como Amazon.com estão posicionados de uma forma singular para atender uma imensa demanda que os varejistas tradicionais não podem atender de uma maneira econômica.

Os fornecedores de soluções complexas de software de linha de negócios se deparam com uma curva de mercado semelhante. A Figura 2.1 mostra este cenário.

### The Long Tail / A Cauda Longa



**Figura 2.1** Gráfico da Cauda Longa

O gráfico demonstra que, conforme diminui-se o custo de adoção, um número maior de clientes pode adotar uma solução, sendo que este número tende ao infinito, uma vez que a curva não toca o eixo “x”. Assim, no modelo SaaS de fornecimento de software, precisa-se desenvolver soluções e infra-estruturas de baixo custo, com alto aproveitamento de recursos por um número muito grande de usuários, para assim atingir um público não suportado hoje em dia, devido os custos proibitivos de entrada.

Outro conceito importante do modelo é o “micro-pagamento”. Na cauda longa, um número muito grande de usuários poderá adotar nossa solução pagando pelo uso, por demanda, o que deve gerar um valor muito baixo de pagamento. Neste contexto, procura-se o chamado “milhões de mercados de poucos” ao invés dos atuais “poucos mercados de milhões”.

## Caracterização

Existem três diferenças principais entre SaaS e softwares tradicionais: tipo de produto, preço e forma de entrega [8]. Em primeiro lugar, SaaS fornece aos usuários softwares na forma de serviços, geralmente padronizadas e com certas customizações enquanto softwares tradicionais fornecem aplicações personalizadas.

Em segundo lugar, eles adotam modelos de preços distintos. Usuários de softwares tradicionais compram o software. No modelo de SaaS, os usuários pagam cada vez que utilizam os serviços. Em terceiro lugar, eles empregam métodos de entrega diferentes. Software tradicionais são instalados onde os usuários o utilizam. SaaS estão localizados em um provedor e os serviços são entregues através da Internet ou rede privada.

O modelo de SaaS está relacionado ao modelo da Arquitetura Orientada a Serviços (SOA), o que causa uma dificuldade em distinguir os dois modelos, já que estes estão no nível conceitual. Muitos profissionais de TI consideram o mesmo conceito e este equívoco estes os dois termos ocasiona vários problemas, principalmente de projeto. Laplante et al. [9] afirmam que “a diferença entre SaaS e SOA é que o primeiro é um modelo de entrega de software e o segundo é um modelo de construção de software”.

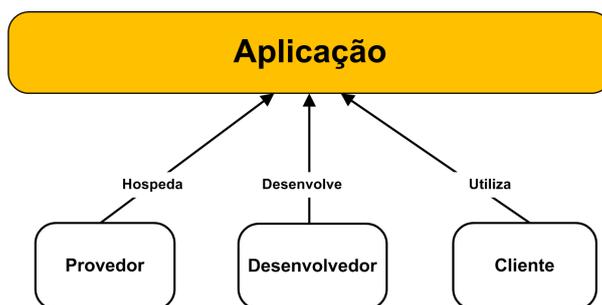
SaaS entrega software como serviços e utiliza uma cobrança baseada no uso. No modelo de SaaS, ocorre uma separação entre o proprietário do software e o usuário. Os serviços são prestados sob demanda para os usuários através da Internet ou uma intranet. No modelo SOA, os elementos constitutivos do software são unidades do sistema reutilizáveis ou serviços. Funções são separadas em unidades distintas ou serviços. Os desenvolvedores podem acessar um conjunto de serviços SOA, combinar e reutilizar os componentes disponíveis do sistema para construir novas aplicações de software. Apesar das diferenças, SaaS e SOA se complementam, pois SOA fornece uma coleção de tecnologias para o desenvolvimento de SaaS.

## Papéis

Para compreender melhor o ambiente de SaaS, pode-se introduzir três papéis em um cenário típico de SaaS, como mostra a Figura 2.2. O provedor é responsável por disponibilizar, gerenciar e monitorar toda a estrutura para a solução, deixando os desenvolvedores e usuários finais sem esses tipos de responsabilidades. Os desenvolvedores utilizam os recursos fornecidos e provêm serviços para os usuários finais. Os atores podem assumir

vários papéis ao mesmo tempo de acordo com os interesses. Esta organização em papéis ajuda a definir os atores e os seus diferentes interesses.

Por exemplo Salesforce.com oferece sua aplicação de CRM como um serviço, portanto, agindo como um provedor de SaaS, bem como um desenvolvedor SaaS. Além disso, o Salesforce.com permite outras empresas desenvolverem SaaS e executá-los na sua plataforma, agindo como um provedor de SaaS que hospeda as aplicações de SaaS de terceiros.



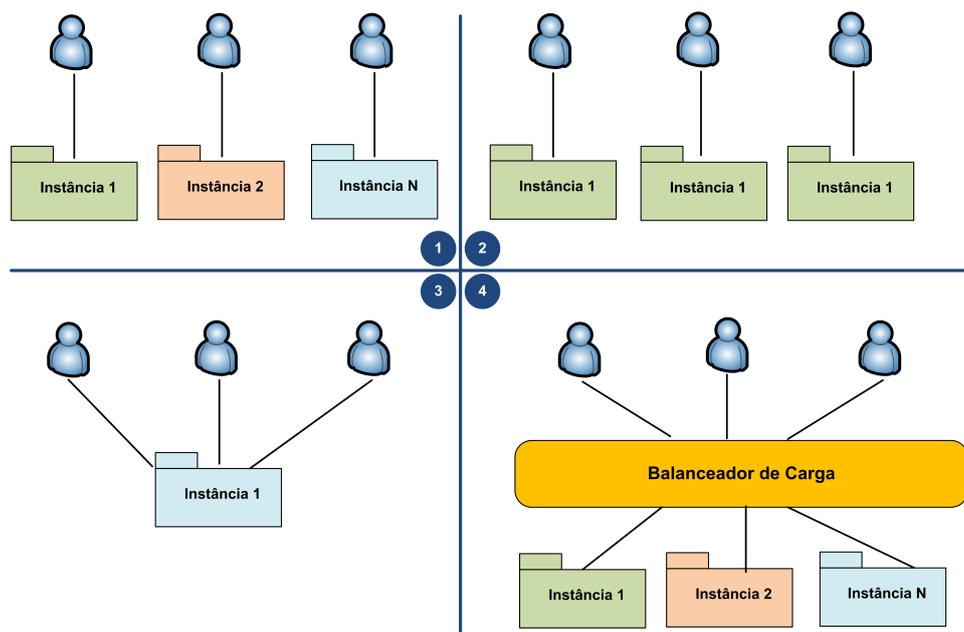
**Figura 2.2** Papéis em um cenário SaaS

## Multi-Inquilino

Um ponto importante no modelo de SaaS é o apoio para vários inquilinos. A fim de explorar economias de escala, ou seja, permitir que um provedor ou desenvolvedor de SaaS oferecer a mesma instância de um aplicativo SaaS para vários inquilinos, a aplicação deve ser multi-inquilino consistente. Multi-inquilino consciente significa que cada inquilino pode interagir com o aplicativo como se ele fosse o único usuário da aplicação. Em especial, um inquilino não pode acessar ou visualizar os dados de um outro inquilino.

