DESENVOLVIMENTO AUTOMÁTICO DE APLICAÇÕES E PLATAFORMAS DE TRABALHO EM NUVENS COMPUTACIONAIS

Hélder Pereira Borges[1,2], José Neuman de Souza[2], Bruno Schulze[3], and Antonio Roberto Mury[3]

[1]Federal Institute of Education, Science and Technology of Maranhão, São Luís, Brasil

[2]Department of Computing, Federal University of Ceará, Fortaleza, Brasil

[3]Department of Computing, National Laboratory for Scientific Computing, Petrópolis, Brasil

helderpb@hotmail.com, {bruno.schulze, neuman.souza, a.roberto.m}@gmail.com

RESUMO

A computação em nuvem tem se estabelecido nos últimos anos como uma importante plataforma de pesquisa. No cenário atual, tarefas como obtenção, compartilhamento, manipulação e exploração de grandes quantidades de dados são comuns, porém demandam de muitos recursos.

A computação em nuvem pode contribuir com este cenário a medida que pode disponibilizar de forma indefinida recursos de processamento, memória, armazenamento, dentre outros para utilização imediata.

Uma das propostas do ambiente de nuvens, o software como serviço - SaaS, apresenta em seu paradigma uma série de vantagens, tanto para os provedores quanto aos consumidores de serviços.

A necessidade de criação de softwares de forma rápida sem comprometer a qualidade a um custo acessível é uma grande preocupação da indústria de software. A utilização de modelos, bem como a sua transformação em código ou outros modelos mais refinados são o cerne para propostas relacionadas ao desenvolvimento dirigido a modelos, incluindo a Model Driver Architecture - MDA.

A computação em nuvem, de forma um tanto quanto generalizada, utiliza o conceito de acordos em nível de serviço (Service Level Agreements – SLA) para

controlar o uso de recursos computacionais advindos de um provedor.

A proposta deste trabalho é combinar estes paradigmas e tecnologias, com a intenção de prover aplicações de forma automática, ou ainda em um cenário alternativo, prover, também de forma automática, uma plataforma de trabalho para os clientes, que corresponde a uma combinação de um sistema operacional com aplicações disponíveis para uso, sendo as duas possibilidades controladas por meio de SLA.

Palavras Chaves: Computação em nuvem, SaaS, MDA, SLA.

SUMÁRIO

LISTA DE FIGURAS

1. INTRODUÇÃO

Este trabalho apresenta um estudo sobre desenvolvimento de software para computação em nuvens, onde se utilizará uma abordagem dirigida a modelos, sendo proposto que o controle do uso dos recursos computacionais seja feito a partir de um SLA.

O propósito deste trabalho é descrever uma abordagem para construção de software baseada em modelos, a MDA, suas características e os desafios inerentes ao desenvolvimento de SaaS, bem como aspectos relevantes, requisitos e funcionalidade de um SLA.

Por fim, será proposto uma abordagem que combina todas estas MDA, SaaS e SLA, com a intenção de oferecer aplicações e ambientes de forma automática no contexto de nuvens computacionais.

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

1.1 MOTIVAÇÃO

A proposta básica da computação em nuvem é que a provisão de recursos computacionais seja de responsabilidade de empresas especializadas ou que seja abstraído o fornecimento dos mesmos em níveis que apenas especialistas venham se preocupar em gerenciá-los e mantê-los, e ainda os mesmo sejam disponibilizados como serviços [Carr, 2008].

Neste contexto, a provisão de recursos precisa ser vista em várias camadas, onde cada camada representa um gênero específico de recursos que podem ser providos de diferentes formas, no caso deste trabalho relaciona-se a SaaS, que apresenta uma série de desafios.

Considerando que, [SELIC, 2003], [FOWLER, 2004], comprovam que o uso de modelos no desenvolvimento de software é relevante e ainda que os mesmos

elevam o nível de abstração do desenvolvimento de sistemas, ajudando no planejamento e entendimento dos mesmos, conclui-se como pertinente a adoção desta abordagem.

Um grande desafio no ambiente de nuvens é controlar de forma eficiente o uso dos recursos computacionais disponíveis, além de certificar-se que aquilo foi prometido aos clientes está sendo de fato cumprido.

A utilização de SLA pode delimitar de forma clara a responsabilidade de cada parte, bem como definir, de forma inequívoca, os requisitos que estão sendo contratados, para que seja possível, em futuras avaliações, verificar-se o cumprimento dos contratos e a eficiência dos recursos.

1.2 OBJETIVO E CONTRIBUIÇÃO

Este trabalho apresenta um estudo sobre software como serviço, modelagem dirigida a modelos e ainda contratos com relação a serviços, onde o objetivo principal é descrever a estrutura conceitual de cada uma destas partes, abordando seus conceitos e particularidades básicas, para por fim apresentar uma proposta com objetivo de disponibilizar ambientes e aplicações de forma automática nas nuvens.

A principal contribuição deste trabalho é a proposta de um ambiente que disponibiliza aplicações e plataformas de trabalho, em uma nuvem, de forma transparente e automática para os clientes, a partir de uma simples seleção de funcionalidades.

1.3 ESTRUTURA DO TRABALHO

Os próximos capítulos deste documento estão estruturados da seguinte forma:

- Capítulo 2: apresenta e discute a abordagem dirigida a modelos denominada de MDA, descrevendo suas propostas essências e estruturas participantes. Esta abordagem é fundamental para o contexto de disponibilização automática de serviços ou plataformas.
- Capítulo 3: descreve o software como serviço, destacando aspectos

pertinentes a sua adoção, características, desafios e vantagens. Construir um SaaS de forma automática e transparente, baseado nos conceitos do capítulo 2, nos leva a necessidade de entender mais profundamente a sua proposta.

- Capítulo 4: aborda o contexto de contratos em relação a serviços, descrevendo suas características, funcionalidades e conceitos essenciais. No contexto do ambiente que será proposto no capítulo 6, SLA acrescenta regras e garantias para os serviços que serão oferecidos.
- Capítulo 5: descreve publicações que possuem relação com o tema deste trabalho.
- Capítulo 6: apresenta uma proposta sobre o desenvolvimento automatizado de software e ambientes de trabalho.
- Capítulo 7: apresenta as conclusões e possíveis trabalhos futuros.

2. MODEL DRIVEN ARCHITECTURE - MDA

2.1 INTRODUÇÃO

O desenvolvimento de software tem apresentado grandes desafios, visto que este processo tem se tornado cada vez mais complexo, menos produtivo e mais dispendioso financeiramente. Por isto a busca por abordagens que agreguem vantagens neste contexto é uma constante.

A necessidade de criação de softwares de forma rápida sem comprometer a qualidade a um custo menor intencionando-se a boa utilização dos recursos computacionais e novas tecnologias é uma grande preocupação da indústria de software. Sendo assim, a portabilidade, interoperabilidade, preparação para manutenções e documentação se apresentam como fatores cruciais na qualificação de um sistema.

Um software para ser considerado bom e conseguir grande longevidade deve apresentar uma série de qualidades, dentre elas uma de suma importância está relacionada com a premissa sempre presente no mundo do desenvolvimento de software, as mudanças, e especificamente relaciona-se a adaptabilidade tanto em relação aos requisitos quanto ao ambiente em que a aplicação está inserida, principalmente quando considerado o contexto de produtividade.

A utilização de modelos, bem como a sua transformação em código ou outros modelos mais refinados são o cerne para propostas relacionadas ao desenvolvimento baseado em modelos, incluindo MDA.

Modelos elevam o nível de abstração do desenvolvimento de sistemas, ajudando no planejamento e entendimento dos mesmos, sendo que a importância do uso de modelos no desenvolvimento de software é um fato comprovado [SELIC, 2003], [FOWLER, 2004].