A forma de organização dos inquilinos e das aplicações ajuda a determinar a qualidade de uma solução multi-inquilino. De acordo com o nível de implementação do conceito de multi-inquilino, pode-se definir a maturidade do SaaS. A Figura 2.3 mostra o modelo de maturidade SaaS, sobre a evolução do suporte multi-inquilino [7]. Em cada quadrante é possível observar as diversas em relação ao suporte multi-inquilino: ad-hoc/personalizado configurável, configurável e eficiente para vários inquilinos e escalonável, configurável e eficiente para vários inquilinos, respectivamente.

No primeiro quadrante, existe um software destinado para cada inquilino. Isso



**Figura 2.3** Modelo de maturidade SaaS, sobre a evolução do suporte multi-inquilino.

garante o atendimento das demandas do usuário, mas custo elevado devido a grande customização e disponibilidade de recursos individuais. O segundo quadrante apresenta uma única solução destinada para todos os inquilinos separadamente. Neste caso, não há nenhuma customização presente, garantindo menor custo de manutenção.

No terceiro quadrante, existe uma única solução para todos os inquilinos e, diferente do quadrante dois, esse solução é única e disponível para todos. Neste caso, uma solução multi-inquilino, onde ocorre total compartilhamento de recursos. No quarto quadrante, o atendimento é diferenciado para inquilinos que exigem elevada demanda de recursos, havendo uma carga balanceada na infra-estrutura do provedor da solução SaaS.

A fim de permitir aplicações multi-inquilinos, a aplicação deve ser dividido em duas partes diferentes. A primeira parte da aplicação descreve os artefatos que são fixos e podem ser usados por todos os inquilinos. A segunda parte descreve a parte flexível, correspondente as configurações e denominada “metadados”. Em relação aos dados, estes podem ser gerenciados de diversas formas, discutidas ao final deste texto.

## 2.3 DESENVOLVIMENTO DE SAAS

Muitos aspectos distintos devem ser considerados no desenvolvimento de SaaS. Esta seção destaca aspectos de integração, configuração e personalização de perspectivas do software.

Os usuários de SaaS podem acessar software legado, o que requer integração entre aplicações existentes e SaaS. A integração também pode ocorrer quando um usuário utiliza diferentes soluções de SaaS. A integração ocorre nas três principais camadas de aplicações: interface com o usuário, lógica de negócio e parte de dados [10]. Por exemplo, para a integração da interface do usuário, pode-se utilizar *Single Sign On*, que permite que os usuários acessem todas as interfaces após um login único e *mash-up* para que os usuários possam acessar soluções de SaaS ou dados de software legado [11].

De acordo com um estudo sobre a tendência de adoção de soluções de SaaS realizado pelo IDC [12], em 2004, mais de 50 % dos entrevistados enfatizou a melhor integração com aplicações existentes. Um relatório da AMR Research [13], em 2005, apontou que mais de 70% do total de entrevistados na pesquisa esperam que as soluções de SaaS possam ser integradas com os softwares legados e outras soluções de SaaS que eles utilizam ou tenham planos de utilizar. Alguns dos produtos que tratam de questões de integração de SaaS são o *Salesforce.com AppExchange* [3], *IBM SaaS showcase* [14], *Jamcracker* [15] e *OpenKapow* [16].

Em [10] os autores abordam SaaS como serviços web e afirmam que a maioria dos requisitos não-funcionais dos serviços web também existem em soluções de SaaS, tais como segurança, privacidade, qualidade de serviço (QoS) e nível de acordo de serviço (SLA). Os autores definem um processo de desenvolvimento para projetos de integração de SaaS contemplando revisão de processos, projeto de integração, integração da implementação, migração de dados, testes, execução no ambiente de produção e monitoramento. Com isso, eles introduzem um framework de integração de SaaS como uma extensão da descrição de serviços web. Por fim, os autores concluíram que na integração de SaaS, a maioria dos requisitos funcionais podem ser tratados com tecnologias atuais de integração SOA e os requisitos não-funcionais podem ser tratados através da extensão das tecnologias de integração existentes.

A configuração e personalização são questões cruciais no desenvolvimento de SaaS, visto que estas são normalmente projetadas e desenvolvidas com funcionalidades padronizadas. Entretanto, os usuários solicitam alterações de forma a cumprir as neces-

sidades dos seus negócios. Dessa forma, as alterações devem ser realizadas através de um processo fácil de configuração e personalização.

A maior parte das aplicações corporativas enfrenta dificuldades neste contexto. Estas devem ser adaptadas de forma a atender os diferentes requisitos dos usuários. Contudo, é claro que o desenvolvedor de SaaS não pode destinar recursos para desenvolver e manter uma versão do código da aplicação para cada usuário individualmente. Portanto, uma solução para este problema influencia o sucesso de um produto baseado em SaaS. Quanto maior a complexidade de um SaaS, maior será os esforços necessário para sua adaptação. Por isso, as principais soluções de SaaS são sistemas de CRM, RH e serviços de email.

Software simples precisam de pouca modificações para utilização de multi-inquilinos. O webmail é um SaaS com funções relativamente simples, já que os usuários geralmente necessitam apenas adaptar o serviço, por exemplo, definir parâmetros e tamanho do armazenamento. Entretanto, com o aumento de complexidade das funcionalidades do software, os esforços para adaptação dificultam no suporte para servir um cliente específico.

As duas principais abordagens utilizadas para resolver o problema de mudanças de requisitos são a configurações e a personalização. A principal diferença entre elas é que a configuração não envolve alterações no código-fonte, enquanto a personalização necessita de alterações no código-fonte. Configuração utiliza parâmetros pré-definidos para alterar as funcionalidades do software. Para as funcionalidades não atendidas pelo escopo pré-definido, a personalização será necessária. Como a personalização provoca alterações no código-fonte, está requer mais recursos, aumentando os custos e conseqüentemente o ciclo de desenvolvimento do software. Dessa forma, o desenvolvimento de SaaS deve evitar a personalização usando configurações para atender às exigências dos usuário e ampliar os limites dos itens configuráveis.

Sun et al. [17] esclarecem as definições e o significado de configuração e personalização de soluções de SaaS, além de desenvolverem um modelo de competência e um framework baseado em uma metodologia. A metodologia proposta envolve cinco etapas: 1) compreensão do ambiente 2) definição da estratégia 3) avaliação da competência 4) identificação de brechas e ações e 5) priorização e execução. Na primeira etapa, os desenvolvedores devem investigar o ambiente de configuração e personalização do SaaS, focando principalmente nos requisitos dos usuários. Na segunda etapa, os desenvolve-

dores elaboram um plano de configuração para atender os usuários e prováveis pontos de personalização. Na terceira etapa, os desenvolvedores avaliam seus níveis de competência de configuração e personalização do SaaS. Na quarta etapa, as brechas na competência são identificadas para orientar na definição de ações. Na etapa final, os desenvolvedores definem as prioridades e execução as ações.

## 2.4 ARQUITETURA ORIENTADA A SERVIÇOS

A Arquitetura Orientada a Serviços (SOA), está sendo bastante utilizado como forma de software na indústria. SOA permite que novas aplicações sejam criadas através da combinação dos serviços, que podem ser novos ou desenvolvidos a partir de funcionalidades de aplicações existentes. Uma vez criado, um serviço pode ser reutilizado ou re combinado na construção de diferentes sistemas. Neste sentido, SOA foi desenvolvido no intuito de prover ao software uma arquitetura flexível, adaptável, agilidade e extensível.