Uma abordagem que cresceu bastante a partir dos anos 2000, é a da geração automática de código fonte a partir do modelo do sistema, e o desenvolvimento dirigido por modelos segue esta proposta, promovendo o modelo a artefato principal do desenvolvimento de software em detrimento do código.

A Arquitetura Dirigida a Modelos (Model Driven Architecture – MDA) especificada pelo OMG (Object Management Group) [OMG, 2003] é uma das iniciativas para esta abordagem que além de pregar o modelo como artefato principal, possibilita a geração automática de código a partir dos modelos e ainda introduz o conceito de separação entre modelo e plataforma de suporte, agregando com isto independência da solução computacional em relação a tecnologia de implementação, o que seguramente melhora a portabilidade, interoperabilidade e reusabilidade [KLEPPE, 2003] .

A proposta da MDA é promover o desenvolvimento de modelos que sejam independentes dos detalhes de implementação, criando-se, portanto, sistemas mais flexíveis e de fácil portabilidade, agregando ganhos imponentes em relação a qualidade do produto devido o foco dos desenvolvedores estar aplicado as regras de negócio, e em relação a produtividade dado a automatização intrínseca a proposta da abordagem.

2.2 A CONTEXTUALIZAÇÃO

Esta seção relata os desafios relativos ao processo de desenvolvimento de software advindos de uma exigência cada vez mais exacerbada dos usuários dada a disponibilidade de maiores recursos computacionais e de diversas tecnologias emergentes.

2.2.1 O Problema e seus Desafios

A Engenharia de Software, conforme [PRESSMAN, 2001], tem suas origens na engenharia de sistemas e de hardware, sendo que ela abrange um conjunto de três elementos fundamentais, os métodos que proporcionam os detalhes de como fazer para construir um software, as ferramentas que proporcionam apoio automatizado ou semi-automatizado aos métodos e por fim, os procedimentos, que constituem os elos que mantém junto os métodos e as ferramentas possibilitando o desenvolvimento racional do software do computador.

Desta forma, possibilita-se ao gerente o controle do processo de desenvolvimento de software e oferece ao profissional uma base para a construção

de software de alta qualidade produtivamente. E ainda segundo [NAUR, 1969], que é considerado como a primeira proposta de definição, a Engenharia de software é a criação e a utilização de sólidos princípios de engenharia a fim de obter software de maneira econômica, que seja confiável e que trabalhe eficientemente em máquinas reais.

Com o desenvolvimento da engenharia de software e a partir dos fundamentos erigidos por Pressman, [PRESSMAN, 2001], em relação ao desenvolvimento de sistemas, surgiram múltiplos métodos que agregam características de modelos distintos intentando a obtenção de melhores resultados, bem como interação entre modelos para proporcionar soluções mais complexas.

Assim sendo, genericamente, a proposta básica da área de informática é produzir sistemas de forma eficaz e eficiente utilizando-se das tecnologias disponíveis para agregar qualidade ao processo e, secundariamente, diminuir o custo dos sistemas e aperfeiçoar as soluções.

O fato é que com o passar dos tempos os softwares evoluem de forma crescente em complexidade e os usuários se tornam cada vez mais exigentes em relação a muitos requisitos, e estas realidades nos levam a um cenário onde o desenvolvimento de software geralmente possui custo elevado e apresenta produtividade que normalmente pode ser definida como baixa, porém sendo justificada por vários fatores, como o fato do código possuir um nível muito mais baixo que o das abstrações utilizadas, além de normalmente ser dependente da plataforma de desenvolvimento, e ainda, o uso da reutilização de objetos com baixa granularidade, bem como as eventuais perdas dos mapeamentos de informações.

Com o passar do tempo, cada vez mais camadas de abstrações foram formalizadas e novas ferramentas tiveram que ser construídas para suportar estes conceitos, em suma, os detalhes estavam cada vez mais sendo escondidos entre cada camada e as linguagens de alto nível ganhavam o potencial de serem mapeadas em linguagens de níveis inferiores, como pode-se observar na figura 2.1.

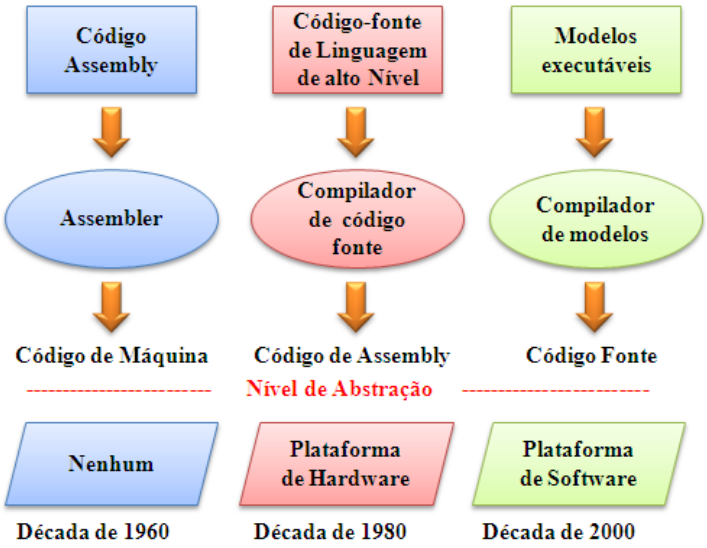

Figura 2.1 – Nível de abstração

Pode-se perceber a partir da figura 2.1que ocorre um aumento substancial do nível de abstração ao longo dos anos e por sua vez da complexidade dos sistemas, sendo que nos primórdios da década de 60 não havia abstração, os sistemas eram desenvolvidos diretamente em linguagem de máquina. Na década de 80 já era possível ocultar características do hardware físico, sendo disponibilizadas plataformas eficientes para execução de aplicações sem haver uma preocupação em como utilizar de forma específica cada hardware disponível. A partir dos anos 2000, vivenciou-se a possibilidade de compatibilização entre diferentes plataformas de forma simplificada.

O nível de reuso também foi alterado de forma significativa no decorrer dos anos como pode ser observado na figura 2.2.

Diferentes implementações de um mesmo algoritmo exigiam um controle mais complexo de correções, pois era necessário identificar se os erros eram no algoritmo ou estavam relacionados com uma implementação específica, além do árduo trabalho de atualização, quando necessária, ser feita em todas as implementações.

Um avanço nos anos 70 foi a utilização de funções, que poderiam ser reaproveitadas sem a necessidade de duplicação do código. Nos anos 80, foram desenvolvidas as linguagens orientadas a objetos trazendo um novo potencial de

agrupamento e organização de funcionalidades, tornado mais fácil o reuso de artefatos já produzidos.

A utilização de frameworks, que disponibilizam um conjunto de funcionalidades e ainda o uso de componentes, que são artefatos auto-contidos, marcou a década de 90. Já a partir dos anos 2000, era possível construir aplicações reutilizando características comuns entre domínios utilizando-se conceitos e funcionalidades compartilhadas.

Pode-se perceber que o processo de reutilização foi amplamente desenvolvido, aumentando-se os ganhos inerentes ao seu uso como o aumento de produtividade e utilização de artefatos já validados em outros domínios ou sistemas.