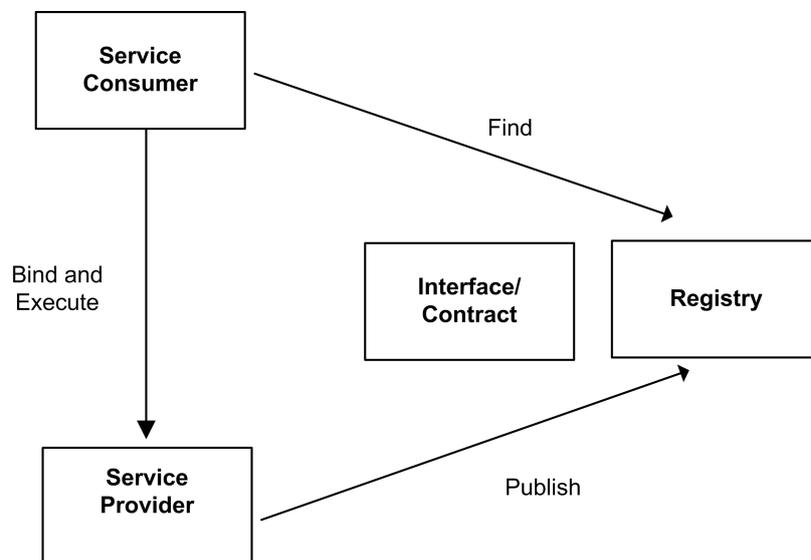
Além destes aspectos, SOA é considerada como uma boa prática na Engenharia de Software [18]. Algumas empresas, como a IBM, despertaram interesses em SOA e desenvolveram ferramentas específicas para o desenvolvimento de software utilizando esta arquitetura. Entretanto, SOA ainda impõe alguns desafios e necessita de muita pesquisa tanto nas áreas da indústria quanto científica.

Em [18] foi realizada uma pesquisa com alunos da Universidade de Los Andes onde relataram alguns problemas para o aprendizado de SOA e suas tecnologias. Eles fizeram um experimento onde expuseram aulas de SOA em uma disciplina de graduação e colocaram os alunos para construir uma solução SOA. Entretanto, contestaram que o tempo de um semestre é curto para o aprendizado. SOA é composto de várias tecnologias, implementações e padrões, o que torna bastante complexo a implementação de soluções nesta arquitetura.

Um aspecto importante é que SOA auxilia no processo de migração do software. Com isso, os projetistas ou desenvolvedores devem levar em consideração a identificação e mineração de serviços existentes nos sistemas legados, a integração de serviços individuais e/ou compartilhados, custo para aquisição de uma infra-estrutura para abrigar uma solução SOA, o desenvolvimento de aplicações para serviços, desenvolvimento de políticas de segurança e análise de desempenho, escalabilidade e outros requisitos de QoS (*Quality of Service*) [19].

Em uma SOA, um serviço é uma função de um sistema computacional que é disponibilizado para outro sistema. Um serviço pode operar de forma independente ou em conjunto com outros serviços por meio da composição de serviços. Para que um serviço possa trabalhar em conjunto com outros, é necessário que este serviço possua uma interface de comunicação bem definida. Geralmente, a comunicação entre o sistema cliente e aquele que disponibiliza o serviço é realizada através de WebServices. Os WebServices são componentes que permitem às aplicações enviar e receber dados em um formato padrão, como por exemplo, XML.

Uma SOA pode ser representada pelo processo chamado “*find-bind-execute paradigm*”. Este processo define o ciclo que envolve a utilização de SOA. Um serviço em uma SOA necessita ser registrado por meio de um registro de serviços (*Service Provider*). Antes da execução o cliente sabe apenas qual interface deve ser implementada para satisfazer as suas necessidades. O cliente acessa o registro de serviços por meio de um protocolo específico, requisitando alguma implementação para uma determinada interface. O registro de serviços responde dando uma identificação para o servidor que implementa tal interface. Finalmente, o cliente acessa o serviço fazendo uso de seu identificador e recebe os resultados. A Figura 2.4 ilustra esse processo.



**Figura 2.4** Esquema do paradigma find, bind e execute

O termo SOA expressa um conceito onde aplicativos são disponibilizados como serviços em uma rede de computadores de forma independente e se comunicando por meio de padrões. Portanto, uma implementação de SOA pode utilizar qualquer tecnologia padronizada baseada na Web. A maior parte das implementações de SOA se utilizam

de WebServices, SOAP, REST e WSDL (*Web Services Description Language*). SOAP (*Simple Object Access Protocol*) é um protocolo padronizado para troca de informações estruturadas entre plataforma distribuída usando como base o XML (*eXtensible Markup Language*). Já WSDL trata-se de um documento XML que descreve o serviço, especifica como acessá-lo e quais operações ou métodos estão disponíveis. Uma UDDI (*Universal Description, Discovery and Integration*) atua como um meio de realizar uma especificação que define um serviço de registro para WebServices. Os provedores podem utilizar UDDI para publicar os serviços que eles oferecem.

A integração de serviços é outro aspecto importante em uma solução SOA. Muitas organizações necessitam integrar seus sistemas com outros, seja da mesma organização ou externos. Entretanto, é necessário levar em consideração que pode haver uma heterogeneidade de tecnologias na construção de serviços, e ainda que a QoS exigida pelos sistemas possa ser descumprida [19]. Para isso, o uso de SLA pode surgir como uma alternativa na negociação do consumo de um serviço com garantia de QoS. Um dos principais problemas em aberto neste contexto é mensurar um esforço para a integração e quais são as diversas partes do sistema que compõe o custo.

A integração entre vários serviços, onde cada serviço pode ser uma parte de um sistema, é que forma um processo de negócio. Em SOA, estes serviços são conectados a um barramento central chamado de ESB (*Enterprise Service Bus*). Esse barramento possui uma serie de conectores que permitem a comunicação com diferentes tecnologias, desde SOAP (webservices), CORBA, RMI e tecnologias legadas. Por esses exemplos, podemos lembrar que WebServices é uma possível tecnologia utilizada para integração, talvez a mais recente, mas não a única. Além disso o correto seguindo a arquitetura SOA é que essa integração seja feita através do ESB, e não entre sistemas.

Para gerenciar e executar uma aplicação SOA é necessário ter uma infra-estrutura SOA. Uma infra-estrutura SOA consiste em diversas peças que suporta os principais aspectos de SOA, incluindo segurança, administração, orquestração, virtualização etc. A forma de modelar processos de negocio da arquitetura SOA é utilizando BPEL (*Business Process Execution Language*). BPEL é uma linguagem baseada em XML onde podemos definir toda a orquestração dos serviços e regras do processo. Existem várias ferramentas visuais que permitem criar processos em BPEL e impedem que a pessoa trabalhe diretamente no XML.

Para auxiliar a compreensão das diferenças e semelhanças entre SOA e SaaS, La-

plante et al. [9] apresentam um estudo. De acordo com os autores, SOA é um modelo que possui uma lista de potenciais serviços a serem usados em um sistema de software a ser construído. Já o SaaS possui uma lista de possíveis serviços a serem distribuídos. É importante ressaltar que SaaS não significa que o software é entregue com um único serviço, visto que este pode ser entregue com vários serviços. SOA representa um modelo de arquitetura que descreve os padrões de interação entre os componentes constituintes de serviço ao passo que SaaS descreve os padrões de interação entre os elementos constitutivos que não são necessariamente serviços.