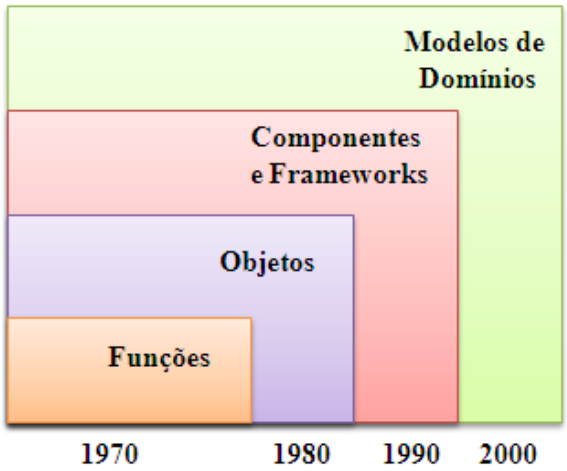

Figura 2.2 – Nível de reuso

Desta forma avanços portentosos podem ser observados, porém dentro deste cenário, podem ser destacados alguns grandes desafios apresentados para o desenvolvimento de software atualmente:

- A crescente complexidade das regras de negócio;
- Desenvolvimento rápido;
- Manter a viabilidade relacionada a qualidade, longevidade e custo pré-estabelecido, visto que os projetos de software se tornaram muito dispendiosos;
- Compatibilidade entre diferentes plataformas;

- Adaptação a novas tecnologias emergentes;
- Comunicação e consistência entre sistemas, legados ou não.
- A mudança freqüente de requisitos, indispensável para satisfação e completo atendimento das necessidades dos usuários;
- Retorno satisfatório do investimento em desenvolvimento.

2.2.2 Uma Solução para o Problema

Em resumo, pôde-se perceber a existência de um grande desafio relacionado a complexidade no desenvolvimento, manutenção e evolução de software, sendo que os paradigmas de orientação à objetos e componentes já não conseguem gerenciar eficientemente esta complexidade e uma abordagem centrada em middlewares também não é suficiente, tornando-se necessária uma mudança de paradigma.

A Arquitetura Orientada a Modelos – MDA, desponta como uma solução emergente para os problemas inerentes ao desenvolvimento de software, isto porque possui uma abordagem centrada no conceito de modelos, no relacionamento entre eles e nas transformações entre estes modelos. O seu principal objetivo é isolar a lógica de negócio da aplicação, da evolução e manutenção da tecnologia, e apresenta como proposta a construção de sistemas de forma rápida, consistente e independente de plataforma.

O desenvolvimento será rápido porque a maior parte do código será gerado a partir das especificações de correspondências e transformações entre modelos, sendo que novos modelos compatíveis com uma plataforma específica são obtidos sem a necessidade de um novo processo de desenvolvimento. Automaticamente também são geradas as conexões para comunicação entre os diversos modelos envolvidos, sendo então o código gerado, que também por ser automático, está menos propenso a erros humanos, garantindo-se desta forma uma maior consistência.

Exemplificando de forma concreta, um documento hipermídia feito para um *middleware* especifico não poderá ser interpretado por um *middleware* de outros padrões, sendo necessário uma tradução manual entre os diferentes padrões, o que

obviamente aumentará o esforço em sua autoria, desperdiçando, portanto, tempo e recursos. Para resolver este problema, [LIMA, 2007], utilizou MDA para automatizar o processo de tradução de um documento hipermídia em diferentes padrões, sendo que a transformação pode ser feita em um nível mais abstrato denominado modelo.

2.3. A HISTÓRIA

O desenvolvimento dirigido a modelos não era uma abordagem nova para o desenvolvimento de software, os especialista em software já se utilizavam de modelos para capturar a arquitetura e definições de sistemas, porém o conceito de MDA definido pelo OMG, Object Management Group, foi uma novidade no mundo da orientação a modelos.

O OMG é um consórcio de empresas, dentre elas (Adobe Systems Inc, AT&T, Boeing, Borland Software Corporation, Ericsson, GNOME Foundation, Hewlett-Packard, Massachusetts Institute of Technology (MIT), Motorola, NASA, Nokia, Oracle, Sun Microsystems, W3 Consortium), organizações e pessoas, ou seja, é amplamente aceito, difundido e organizado, que tem como objetivo a definição de padrões de processos relacionados ao desenvolvimento de software, com a intenção de solucionar problemas de integração de aplicações distribuídas através do fornecimento de especificações de interoperabilidade abertas e independentes de fornecedor, facilitando o reuso de componentes e garantindo a portabilidade.

A situação muito comum, a necessidade de correção em aplicações que estão em uso, bem como o imperativo de integração com outros sistemas, além de alterações na infra-estrutura e nos requisitos, bem como a evolução e criação de novas tecnologias, levou a OMG a criar um padrão de especificação onde os detalhes da implementação e da plataforma foram abstraídos, sendo o foco direcionado apenas para a modelagem das regras de negócio, então em 2001 foi criada a MDA.

Segundo o OMG [MILLER, 2001], MDA define uma abordagem para especificação em sistemas computacionais que separa a especificação das funcionalidades do sistema da implementação destas funcionalidades em uma plataforma tecnológica específica, sendo então definida uma arquitetura baseada em

modelos que fornece um conjunto de diretrizes para estruturar especificações expressas como modelos permitindo que um mesmo modelo com especificações de funcionalidades seja utilizado em múltiplas plataformas através de mapeamentos auxiliares ou com pontos de mapeamentos para plataformas específicas, permitindo com isto que aplicações distintas sejam integradas baseando-se na relação explícita de seus modelos.

Em [OMG, 2003], a portabilidade, a interoperabilidade e o reuso a partir da separação arquitetural de interesses, são apontados como as três metas primárias da MDA.

2.3.1 As Tecnologias Envolvidas

O OMG criou um modelo de objeto padrão para definir o comportamento dos objetos em um ambiente distribuído, o Object Management Architecture (OMA), que utiliza o Common Object Request Broker (CORBA), [OMG, 2008], como componente de comunicação, sendo esta a arquitetura padrão do OMG para estabelecer e simplificar a troca de dados entre sistemas distribuídos heterogêneos, onde os objetos ou componentes de software se comunicam de forma transparente ao usuário, apesar da necessidade de interoperar com outras aplicações desenvolvidas com ferramentas distintas e ainda em outros sistemas operacionais.

O objetivo da interoperabilidade é alcançado pelo CORBA através da definição de um ORB (Object Request Broker), que em computação distribuída é uma peça do middleware que permite aos desenvolvedores fazerem chamadas de um computador a outro através de uma rede, sendo que o mesmo é o um componente chave do OMA sendo a base para construção de aplicações que utilizam objetos distribuídos interoperáveis, habilitando-os a enviar e receber requisições, bem como receber as respostas de suas requisições, ou seja, trafegar os dados e requisições dos objetos do ambiente local para o remoto e vice versa.

A proposta básica do padrão CORBA é eliminar os problemas típicos em sistemas heterogêneos como a dependência de aspectos incompatíveis entre plataformas, provendo a independência de linguagem, de implementação, de arquitetura, de sistema operacional e também de protocolo / transporte dado que o ORB decide dinamicamente que protocolo e transporte devem ser usados.

O OMA [OMG, 2010] incorpora a visão do OMG para o ambiente de componentes de software fornecendo instruções de como padronizar componentes de interfaces baseado na tecnologia de objetos e define a arquitetura sob a qual os objetos locais e remotos devem se comunicar.

Embora aplicações possam ter regras de negócio totalmente diferentes é comum elas compartilharem várias funcionalidades, como um objeto precisar notificar outro quando um evento ocorrer ou a referência a objetos novos ou destruídos serem propagados e, além disto, aplicações dentro de um mesmo domínio de negócios normalmente compartilham uma série de funcionalidades.

O OMA abstrai as funcionalidades comuns de aplicações CORBA em um conjunto de objetos padrão com funções claramente definidas que podem ser acessadas através de interfaces padronizadas pelo OMG que especificam o que o objeto faz, podendo então ser criadas implementações para serviços específicos. Com a utilização do OMA espera-se alcançar uma codificação e implantação mais rápida, uma melhor arquitetura devido o projeto ser desenvolvido em torno de serviços discretos e considerar a divisão em grupos de objetos baseado na funcionalidade e ainda robustez e escalabilidade.

A estrutura de um OMA é composta basicamente por quatro elementos, sendo que a maioria dos **Objetos de Serviços (CORBA Services)** oferece funcionalidades básicas para aplicações com objetos distribuídos onde é executado um serviço que chama bibliotecas de um determinado sistema operacional, embora existam serviços mais complexos, como a segurança, que devido a estreita ligação com a infra-estrutura ORB exigem sua colocação nesta categoria.