Tanto SOA como SaaS necessitam identificar uma tecnologia, tais como serviços web, para realizar a interação entre as informações do sistema. Linguagens e protocolos também devem ser identificados. Dessa forma, SOA e SaaS são dependentes de plataforma. Para construir um software baseada em SOA, o desenvolvedor deve escolher uma plataforma de serviços web e para construir SaaS, o desenvolvedor deve decidir qual plataforma utilizar na execução dos serviços. SOA está relacionada diretamente a coordenação, comunicação e colaboração entre serviços e SaaS. Já SaaS trata da gestão da coordenação, comunicação e colaboração entre os seus componentes internos. A Tabela 2.1 mostra um resumo deste estudo por meio de um comparativo entre SOA e SaaS.

<b>Perspectiva</b>	<b>SOA</b>	<b>SaaS</b>
<i>Objetivo/escopo</i>	Lista de serviços possíveis para uso	Lista de serviços possíveis para distribuir
<i>Modelo de negócio</i>	Lista de serviços de negócios para uso	Lista de serviços de negócios para fornecer
<i>Modelo de informações do sistema</i>	Modelo de interação dos componentes do serviço	Modelo de interação de componentes
<i>Modelo de tecnologia</i>	Modelo de interação dos componentes do serviço dependentes de tecnologia e plataforma	Modelo de interação dos componentes dependentes de tecnologia e plataforma
<i>Representação detalhada</i>	Lista de linguagens dependentes de tecnologia e protocolos usados (tais como UDDI, SOAP, XML, WSDL) e serviços atuais usados	Arquitetura publish/subscribe e facilidades de notificação; lista de linguagem dependente da tecnologia, protocolos e serviços usados (caso haja)
<i>Funcionamento do sistema</i>	Comunicação, coordenação e colaboração entre serviços	Comunicação, coordenação e colaboração entre componentes

**Tabela 2.1** Comparação entre SOA e SaaS

## 2.5 DESAFIOS

### 2.5.1 Construção de SaaS

A construção de software com um serviço é um grande desafio na consolidação deste tipo de software [1]. O software deve ser flexível de forma a permitir a configuração e customização. No desenvolvimento baseado em serviço, o processo de seleção e descoberta de serviços deve ser simples e permitir a alteração de um serviço em tempo de execução. A SOA tem auxiliando na especificação de serviço, proporcionando transparência na implementação e minimizando o impacto de alterações nos sistemas de software [20]. Entretanto, a construção de SaaS utilizando SOA ainda apresenta limitações, principalmente no suporte ao modelo multi-inquilino. Dessa forma, as atuais técnicas para a construção de SaaS devem ser melhoradas, assim como o desenvolvimento de novas técnicas.

### 2.5.2 Integração de Serviços

Os usuários e os desenvolvedores de software têm interesse que seus serviços possam acessar outros serviços [10]. Entretanto, a quantidade de tecnologias envolvidas é muito grande, o que torna difícil padronizar as diversas *interfaces* e serviços. Com isso, a integração de serviços é um grande desafio, no sentido de oferecer interoperabilidade entre sistemas heterogêneos para prover funcionalidades mais complexas e interação entre os serviços.

A integração também deve considerar que os serviços de terceiros irão fornecer os requisitos funcionais e não-funcionais de acordo com o contrato definido. Quando o software é fornecido como um serviço, o SLA deve ser monitorado. Esse problema se torna ainda mais complicado no caso de um serviço composto por outros serviços. Nesse caso, a qualidade final do serviço dependerá da qualidade de cada serviço [21]. Assim, o uso de tecnologias de integração de dados, serviços e linguagens devem ser empregadas e adaptadas para prover a integração e acelerar a utilização de software como um serviço.

### 2.5.3 Modelo de Negócio

Embora pesquisas em computação tenham investigado vários modelos econômicos de infra-estrutura computacional durante a última década, o modelo de software como um serviço apresenta uma abordagem mais aplicada aos negócios e relacionada ao custo. Isso ocorre porque o usuário não compra o software e sim o serviço que fornece o software. Dessa forma, o usuário pode pagar pelo uso do serviço de algumas formas. Por exemplo, o usuário pode contratar um serviço por meio de uma assinatura de serviços básicos e ser cobrado um valor fixo independente da quantidade de acessos ou na quantidades de usuários registrados no serviço utilizado. Outra alternativa seria o usuário pagar apenas pelo uso com base no número de acessos ou de dados manipulados. Assim, o desafio é desenvolver um modelo de negócio flexível e que atenda as necessidades dos diferentes usuários [8].

## 2.6 CONCLUSÃO

Este capítulo apresentou uma introdução a SaaS, destacando suas principais características e arquitetura. Também foram apresentados aspectos relacionados aos papéis e ao desenvolvimento de SaaS.

## CAPÍTULO 3

# BANCO DE DADOS COMO UM SERVIÇO

Este capítulo descreve o banco de dados como um serviço, um exemplo de SaaS. São apresentados os requisitos destes sistemas, alguns trabalhos relacionados e, ao final, é apresentando uma discussão sobre estes sistemas e problemas em aberto.

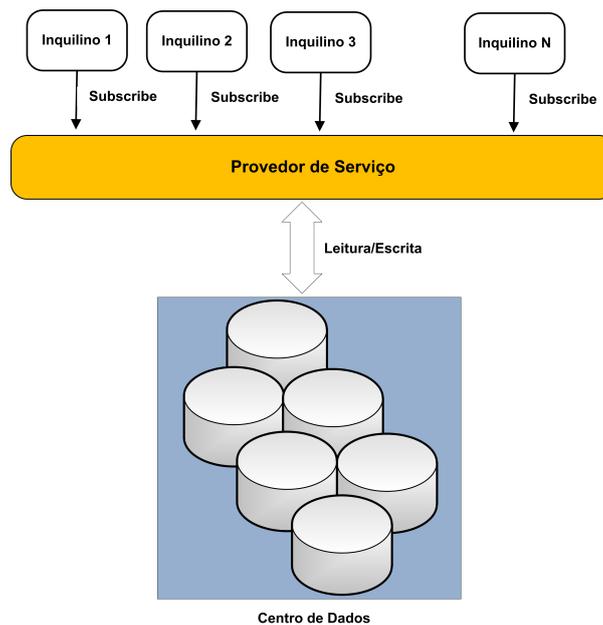
### 3.1 INTRODUÇÃO

Um grande impacto foi causado na indústria de software ao prover um Sistemas de Bancos de Dados como Serviços (DaaS) e diversos desafios de pesquisa foram impostos a comunidade de banco de dados, incluindo, por exemplo, segurança, gerenciamento de recursos compartilhados e a extensibilidade. Devido a estas necessidades, os DaaS estão surgindo como um novo paradigma para a gestão de dados em ambientes corporativos, em que um prestador hospeda um banco de dados e o fornece como um serviço [22].

Neste novo paradigma de gestão de dados, o usuário utiliza o serviço de dados por meio de diversas funcionalidades como por exemplo a configuração das bases de dados, esquemas, carga de dados no serviço de dados e interfaces padronizadas de interação com a base. As atividades de gerenciamento dos aplicativos de banco de dados e dos custos são transferidos do usuário para o provedor de serviços. A Figura 3.1 exibe a organização entre usuários e provedor de DaaS.

De acordo com a figura, cada inquilino contrata os serviço fornecido pelo provedor. Este provedor mantém um conjunto de banco de dados hospedados, em geral, em centros de dados. O provedor deve garantir aspectos de disponibilidade, desempenho e qualidade do serviço definidos em SLA.

Do ponto de vista crítico essa organização oferece uma série de benefícios aos usuários, pois estes dispõem de um ambiente altamente escalável, disponível e rápido. Além disso, não há preocupação de manter a infra-estrutura de hardware e software para fornecer o serviço de dados. Um outro ponto importante é que o provedor de



**Figura 3.1** Banco de Dados como um Serviço

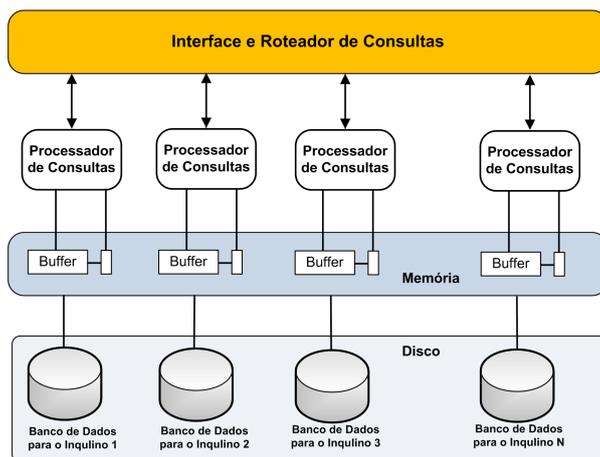
serviços garante a QoS requerida pelo usuário. A própria arquitetura distribuída fornece confiabilidade no tocante a replicação de recursos.

Um sistema de banco de dados multi-inquilino deve oferecer esquemas que sejam flexíveis em dois aspectos [23]. Primeiro, deve ser possível estender o esquema base para suportar múltiplas versões do aplicativo, por exemplo, para regiões geográficas distintas. Uma extensão pode ser privada para um inquilino individualmente ou compartilhada por vários inquilinos. Em segundo lugar, deveria ser possível evoluir de forma dinâmica o esquema base e suas extensões, enquanto o banco de dados está em execução. Evolução de uma extensão deve ser totalmente “self-service”: o provedor de serviço não deve ser envolvido; caso contrário os custos operacionais serão muito altos.

Jacobs e Aulbach [24] afirmam que um sistema de banco de dados pode multi-inquilino pode compartilhar máquinas, processos e tabelas. De acordo com [25], existem três abordagens para a construção de um banco de dados multi-inquilino de acordo com o tipo de recurso compartilhado. A seguir, cada uma destas abordagens é apresentada e as vantagens e desvantagens são discutidas.

### **Banco de Dados Independentes e Instância de Banco de Dados Independente**

Nesta abordagem os inquilinos compartilham apenas o hardware. O provedor de serviços executa instâncias do banco de dados independente para servir inquilinos independentes, de forma similar a instâncias de um sistemas de banco de dados convencional. Cada inquilino cria seu próprio banco de dados e armazena seus dados através da interação com uma instância dedicada do banco de dados. A Figura 3.2 ilustra esta abordagem.



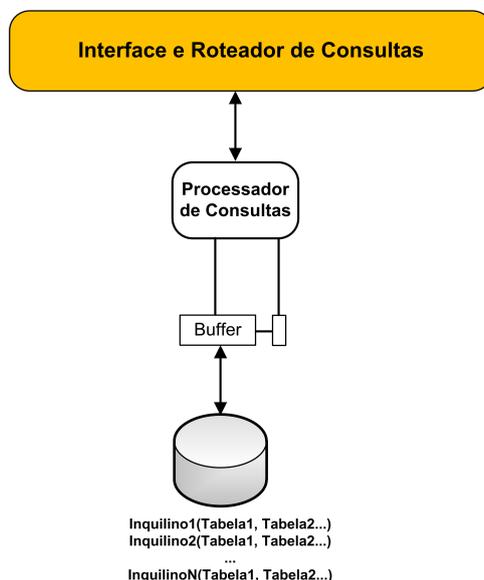
**Figura 3.2** Banco de Dados Independentes e Instância de Banco de Dados Independente

Esta abordagem é a mais simples de implementar um banco de dados multi-inquilino, sendo inteiramente construída em cima dos atuais banco de dados sem nenhuma extensão. Ela fornece um bom nível de isolamento dos dados e segurança. No entanto, esta abordagem pode apresentar um alto custo de manutenção. Para gerenciar diversas instâncias de banco de dados, o provedor de serviço precisa realizar muito trabalho de configuração. Outro ponto importante a escalabilidade, já que o número de instâncias de banco de dados cresce linearmente com o número de inquilinos. Técnicas de virtualização podem ajudar a minimizar alguns problemas desta abordagem, visto que estas técnicas otimizam os recursos de hardware e facilitam a configuração e administração de sistemas de banco de dados.

### Tabelas Independentes e Instância de Banco de Dados Compartilhados

Nesta abordagem, os inquilinos compartilham o hardware e instâncias de banco de dados. O provedor de serviços mantém uma grande base de dados compartilhada e serve todos os inquilinos. Cada inquilino utiliza esquemas privados do banco de dados compartilhado. Cada nome de estrutura privada possui um identificador para indicar quem é o

proprietário da tabela. Quando uma consulta é submetida, esta é reescrita de forma a identificar o nome das tabelas modificadas e obter as respostas corretas. Esta reescrita pode ser feita por um roteador de consultas. A Figura 3.3 mostra esta abordagem.



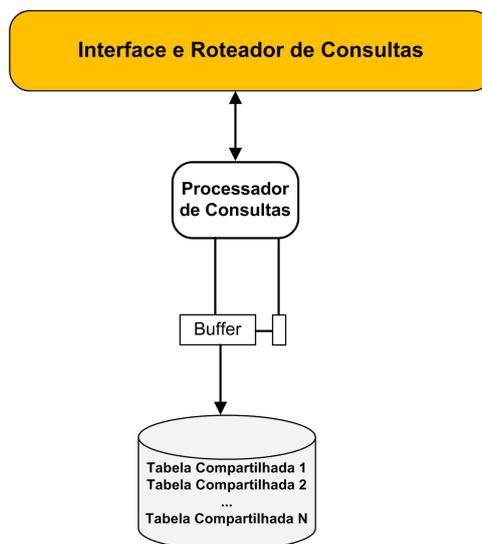
**Figura 3.3** Tabelas Independentes e Instância de Banco de Dados Compartilhados

Esta abordagem reduz o custo de manutenção de gerenciamento de diferentes instâncias. Entretanto, o número de estruturas privadas no banco de dados compartilhado cresce linearmente com o número de inquilinos. Assim, a escalabilidade desta abordagem está limitada ao número de estruturas que o SGBD pode manipular e da memória disponível.