Exemplos de CORBA Services são o serviço de nomes, o serviço Object Trader, o serviço de persistência de estado, o serviço de transações, os serviços para gerenciamento do ciclo de vida de objetos remotos e segurança.

Os **objetos de Aplicação** normalmente são customizados para cada aplicação individual e não necessitam de padronização, são os objetos que podem ser considerados visíveis ao nível de aplicação.

Por fim, as Facilidades Comuns que são dedicadas a usuários finais de aplicações e podem ser divididas em **Facilidades CORBA Horizontais**, são

potencialmente utilizadas em vários domínios de negócios, como serviços de e-mail, impressão, interface de usuário, etc.; e **Facilidades CORBA Verticais** que são específicas para um domínio de aplicação onde empresas de um mesmo segmento podem compartilhar objetos.

Após a criação do padrão de interoperabilidade CORBA, a OMG o utilizou quase que exclusivamente na criação de padrões para domínios específicos, porém a partir de 1997 foram apresentadas especificações que não eram baseadas em CORBA.

Dentre elas a Unified Modeling Language - UML, [OMG, 2011], que surgiu para resolver o problema da falta de padronização entre as notações usadas no desenvolvimento de software, sendo uma especificação que define uma linguagem gráfica para visualizar, especificar, construir e documentar os artefatos de um sistema de forma que os relacionamentos entre os componentes sejam melhor compreendidos e visualizados e tendo como objetivo principal descrever qualquer tipo de sistema em termos de diagrama

Outra especificação, o Meta Object Facility (MOF) foi concebido como uma iniciativa da OMG para a padronização da representação e manipulação de meta-modelos, que é um modelo de uma linguagem de modelagem [FAVRE, 2004][OMG, 1997], através de uma linguagem abstrata para especificação, construção e gestão dos meta-modelos independente da tecnologia de implementação.

Posteriormente, o XML Metadata Interchange (XMI) [OMG, 2006a] é definido como padrão para a troca de informações de metadados via Extensible Markup Language (XML) com o propósito de facilitar o intercâmbio entre ferramentas de modelagem UML e repositórios baseados em MOF em ambientes heterogêneos e distribuídos fornecendo um mapeamento entre MOF e UML.

Ainda foi definido o Common Warehouse Metamodel (CWM) [OMG, 2006b], que é um padrão de metadados cujo objetivo é permitir a integração de sistemas de data warehouse, e-business e sistemas de negócios inteligentes em ambientes heterogêneos e distribuídos, através de uma representação e de um formato de troca de metadados.

2.3.2 A Evolução

As especificações descritas na seção anterior foram largamente implementadas pelos fornecedores ao longo dos anos seguintes, agregando aos sistemas desenvolvidos com componentes que suportavam as padronizações já desenvolvidas pelo OMG grandes facilidades em tarefas como a criação de soluções multi-fornecedor e na integração entre aplicativos, gerando com isto boas expectativas em relação a outras especificações.

Até o momento em que o OMG produziu apenas padrões baseados em CORBA a forma de ligação entre os padrões era entendida e mapeada pelo OMA sem grandes problemas, porém com o surgimento acelerado e de certo modo desordenado de novos padrões, foi necessário expandir a visão da arquitetura OMG para dar maior suporte as tecnologias emergentes.

Considerando-se que a vida de um sistema computacional normalmente é longa e nem sempre é possível ou simplório fazer a modificação de sistemas mais antigos com a intenção de que eles venham suportar um ou vários padrões e ainda a crescente necessidade de incorporação de front-ends baseados na web desenvolvidos com interfaces proprietárias, forçam os desenvolvedores / integradores a retomarem atividades pouco produtivas, porém essenciais, para manter os diversos componentes funcionando junto, e ainda é importante destacar a enorme quantidade de trabalho necessária para realizar a modificação e integração destes sistemas visando a utilização de tecnologias emergentes, portanto, existe um limite de interoperabilidade que pode ser alcançado com a criação de um conjunto de interfaces padrão para o desenvolvimento de sistemas.

A partir desta constatação, o OMG expandiu sua visão concernente aos requisitos para suportar a interoperabilidade com especificações para integração durante todo o ciclo de vida dos sistemas, da modelagem do negócio ao projeto do sistema, da construção do componente até a montagem, integração, desenvolvimento, gerenciamento e evolução [MILLER, 2001], sendo esta nova visão incorporada ao desenvolvimento da MDA. A mesma descreve as relações entre os padrões OMG e como usá-los coordenadamente e ainda como esta abordagem auxilia na criação, manutenção e evolução dos padrões.

Desta forma, MDA foi uma proposta para expandir o OMA e não para substituí-lo, incorporando o trabalho feito até então e apontando o caminho para os futuros padrões de integração, oferecendo mais suporte para uma criação rápida e eficaz de novas especificações que integram múltiplos padrões dentro e fora da organização.

A figura 2.3 ilustra a arquitetura da MDA, revelando como os padrões OMG trabalham com MDA e fornecendo uma visão geral onde vários padrões podem ser identificados, além de se perceber três camadas de especificações, onde o núcleo é formado por especificações que seguem os padrões definidos pelo OMG (UML, MOF e CWM) e não levam em consideração características específicas de uma plataforma, sendo totalmente independentes.

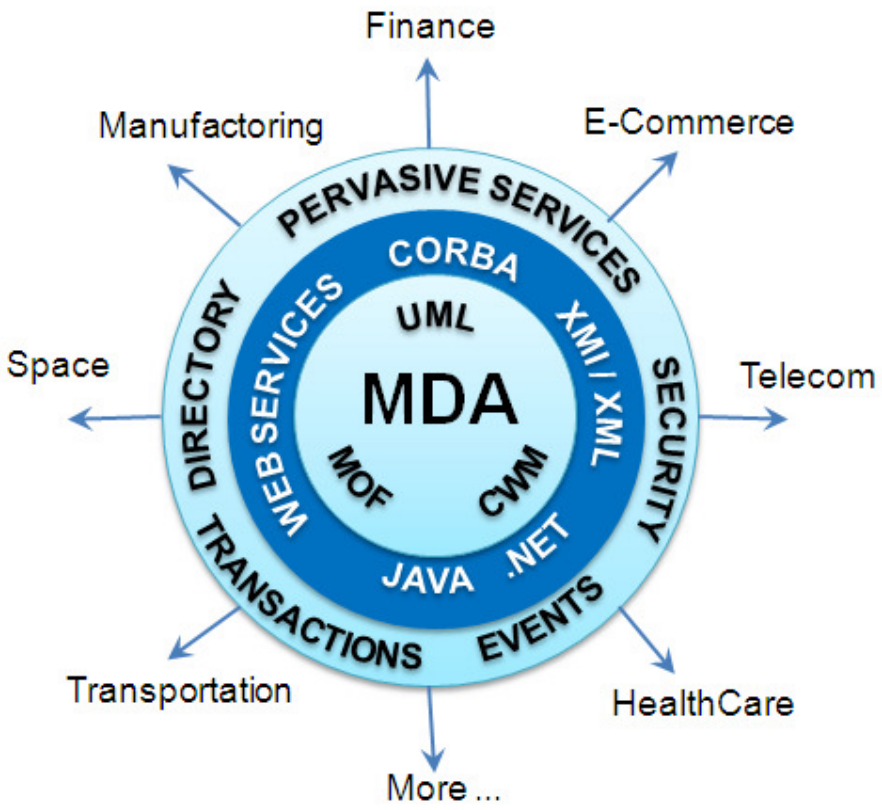

Figura 2.3 - Arquitetura MDA

Na segunda camada encontram-se modelos que já possuem detalhes de uma determinada tecnologia, como *Java*, *.NET*, CORBA, *Web* etc. Na camada mais

externa, são exibidos os serviços que a maioria dos domínios de aplicações necessitam, tais como segurança, persistência, controle de transações, tratamentos de eventos etc. Por fim, observa-se a aplicação da MDA em diferentes domínios.