### Tabelas Compartilhadas e Instância de Banco de Dados Compartilhados

Nesta abordagem os inquilinos compartilham tabelas e instâncias do banco de dados. Na fase de configuração do serviço, o provedor de serviços inicializa o banco de dados compartilhado criando tabelas vazias de acordo com o esquema base. Os inquilinos armazenam suas tuplas em tabelas compartilhadas anexando um identificador a cada tupla, que indica a qual inquilino a tupla pertence e define os atributos não utilizados com NULL. Para recuperar as tuplas na tabela compartilhada, um roteador de consultas é usado para reformular as consultas de acordo com o identificador. A Figura 3.4 ilustra esta abordagem.

Nesta abordagem, o provedor de serviço mantém apenas uma instância de banco



**Figura 3.4** Tabelas Compartilhadas e Instância de Banco de Dados Compartilhados

de dados simples, o que reduz o custo de manutenção. Além disso, o número de tabelas do banco de dados é determinado pelo esquema base, sendo independente do número de inquilinos, o que fornece melhor escalabilidade.

Entretanto, esta abordagem apresenta algumas desvantagens, tais como a necessidade de indexação e otimização, já que os inquilinos compartilham as mesmas tabelas, mas apresentam requisitos diferentes. Também não existem estudos detalhados referentes aspectos de isolamento do banco de dados e segurança neste contexto.

## 3.2 REQUISITOS

Os requisitos de banco de dados com um serviço estão muito associados aos usuários, provedores e suas aplicações. Em [26] é apresentada uma lista de requisitos de um banco de dados como um serviço da perspectiva do usuário e do provedor, conforme apresenta a Tabela 3.1.

Da perspectiva do usuário, a principal necessidade é um serviço de banco de dados com uma interface muito simples que não necessite de ajuste e administração. Trata-se de uma melhoria em relação a soluções tradicionais que requerem soluções de provisionamento, seleção de SGBDs, instalação, configuração e administração. O usuário quer um bom desempenho, expressa em termos de latência e vazão, que é independente de tamanhos da base de dados e de mudanças de carga. Esta é atualmente uma tarefa difícil que

<b>Requisitos do Usuário</b>	
<i>U1</i>	API simples com pouca configuração e administração (ex. sem tuning)
<i>U2</i>	Alto desempenho (ex. vazão, escalabilidade)
<i>U3</i>	Alta disponibilidade e confiança (ex. hot stand-by, backup)
<i>U4</i>	Acesso fácil à características avançadas (ex. snapshot, evolução de esquema, mineração de dados)
<b>Requisitos do Provedor</b>	
<i>P1</i>	Atender o SLA do usuário (ex. pontencialmente sob carga de trabalho dinâmica)
<i>P2</i>	Limitar hardware e custo de energia (ex. multiplexação intensiva)
<i>P3</i>	Limitar custo de administração (ex. custo com pessoal)

**Tabela 3.1** Requisitos para Banco de Dados como Serviço

exige uma análise ampla do pessoal de TI, software caro e atualizações de hardware. Alta disponibilidade é outro requisito fundamental, que normalmente é oferecido pelos bancos de dados tradicionais, mas exige cuidados de configuração e manutenção. Finalmente, as características avançadas de gerenciamento do banco de dados, tais como snapshot, evolução de esquema, mineração de dados, devem estar prontamente disponível e de fácil utilização.

Da perspectiva do provedor, o objetivo deste é atender aos acordos de níveis de serviço, apesar dos dados e mudanças na carga de trabalho. O sistema deve ser eficiente, utilizar eficazmente os recursos de hardware. O modelo de serviço proporciona a oportunidade de fazer isso, por multiplexação de cargas de trabalho e ajustando dinamicamente a alocação de recursos. Finalmente, a quantidade de tarefas de administração deve ser minimizada. Para tanto, ferramentas sofisticadas de análise de carga de trabalho e centralização do gerenciamento de muitos bancos de dados podem ser utilizados.

### 3.3 SISTEMAS DE GERENCIAMENTO DE DADOS COMO UM SERVIÇOS

Segundo [25] os Bancos de Dados como Serviços (DaaS) podem ser classificados em duas categorias: serviços de banco de dados terceirizados e sistemas de banco de dados relacionais estendidos. Os serviços terceirizados de banco de dados consistem em projetar um sistema que disponibiliza um serviço de banco de dados. Neste caso, o usuário não administra a infra-estrutura necessária para utilizar o serviço. Os usuários acessam mecanismos para criar e acessar suas bases em um sítio que é gerenciado pelo provedor. O foco dos serviços de banco de dados terceirizados é fornecer um serviço para gerenciar os dados e, em geral, questão de compartilhamento da base de dados e recursos não são tratadas.

Os sistemas de bancos de dados relacionais estendidos são projetados para suportar bases de dados e esquemas compartilhados. Nestes sistemas, as relações possuem muitos atributos que podem vir a serem nulos para muitas tuplas. A razão para isto é que em alguns casos esses atributos são necessários para identificar o proprietário das tuplas. Além disso, alguns campos servem de controle para o próprio sistema.

O NetDB [27] é um exemplo de sistema utilizado como serviço terceirizado de banco de dados. Este sistema fornece um banco de dados como serviço incluindo ferramentas para o desenvolvimento de aplicativos, criar e carregar tabelas e execução de consultas e transações para usuários por meio da Internet. Com isso, o usuário não preocupa com atualização do sistema, manutenção da infra-estrutura de armazenamento ou softwares adicionais, necessita somente de um navegador web. O NetDB foi projeto para ser portátil e possuir um alto desempenho, escalabilidade e interoperabilidade. Para isso, foi implementado em uma arquitetura em três camadas utilizando a plataforma Java.

Em [23] é exibido uma nova técnica de mapeamento de esquemas para bancos de dados multi-usuários chamado Chunk Folding. O banco de dados é um misto para aplicativos específicos convencionais e um vasto conjunto fixo de estruturas genéricas chamadas Chunk Tables. Em uma tupla de uma Chunk Table existem alguns Chunks. Em um Chunk pode-se armazenar uma tabelas convencionais ou Chunk Tables. Essa técnica pode ser implementada sobre qualquer banco de dados relacional. Experimentos mostram que essa técnica genérica é eficaz no gerenciamento de grandes quantidades de dados e na manutenção da estrutura de dados utilizada neste contexto.

O SHAROES [28] é uma plataforma que fornece a capacidade de compartilhamento de dados sobre um ambiente de armazenamento terceirizado. Este trabalho está inserido na segunda classificação de DaaS. O SHAROES utiliza um provedor de serviço de armazenamento (SSP) para persistir os dados em um SSP remoto gerenciado por meio de um ambiente de rede. O SSP prover o serviço de armazenamento sob demanda que pode crescer ou diminuir conforme a necessidade. Como exemplos de SSP pode-se citar o Amazon S3 e o SUN Grid. O SHAROES fornece um rico compartilhamento semântico de dados sobre um SSP sem interferir no SSP com relação a confidencialidade e controle de acesso. O SHAROES é composto de uma ferramenta de migração responsável pelo transporte dos dados entre o SSP e o modelo, uma ferramenta de servidor de dados SSP para servir dados de um sítio remoto, e um sistema de arquivos para acesso local dos dados de um SSP.

Em [29] uma estratégia para publicação de dados e sistema de compartilhamento. Trata-se de um framework para suportar esses dois benefícios. O sistema utiliza objetos de dados descritos com meta-dados criado pelos usuários. Além disso, esse sistema armazena, classifica esses objetos e prover várias interfaces de consultas e navegação. O tipo de dados que o usuário utiliza para definir pode ser de qualquer categoria e possuir diversas propriedades. Com isso, essa estratégia consegue abranger diversos domínios de aplicativos. Além disso, o framework inclui tipos de dados categorizados que podem ser distribuídos para prover melhor desempenho, indexação para tipos de dados e um processador de consultas com diversas funções.

### 3.4 DISCUSSÃO

De acordo com as alternativas para construir um sistema de banco de dados multi-inquilino, pode-se perceber que é necessário uma reescrita da consulta em muitos casos. Isso ocasiona um esforço adicional para o sistema, o que pode comprometer a escalabilidade. Com a abordagem de tabelas privadas, os dados de cada inquilino podem ser manipulados de forma independente. Assim, pode-se elaborar soluções que permitam utilizar um esquema base, comum para todos os inquilinos, e extensões do esquema, gerenciadas diretamente por cada inquilino.

A evolução do esquema quanto o sistema de banco de dados está em execução é um grande desafio, visto que outros inquilinos podem estar acessando a mesma instância do banco de dados ou uma tabela compartilhada. Assim, um sistema de banco de dados multi-inquilino deve ter formas para tratar isto, como por exemplo, criar esquemas temporários para o processo de evolução [23].

Um aspecto relevante, mas pouco abordado pelos trabalhos, é a distribuição dos dados, realizado por meio do particionamento e replicação. Com isso, os dados dos inquilinos estarão alocados em diferentes máquinas. A distribuição deve garantir que os dados dos inquilinos não sejam perdidos e que a recuperação seja simples. Para a recuperação, pode-se estender a linguagem de consulta de forma a permitir o acesso considerando a distribuição dos dados dos inquilinos.

Em relação à replicação, esta deve ser implementada em uma arquitetura não compartilhada e, assim, garantir a privacidade dos dados. A replicação pode ajudar a melhorar a disponibilidade, no caso de falha, o desempenho, através do acesso concorrente

aos dados, além de contribuir para a escalabilidade, aspecto fundamental em um sistema de banco de dados multi-inquilino.

Novas estruturas propostas tais como o BigTable permitem organizar os dados em colunas [30]. Neste caso, todos os valores de uma coluna são serializados em conjunto, os valores da coluna seguinte também, e assim por diante. Com isso, os dados semelhantes, de mesmo formato, podem ser agrupados, auxiliando no armazenamento e na recuperação de informação. Estas novas estruturas fornecem alternativas para a construção de sistemas de banco de dados multi-inquilino. Estas estruturas podem ser incorporadas aos sistemas existentes e, assim, melhor o suporte multi-inquilino.

Com base nesta discussão, pode-se perceber que a construção de um sistema de banco de dados multi-inquilino ainda é um desafio. Isso ocorre principalmente em função dos aspectos de compartilhamento, evolução do esquema e características de distribuição. A comunidade científica tem apresentado soluções para alguns dos problemas apresentados e acredita-se que, em breve, novas propostas possam ajudar a consolidar o modelo multi-inquilino em sistemas banco de dados.

## 3.5 PROBLEMAS EM ABERTO

### 3.5.1 Implantação e Provisionamento

O processo de implantação de um banco de dados como um serviço ainda não é bem definido, visto que não existem padrões para a disponibilização de serviços de banco de dados. É importante definir uma infra-estrutura que possa fornecer o banco de dados como um serviço, assim como tecnologias que permitam ajudar o gerenciamento do serviço. A infra-estrutura deve contemplar aspectos de distribuição, tais como particionamento e replicação.

Vale ressaltar, no sistema de banco de dados como um serviço, diversos inquilinos acessam o sistema e compartilham recursos, o que pode interferir no desempenho do sistema. Dessa forma, o provisionamento dos recursos deve ser eficiente, já que as cargas de sistemas de banco de dados com um serviço são muito variáveis, visto que os inquilinos podem acessar com maior frequência o sistema em determinados momentos, por exemplo, no final do dia. Com a ampla utilização de ambientes virtualizados, altamente compartilhados, a questão do provisionamento torna-se determinante.

### 3.5.2 Qualidade do Serviço de Dados

No modelo de SaaS, os dados geralmente estão na Internet e estes são acessados por diferentes usuários, cada um com um requisito distinto. Estes requisitos são expressos por meio de acordos de níveis de serviços (SLA). Um ponto importante é garantir a qualidade do serviço de dados independente da forma de acesso. Por exemplo, o provedor deve fornecer um serviço com garantias para um usuário que acessa este serviço através de um desktop ou através de um dispositivo móvel. O provedor deve ser capaz de identificar a requisição e, caso necessário, alocar mais recursos de forma a garantir o SLA com o usuário.

### 3.5.3 Segurança

Questões relativas a segurança são fundamentais no desenvolvimento e disponibilização de banco de dados como um serviço [22]. Criptografia de dados é uma solução para garantir a privacidade dos dados nas bases de dados contra acessos não autorizados. No entanto, as técnicas de criptografia têm implicações significativas de desempenho do processamento de consultas em bancos de dados. Dessa forma, alternativas para a integração de técnicas de criptografia com bancos de dados devem ser investigadas e *trade-offs* devem ser investigados.

A complexidade computacional de criptografar e descriptografar dados para executar uma consulta aumentam o tempo de resposta da consulta. Em [22] é apresentado uma abordagem segura e escalonável para preservar a privacidade. Em vez de criptografia, que é computacionalmente caro, é utilizada uma estratégia de distribuição dos dados em vários sítios do provedor e técnicas para acessar as informações de forma secreta e compartilhada. Este pode ser um direcionamento importante para futuras pesquisas .

### 3.5.4 Modelo de Banco de Dados Multi-Inquilino

A utilização de uma arquitetura multi-inquilino ajuda os provedores de serviços a reduzir os custos de hardware, software e serviços de manutenção e, conseqüentemente, o valor do contrato do serviço para o usuário. Dessa forma, é possível fornecer serviços para um número maior de usuários.

Entretanto, o acesso de inquilinos diferentes que compartilham sistemas de banco de dados, instâncias ou tabelas prejudica o desempenho das consultas destes inquilinos [23]. Caso a degradação não seja tolerada, o inquilino pode estar disposto a não manter o contrato com o serviço. Dessa forma, um desafio é desenvolver uma arquitetura eficiente e técnicas para maximizar a quantidade de inquilinos, com baixo custo e garantindo o desempenho dentro dos limites estabelecidos.

### **3.6 CONCLUSÃO**

Este capítulo descreveu o banco de dados como um serviço. Os requisitos e trabalhos relacionados foram descritos e uma discussão foi realizada. Além disso, foram abordadas alguns problemas em aberto. O capítulo a seguir apresenta as conclusões deste trabalho.

## CAPÍTULO 4

# CONCLUSÃO

Esta monografia apresentou um estudo sobre software como um serviço. Este estudo abordou os principais conceitos sobre software como um serviço e destacou aspectos do gerenciamento de dados como um serviço. Também foram apresentados e discutidos alguns desafios neste contexto.