2.4 ARQUITETURA DIRIGIDA A MODELOS - MDA

A MDA consiste de uma abordagem para especificação de sistemas que separa a especificação das funcionalidades ou lógica de negócio da implementação em uma plataforma específica.

Para alcançar este propósito, foi definido uma arquitetura para modelos que fornece um conjunto de diretrizes para estruturação das especificações, onde a abordagem e os padrões que lhe dão suporte permitem que um mesmo modelo de especificação seja utilizado para múltiplas plataformas através de um mapeamento padrão ou de pontos de mapeamento para plataformas específica, permitindo então que aplicações diferentes sejam integradas por meio de uma relação explícita entre seus modelos, habilitando desta forma a interoperabilidade e a evolução dos sistemas de suporte conforme as tecnologias atuais.

O fato de ser orientada a modelos significa que os artefatos principais do processo de desenvolvimento são os modelos, e eles devem orientar o entendimento, o projeto, a construção, a operação, a manutenção e as modificações do sistema [CALIARI, 2007]. Os modelos podem ser utilizados para gerar os sistemas, scripts de bancos de dados, documentação para o usuário, configurações e quaisquer outros elementos que possam fazer parte do processo de desenvolvimento.

O OMG, conforme [OMG, 2003], define formalmente a MDA como uma abordagem que é alavancada pela idéia de separar a especificação das operações de um sistema dos detalhes de como este sistema usa as potencialidades de sua plataforma, possibilitando que ferramentas ofereçam a especificação de um sistema independente de qualquer plataforma, além da especificação de plataformas bem como a transformação da especificação para uma plataforma particular, tendo, portanto como objetivo principal a portabilidade, a interoperabilidade e a reusabilidade através da separação arquitetural dos interesses.

2.4.1 Conceitos Básicos

Existem muitas expressões que flutuam no ambiente da MDA e a proposta desta seção é fornecer o entendimento contextualizado do significado de alguns termos que estão diretamente envolvidos no cotidiano da MDA.

2.4.1.1 Modelo

É um conjunto de elementos que descreve um sistema [MELLOR, 2004] com grau de abstração maior que o do sistema [KLEPPE, 2003]. Também pode ser definido como uma especificação formal de uma função, estrutura e/ou comportamento de um sistema [MILLER, 2001].

Modelos são representações fidedignas, porém simplificadas e contextualizadas, de algo real, por isto, bons modelos servem como meio de comunicação.

Os modelos freqüentemente apresentam uma combinação de desenhos e texto sendo normalmente representados por linguagens de modelagem, como a UML, que é a linguagem recomendada pelo OMG para trabalhar com MDA.

2.4.1.2 Meta-Modelo e Meta-Meta-Modelo

Um meta-modelo é um modelo de uma linguagem de modelagem que define a estrutura, semântica e restrições para um conjunto de modelos que compartilham sintaxe e semântica comuns [MELLOR, 2004]. Por exemplo, um modelo que utiliza diagramas UML está sujeito a um meta-modelo UML, que descreve como modelos UML podem ser estruturados e que elementos contêm.

O meta-modelo da UML é uma instância da infra-estrutura do MOF [OMG, 2001], sendo este um framework para gerenciar meta-dados e serviços de meta-dados intencionando o desenvolvimento e interoperabilidade de sistemas baseados em modelo, ou seja, o MOF objetiva a criação de meta-modelo, sendo portanto chamado de Meta-Meta-Modelo.

2.4.1.3 Plataforma

É um conjunto de subsistemas e tecnologias que provêem um conjugado de funções

coerentes que podem ser utilizadas a partir de interfaces e padrões específicos onde qualquer aplicação suportada pela plataforma pode utilizar sem se preocupar em como a funcionalidade foi implementada.

Pode-se entender plataforma também como a especificação de um ambiente de execução para um conjunto de modelos [MELLOR, 2004].

A independência de plataforma é uma qualidade que um modelo pode apresentar e representa que o modelo não depende de qualquer característica específica da plataforma.

2.4.1.4 Mapeamento

Dado que modelos podem possuir relacionamentos semânticos com outros modelos, os mapeamentos servem para conectar elementos de um modelo fonte a elementos de um modelo alvo que possuem mesma estrutura e semântica.

Desta forma, o relacionamento um para um descreve um elemento do modelo alvo possui a mesma semântica de um elemento do modelo fonte; já no caso de um para muitos, representa que um conjunto de elementos do modelo alvo possui a mesma semântica de um elemento do modelo fonte; e por fim, a relação de muitos para um significa que um elemento do modelo alvo possui a mesma semântica de um conjunto de elementos do modelo fonte.

Em síntese, um mapeamento provê especificações para que um modelo seja transformado em outro.

2.5 VISÃO GERAL DO PROCESSO DE DESENVOLVIMENTO MDA

A arquitetura MDA define modelos que são básicos para que a sua estrutura funcione corretamente, sendo então possível alcançar os objetivos propostos.

Uma sequência de atividades foi definida para o desenvolvimento de softwares, todavia estas atividades são divididas em etapas, iniciando-se com o levantamento dos requisitos do sistema para então, baseado nestes requisitos, criar-se um Modelo Independente de Computação.

2.5.1 Modelo Independente de Computação - CIM

Este modelo, o Modelo Independente de Computação (Computation Independent Model – CIM), descreve o domínio da aplicação e não possui nenhum detalhe sobre a estrutura e processamento do sistema, sendo um modelo conceitual ou de análise, que identifica as entidades e que pode listar suas propriedades e seus relacionamentos, sem, contudo especificar a interdependência entre estas propriedades, sua execução interna e suas interações.

O CIM exibe o sistema no ambiente que ele vai operar e, portanto ajuda a comunicar exatamente o que o sistema deve fazer sendo que os requisitos representados por ele devem ser mapeados para os modelos seguintes da arquitetura.

2.5.2 Modelo Independente de Plataforma - PIM

O segundo modelo da arquitetura MDA, segundo o OMG [MILLER, 2001], é o Modelo Independente de Plataforma (Plataform Independent Model – PIM) que deve ser gerado a partir do CIM, embora algumas publicações considerem o PIM como o início de um processo MDA, talvez porque normalmente o processo de geração do PIM a partir do CIM seja manual.

O foco deste modelo está em expressar as funcionalidades de negócio e comportamento, escondendo detalhes de plataforma, para exibir apenas a especificação do que não varia com a alteração da plataforma. O propósito desta independência é possibilitar o uso do mesmo PIM em múltiplas plataformas.

Os serviços pervasivos da arquitetura MDA podem ser incorporados ao PIM, e o mesmo pode ser refinado quantas vezes forem necessárias para se chegar ao nível de precisão adequado.

Uma transformação deve ser aplicada ao PIM para que o próximo modelo da arquitetura seja gerado. Esta transformação é o processo de convergir de um modelo para outro seguindo um determinado mapeamento que descreve uma correspondência entre os elementos dos modelos, ou seja, a partir de cada elemento de um modelo fonte a transformação gera os elementos correspondentes no modelo alvo, sendo definida por regras de transformação dentro de ferramentas

de transformação.

2.5.3 Modelo Específico de Plataforma - PSM

O modelo advindo do PIM é o Modelo Especifico de Plataforma (Plataform Specific Model – PSM) que apresenta uma visão focada em uma plataforma, combinando as especificações presentes no PIM com os detalhes relativos ao funcionamento do sistema em uma plataforma específica.

Obviamente se faz necessário a definição de qual ou quais plataformas serão utilizadas na transformação, para então serem construídos os mapeamentos necessários para cada plataforma, sendo para tanto necessário o uso de um modelo de plataforma que provê um conjunto de conceitos que representam as diferentes partes da plataforma, os serviços providos por ela e os diferentes tipos de elementos que podem ser utilizados.