### 4.1 CONCLUSÃO E TRABALHO FUTUROS

Software como um Serviço está se tornando cada vez mais comum nas empresas. Isso decorre do aumento significativo do número de provedores de serviços e da ampla utilização de SOA. SaaS tem sido utilizado para distribuir aplicações por meio da Internet com o objetivo de atender uma grande quantidade de usuários. Essas aplicações são utilizadas com base em uma infra-estrutura compartilhada de forma a reduzir os custos e obter economia em escala.

Este trabalho apresentou os principais aspectos sobre SaaS, destacando o gerenciamento de dados com um serviço. Foi possível perceber que SaaS encontra-se em expansão e a sua adoção pelas empresas aumenta constantemente. Foram discutidos alguns desafios importantes sobre software como um serviço, tais como construção de SaaS, integração de serviços, modelo de negócios segurança, qualidade do serviço de dados, entre outros. Estes desafios geram oportunidades de pesquisa que devem ser superados, de forma que software como um serviço seja amplamente aceito e utilizado.

Como trabalhos futuros pretendemos, primeiramente, estudar detalhadamente sobre a construção de software com um serviço. Assim, será possível especificar e projetar SaaS. Em seguida, pretendemos compreender melhor a parte referente a qualidade de um serviço de dados e a arquitetura de banco de dados multi-inquilino. Por fim, com as informações obtidas destes estudos, iremos implementar um serviço de gerenciamento de dados.

## REFERÊNCIAS BIBLIOGRÁFICAS

- [1] Mark Turner, David Budgen, and Pearl Brereton. Turning software into a service. *Computer*, 36(10):38–44, 2003.
- [2] Ralph Mietzner and Frank Leymann. Generation of bpm customization processes for saas applications from variability descriptors. In *SCC '08: Proceedings of the 2008 IEEE International Conference on Services Computing*, pages 359–366, Washington, DC, USA, 2008. IEEE Computer Society.
- [3] Salesforce. "Salesforce", 2009. <http://www.salesforce.com/>.
- [4] Eugene Ciurana. *Developing with Google App Engine*. Apress, Berkely, CA, USA, 2009.
- [5] Ralph Mietzner, Frank Leymann, and Mike P. Papazoglou. Defining composite configurable saas application packages using sca, variability descriptors and multi-tenancy patterns. In *ICIW '08: Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services*, pages 156–161, Washington, DC, USA, 2008. IEEE Computer Society.
- [6] Microsoft. *Software as a Service (SaaS): An Enterprise Perspective*, 2010. <http://msdn.microsoft.com/en-us/library/aa905332.aspx>.
- [7] Microsoft. *Architecture Strategies for Catching the Long Tail*, 2010. <http://msdn.microsoft.com/en-us/library/aa479069.aspx>.
- [8] Dan Ma and Abraham Seidmann. The pricing strategy analysis for the "software-as-a-service" business model. In *GECON '08: Proceedings of the 5th international workshop on Grid Economics and Business Models*, pages 103–112, Berlin, Heidelberg, 2008. Springer-Verlag.
- [9] Phillip A. Laplante, Jia Zhang, and Jeffrey M. Voas. What's in a name? distinguishing between saas and soa. *IT Professional*, 10(3):46–50, 2008.

- 
- [10] Wei Sun, Kuo Zhang, Shyh-Kwei Chen, Xin Zhang, and Haiqi Liang. Software as a service: An integration perspective. In *Service-Oriented Computing - ICSOC 2007*, pages 558–569, Berlin, Heidelberg, 2009. Springer-Verlag.
- [11] Mashups and the Web as Platform. "*Mashups and the Web as Platform*", 2010. <http://www.programmableweb.com>.
- [12] IDC. Software as a service in the mid-market: Adoption trends and customer preferences. Technical report, IDC, 2004.
- [13] AMR. Software as a service: Managing buyer expectations as we pass the tipping point from novelty to necessity. Technical report, AMR Research, 2005.
- [14] IBM. "*IBM SaaS Showcase*", 2010. [www.ibm.com/software/showcase/saas](http://www.ibm.com/software/showcase/saas).
- [15] Jamcrack. "*Jamcrack*", 2010. <http://www.jamcracker.com/>.
- [16] Openkapow. "*openkapow*", 2010. <http://www.openkapow.com/>.
- [17] Wei Sun, Xin Zhang, Chang Jie Guo, Pei Sun, and Hui Su. Software as a service: Configuration and customization perspectives. In *SERVICES-2 '08: Proceedings of the 2008 IEEE Congress on Services Part II*, pages 18–25, Washington, DC, USA, 2008. IEEE Computer Society.
- [18] Nicolas Lopez, Rubby Casallas, and Jorge Villalobos. Challenges in creating environments for soa learning. In *SDSOA '07: Proceedings of the International Workshop on Systems Development in SOA Environments*, page 9, Washington, DC, USA, 2007. IEEE Computer Society.
- [19] Liam O'Brien, Paul Brebner, and Jon Gray. Business transformation to soa: aspects of the migration and performance and qos issues. In *SDSOA '08: Proceedings of the 2nd international workshop on Systems development in SOA environments*, pages 35–40, New York, NY, USA, 2008. ACM.
- [20] Ahmed Elfatatry. Dealing with change: components versus services. *Commun. ACM*, 50(8):35–39, 2007.
- [21] Florian Rosenberg, Philipp Leitner, Anton Michlmayr, Predrag Celikovic, and Schahram Dustdar. Towards composition as a service - a quality of service driven approach. In *ICDE '09: Proceedings of the 2009 IEEE International Conference on*

- 
- Data Engineering*, pages 1733–1740, Washington, DC, USA, 2009. IEEE Computer Society.
- [22] Divyakant Agrawal, Amr El Abbadi, Fatih Emekci, and Ahmed Metwally. Database management as a service: Challenges and opportunities. *Data Engineering, International Conference on*, 0:1709–1716, 2009.
- [23] Stefan Aulbach, Torsten Grust, Dean Jacobs, Alfons Kemper, and Jan Rittinger. Multi-tenant databases for software as a service: schema-mapping techniques. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1195–1206, New York, NY, USA, 2008. ACM.
- [24] Dean Jacobs and Stefan Aulbach. Ruminations on multi-tenant databases. In *BTW*, volume 103 of *LNI*, pages 514–521. GI, 2007.
- [25] Mei Hui, Dawei Jiang, Guoliang Li, and Yuan Zhou. Supporting database applications as a service. In *ICDE '09: Proceedings of the 2009 IEEE International Conference on Data Engineering*, pages 832–843. IEEE Computer Society, 2009.
- [26] Carlo Curino, Evan Jones, Yang Zhang, Eugene Wu, and Sam Madden. Relational cloud: The case for a database service. Technical report, MIT-CSAIL-TR-2010-014. Computer Science and Artificial Intelligence Laboratory, MIT, USA, 2010.
- [27] H. Hacigumus, B. Iyer, and S. Mehrotra. Providing database as a service. In *International Conference on Data Engineering*, pages 29–38, Los Alamitos, CA, USA, 2002. IEEE Computer Society.
- [28] Aameek Singh and Ling Liu. Sharoes: A data sharing platform for outsourced enterprise storage environments. In *ICDE*, pages 993–1002. IEEE, 2008.
- [29] Beng Chin Ooi, Bei Yu, and Guoliang Li. One table stores all: Enabling painless free-and-easy data publishing and sharing. In *CIDR 2007, Conference on Innovative Data Systems Research*, pages 142–153, 2007.
- [30] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. In *OSDI '06: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association.