Um exemplo de transformação de PIM para PSM pode ser visto na figura 2.4. Onde percebe-se no PIM a especificação de atributos que são independentes de qualquer plataforma, sendo que no PSM observa-se que estes atributos já contêm características da plataforma Java.

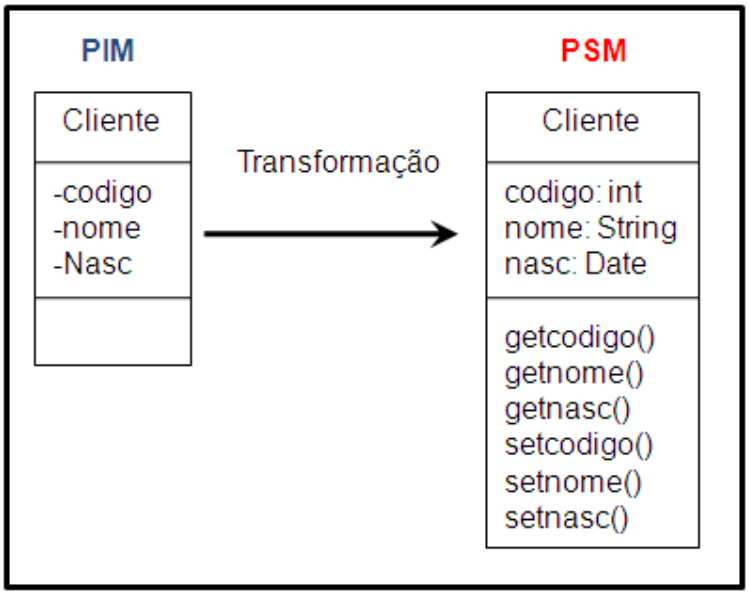

Figura 2.4 – Transformação PIM para PSM

A MDA possibilita que as ferramentas de transformação alterem o PIM em um

PSM ou que seja gerado diretamente o código da aplicação além de que o PSM pode ser refinado em outro PSM ou usado para gerar o código do sistema.

Como resultado da transformação do PIM espera-se além do PSM propriamente dito ou do código, o registro de transformação, que armazena o mapeamento dos elementos do PIM aos elementos correspondentes no PSM descrevendo que parte do mapeamento foi usado por qual parte da transformação, possibilitando desta forma o rastreamento da transformação e possibilitando que a mesma seja bidirecional.

A figura 2.5 retrata o processo completo de transformação que ocorre durante o uso da MDA.

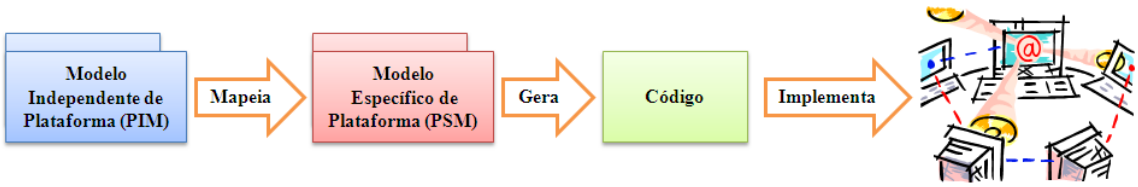

Figura 2.5 – Processo de Transformação MDA

2.6 O CICLO DE VIDA DA MDA

As mesmas fases de desenvolvimento podem ser identificadas tanto nos modelos de desenvolvimento de sistemas tradicionais quanto no modelo da MDA, sendo que as diferenças essenciais se encontram na natureza dos artefatos que são criados durante cada processo de desenvolvimento, sendo estes os modelos formais compreendidos pelo computador.

As imagens 2.6 e 2.7 respectivamente exibem o ciclo de vida tradicional e o ciclo de vida do desenvolvimento com MDA. Os diagramas e textos da figura 2.6 são especificados de acordo com uma arquitetura, logo inovações tecnológicas ou mesmo alterações nas tecnologias podem introduzir incompatibilidades no processo. No caso da figura 2.7, como os artefatos iniciais são independentes de plataforma, a portabilidade é extremamente facilitada garantindo-se um aumento de produtividade dado a utilização da transformação entre os modelos.

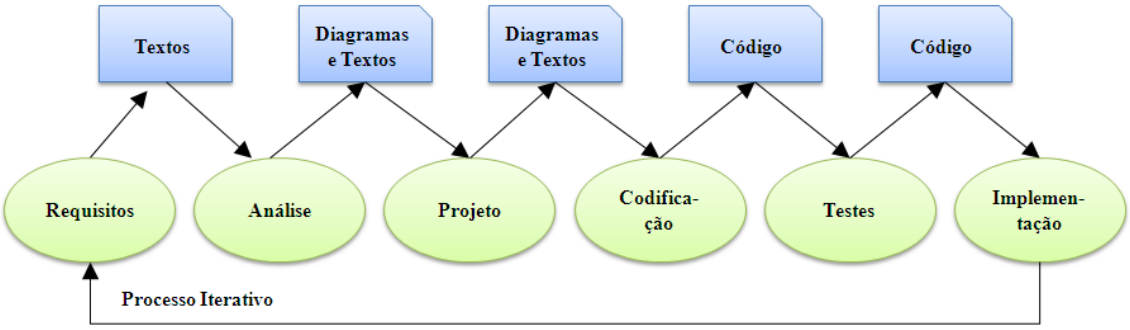

Figura 2.6 – Ciclo de Vida Tradicional

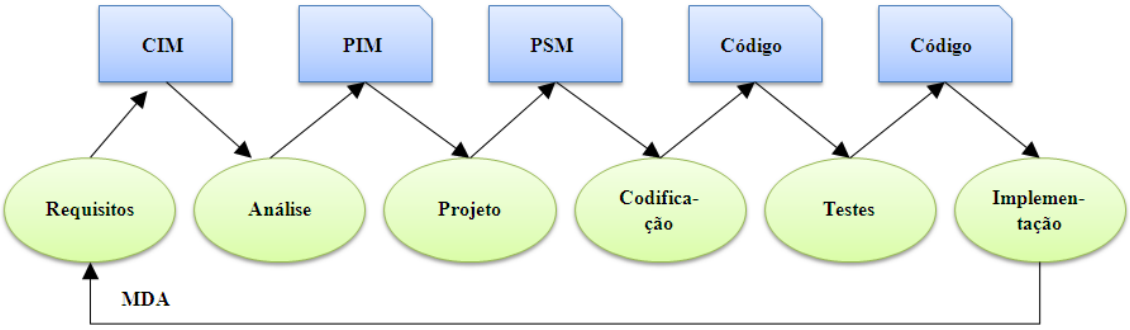

Figura 2.7 – Ciclo de Vida MDA

2.6.1 Comparação de atividades

Considerando o ciclo de vida da MDA, podemos destacar e comparar o que cada profissional envolvido no processo de desenvolvimento de software faz no modelo tradicional e no modelo MDA.

O **Analista de Requisitos** é o responsável por capturar os resultados da análise de requisitos em modelos UML para futura verificação e testes dos modelos gerados no processo de desenvolvimento e ainda reduzir as possibilidades de má interpretação dos resultados da análise de requisitos.

Este papel no processo MDA não muda em relação ao tradicional, é exatamente igual. Apesar dos modelos UML com seus diagramas ajudarem na comunicação e visualização dos requisitos, como esta atividade é baseada em pessoas a sua criação não pode ser automatizada.

O **Analista / Projetista** possui a função de expressar formalmente os requisitos, em termos de diagramas de classe, de estado e métodos é a

responsabilidade deste profissional.

Como estes modelos serão utilizados para transformação em outros modelos ou código a precisão dos mesmos é fundamental de maneira a reduzir a necessidade de modificações para ajuste.

Se forem utilizadas ferramentas com suporte a UML executável, pode-se facilmente demonstrar o comportamento dos modelos aos clientes e consequentemente se obter um retorno mais rápido, agregando uma melhor precisão ao modelo.

É responsabilidade do **Arquiteto** tomar decisões sobre a estrutura geral do sistema, em um processo MDA, continuam a fazê-lo, e esta tarefa envolve selecionar os modelos e mapeamentos entre eles.

O mercado de desenvolvimento de software já disponibiliza uma série de modelos e mapeamentos para reutilização, sendo necessário que os arquitetos apliquem sua experiência e conhecimento no processo de seleção dos modelos e sintonização dos mapeamentos para melhorar o desempenho do sistema.

Os **Analistas** / **Programadores** continuam a desempenhar seu papel praticamente da mesma maneira, porém existem duas grandes diferenças, a primeira está relacionada com a linguagem usada para expressar as abstrações, ao invés de Java, C++, etc deverá ser definida pelo QVT (Query, Views and Transforms) que é a especificação padrão do OMG para frameworks de regras de transformação formal.

A segunda diferença considera que as abstrações criadas pelos desenvolvedores devem exercer grande influencia no sistema, logo, ao invés de criar uma lista para um conjunto de contas, eles devem criar regras para construção de uma lista para qualquer classe que possua o mesmo padrão de acesso que uma conta.

Desta forma, o trabalho fica relacionado com a criação e ajuste dos padrões de geração de código utilizado por ferramentas MDA, gerando-se então, devido a perícia do programador, melhores resultados do que simplesmente desenvolver código manualmente.

De forma geral, as ferramentas MDA menos avançadas podem gerar pelo menos 50% de todo o sistema, embora ferramentas mais sofisticadas possam chegar até 100% do sistema modelado em determinadas circunstâncias, o fato é que trabalhar com MDA, dá mais tempo aos programadores para trabalhar sobre as partes mais interessantes da lógica que o código gerador não conseguiu lidar.

A utilização de ferramentas para geração de scripts de testes a partir dos modelos torna os **responsáveis pelos testes** muito mais produtivos, além de não apresentar nenhuma incompatibilidade com a escrita manual de scripts de testes quando isto se fizer necessário.

A idéia durante os testes é simular todos os caminhos de execução da aplicação, com este propósito, um script de teste precisa combinar pequenos e diferentes conjuntos de parâmetros de entrada em um grande número de combinações, para na sequência poder comparar a saída da aplicação com os resultados esperados.

Com uma ferramenta para geração automática dos scripts de teste, de forma rápida, o testador pode produzir um grande número de testes individuais que são necessários para o teste exaustivo de uma aplicação, isto sem o ônus de tempo e tédio de escrever e reescrever fragmentos de código ligeiramente diferente necessário no processo manual.

Os custos de manutenção representam mais da metade do custo total no desenvolvimento de qualquer aplicação de longa duração, isto normalmente se dá devido o serviço de detetive que os **responsáveis pela manutenção** devem fazer para entender o comportamento da aplicação quando não existe documentação ou ela está desatualizada ou atrasada.

Com o advento da MDA que utiliza um projeto preciso para construir uma aplicação, esta atividade se torna muito mais produtiva, porque quando se faz necessário uma alteração, este procedimento é realizado nos modelos e regras do projeto e não diretamente no código e no caso de modificações no comportamento da aplicação, é o PIM que será modificado.

Se a plataforma de destino for modificada, os mapeamentos são alterados ou se for o caso, totalmente substituídos por outros mapeamentos para uma plataforma

distinta. O resultado disto é uma vida muito mais longa para a aplicação e um trabalho muito menos oneroso para os responsáveis pela manutenção.

2.7 CONCLUSÃO

Este capítulo apresentou diversos aspectos sobre o desenvolvimento de software baseado em modelos, com a intenção de descrever o funcionamento desta abordagem, sendo discutidos os desafios pertinentes ao desenvolvimento de sistemas tendo sido abordado a especificação MDA do OMG, descrevendo-se seus fundamentos tecnológicos e conceitos essências.

A abordagem MDA apresenta uma série de vantagens em sua proposta, e com a intenção de tirar benefício das mesmas, ela foi considerada como pertinente e boa opção para o desenvolvimento de software no ambiente de nuvens computacionais.

3. SOFTWARE COMO SERVIÇO – SAAS

O SaaS é uma das camadas do ambiente de nuvens computacionais que em seu paradigma propõe uma série de vantagens para os provedores e consumidores de serviços. Neste capítulo descreveremos conceitos, características e diversos aspectos relacionados a esta estrutura.

3.1 INTRODUÇÃO

Os SaaS tem se apresentado nos últimos anos como uma tendência para a industria de software e dado suas características tem conseguido diversos adeptos, fato que solidifica sua permanência no mercado e valida novos investimentos e continuidade nas pesquisas intencionando maior amadurecimento e padronização.

É um modelo de negócios caracterizado pelo acesso centralizado das informações através da internet, o que implica que de qualquer lugar e com qualquer dispositivo computacional um usuário pode ter acesso ao serviço.

A proposta essencial é que uma instituição que forneça SaaS, provedora do serviço / software, possua diversos clientes, os inquilinos, que solicitam uma aplicação para um fim específico e que por sua vez, devem ter seus próprios clientes, os usuários finais. Desta forma, o usuário final utiliza uma aplicação que pertence a um cliente do provedor do serviço, sendo este o responsável pela hospedagem, manutenção e suporte de todo o ambiente.

Não é necessário a aquisição de licenças para se utilizar um SaaS, reduzindo-se então custos operacionais, sendo cobrado apenas o serviço que for utilizado de acordo com as tarifas estabelecidas em contratos assinados previamente.

Este modelo é um desafio para os profissionais de Tecnologia da Informação (TI), sendo necessário profissionais competentes, preparados e ágeis, pelo menos tão rápidos quanto os seus concorrentes, e que tenham uma visão clara de mercado, de forma a ser possível expandir a produção e a eficiência do negócio.

3.2 DEFINIÇÃO DE SAAS

Assim como no contexto mais amplo da computação em nuvem, considerando-se apenas SaaS, também ainda não existe um consenso em relação ao seu conceito. Porém, pode-se perceber a existência de uma concordância a respeito de algumas características que devem estar presentes em um SaaS, tais como a implementação em um ambiente centralizado e a necessidade de grande quantidade de usuários.

Em 2008, [SIRTL, 2008] já apontava a orientação a serviços como o núcleo para três fenômenos que ocorreram nas empresas de TI: SaaS, com a proposta de fornecer aplicações compostas de serviços hospedados em nuvens computacionais que reduzem custos e tempo para disponibilização; SOA, com a implementação de serviços com baixo acoplamento e portanto aumentando a flexibilidade e agilidade das empresas de TI; e a Web 2.0 focando no consumo de serviços e na experiência dos usuários.

A definição de Software como Serviço para [DAS , 2010] descreve-o como uma forma de entregar as funcionalidades de software através da Internet a partir de uma única instância de aplicação sendo compartilhado entre todos os usuários.

Uma única cópia de software que pode ser disponibilizada aos consumidores sob demanda como um serviço compartilhado e acessível através de uma rede com localização remota onde é cobrado uma assinatura ou se paga pela quantidade de uso, esta é a definição de [NAMJOSHI , 2009] para o termo SaaS.

[SUN, 2007] declara que o aumento exponencial do poder de processamento, a aprovação e amadurecimento da virtualização, arquiteturas orientadas a serviços, uma maior largura de banda introduziu novo tipo de entrega de software, o SaaS, que é um modelo de fornecimento de software, que permite aos clientes acesso a funcionalidade do negócio, remotamente, normalmente, através da Internet como um serviço.

Conforme [CANDAN, 2009], o alto custo de criação e manutenção de software e infra-estruturas de hardware para oferecer serviços às empresas criou uma tendência para a utilização de serviços de terceiros, sendo provido por estes, redes de comunicação, poder de computacional, espaço para o armazenamento de

dados para clientes e toda infra-estrutura necessária. Estes serviços de terceiros podem melhorar a disponibilidade e escalabilidade de um sistema, diminuindo assim os encargos das empresas com a gestão de infra-estruturas complexas e é denominado software como serviço.

Para [HONG, 2009], SaaS é um modelo de entrega de software que permite que os consumidores alterem e limitem suas responsabilidades. A alteração de responsabilidades se dá considerando a implantação, desempenho e manutenção do software, enquanto a restrição se observa pela necessidade de prover apenas computadores pessoais e conectividade a internet além do pagamento ao provedor do SaaS.

Passar a distribuir SaaS ao invés de fornecer software pelo modelo tradicional, modifica a forma de um provedor pensar em três aspectos, o modelo de negócios, a arquitetura do software e a estrutura operacional, estabelecendo um novo paradigma, [CHONG, 2006].

O modelo de negócios é alterado, pois o provedor passa a hospedar o software e a ter toda a responsabilidade com a infra-estrutura, implementação e gerenciamento. No contexto do cliente se apresentam menores custos com a manutenção, gerenciamento e implementação. Outra vantagem é a garantia de constante desenvolvimento, garantindo à sua rede de usuários finais um modelo tecnológico sempre de vanguarda.

No caso do provedor, a vantagem se apresenta no fato de que o gerenciamento, armazenamento e implementação dos softwares acontece para todos os clientes de uma única vez, logo, quanto maior o número de clientes, menores serão custos com os serviços oferecidos que é um reflexo direto do que [ANDERSON, 2006] descreve com a teoria da cauda longa, que será detalhada posteriormente.

Considerando a arquitetura do aplicativo, o objetivo é que o cliente pague pelo uso do serviço e conclua que a utilização de SaaS também agrega benefícios econômicos, além de uma maior flexibilidade na gestão do negócio, visto que o software poderá ser operado a qualquer tempo e em qualquer lugar. Além destes benefícios, o modelo SaaS permite um desdobramento da utilização dos serviços,

onde o cliente pode disponibilizar o software contratado para uso de terceiros.

A estrutura operacional também é alterada. Segundo [CHONG, 2006], normalmente, o investimento com TI em uma organização é feito em três áreas distintas: software, hardware e serviços profissionais. O gerenciamento das informações está profundamente atrelado ao software, embora os outros aspectos sejam totalmente relevantes no ambiente de TI, geralmente o software retém maior destaque, porque acaba sendo o objeto de maior visualização.

Encerrando este tópico sobre definições sem obviamente chegar a exaustão na descrição de conceitos sobre SaaS disponíveis na literatura, considera-se importante desfazer interpretações equivocadas relacionadas os conceito de SaaS, afirmando que não é o mesmo que Arquitetura Orientada a Serviços (SOA) ou Webservice.

Os webservices são uma solução utilizada para a integração de sistemas e a comunicação entre aplicações distintas, permitindo a interação entre aplicativos novos e legados e que sistemas desenvolvidos em plataformas diferentes sejam compatíveis. São componentes que permitem as aplicações enviar e receber dados em um formato especifico, desta forma cada sistema pode ser desenvolvido em uma plataforma ou linguagem distinta, bastando apenas observar o formato que normalmente é XML.

O princípio da SOA é que as funcionalidades implementadas por uma aplicação devem ser disponibilizadas como serviços, representando uma metodologia de desenvolvimento de software que possibilita que partes do código de uma aplicação, que sejam significativos e com uma funcionalidade bem definida, sejam vinculados a outros componentes, possibilitando o compartilhamento e reutilização de diversas maneiras.

De forma resumida, SOA é uma arquitetura para criação de aplicações, normalmente dentro de uma empresa, que demanda uma metodologia específica de desenvolvimento de software, enquanto que WebServices são mecanismos de comunicação, criados sob a World Wide Web para conectar aplicações e ainda SaaS representa como o software pode ser utilizado como serviço.

Por fim, estas tecnologias podem ser integradas, porém cada uma tem um

propósito específico, sendo que existem soluções de SaaS baseadas em SOA e que utilizam Webservices, sendo possível combiná-las, porém este fato não as faz uma arquitetura única.

3.3 ASPECTOS GERAIS SOBRE SAAS

Sempre existe muito entusiasmo com o surgimento de novos paradigmas e tecnologias, principalmente quando as tendências sugeridas vão se confirmando ao longo dos anos, porém sempre é salutar observar atentamente para entender e avaliar as conseqüências na adoção de um determinado modelo.

A proposta de SaaS incorpora novos mercados, possibilitando o alcance de novos clientes e tornando efetiva a concorrência da empresa no mercado de negócios.

Algumas das premissas que precisam ser analisadas, porque efetivamente podem se tornar barreiras para adoção, quando se pensa em utilizar aplicações SaaS em detrimento de softwares tradicionais são descritas a seguir.

A **confiança** no fornecedor é um quesito fundamental em qualquer solução. Para software licenciado, confiança resume-se basicamente na qualidade dos serviços de suporte e atualização e na longevidade do fornecedor.

Qualquer solução SaaS oferece uma interface mais nítida de terceirização, com responsabilidades definidas e vinculadas ao serviço entregue. Isto permite estabelecer relações de confiança em um novo nível, onde qualquer falha interna é imediatamente imputada diretamente e somente ao provedor.

De certa forma, esta analogia também pode ser utilizada em relação a **segurança**, lógica e física, visto que, teoricamente, quanto mais encapsulado for o produto final, mais seguro será o serviço, porque os componentes, e conseqüentes possíveis falhas, ficam encapsulados na infra-estrutura do fornecedor, sendo este o único responsável.

A **propriedade de dados** é um aspecto muito impactado com esta mudança de paradigma, onde a questão preponderante seria, onde estão armazenados os dados e quem tem acesso a eles?

Os serviços normalmente oferecem garantias de recuperação, cópia e redundância de dados em um nível muito mais abstrato que partições e tabelas. Algo que é possível porque a própria infra-estrutura sob demanda exige grande abstração para permitir eficiência no gerenciamento e escalabilidade.

As **plataformas proprietárias** sempre geram uma preocupação legítima, isto é discutido ao longo dos anos e a discussão repete-se agora sobre plataformas de computação em nuvem.

A baixa interoperabilidade entre plataformas não impediu de forma consistente o avanço das mesmas no passado, porém se deve escolher um provedor com expectativas a longo prazo e com boa capacidade de execução.

Quanto a aplicações, a interoperabilidade em software licenciado é mínima e a vinculação a um fornecedor é uma condição comum. Mais uma vez, a maior abstração de aplicações e componentes para serviços encapsulados favorecem o surgimento de protocolos de interoperabilidade em relação a negócios que são garantidos contratualmente.

Não é difícil imaginar que aplicações rodando na nuvem podem substituir em larga escala aplicativos que precisam de licença para utilização.

Avaliando o aspecto **disponibilidade**, por exemplo, aplicações de produtividade precisam de vários recursos disponíveis para funcionar a partir de um dispositivo, da rede sem fio, da internet e do provedor.

Somado a isto, o acelerado crescimento das plataformas móveis gera a necessidade de disponibilidade constante, em qualquer lugar e a qualquer tempo, logo, como este quesito afeta diretamente os consumidores, obviamente os produtos precisam apresentar esta característica.

3.4 CARACTERÍSTICAS DE UM SAAS

Neste tópico são descritas características, funcionalidades e conceitos que estão relacionados com SaaS e podem ser identificadas em diversas propostas deste modelo.

3.4.1 Níveis de maturidade

A figura 3.1 descreve os quatro níveis de maturidade de SaaS apresentado por [CARRARO, 2006], onde no nível 1, o que estava no cliente foi trazido para dentro da empresa do provedor, com um ambiente separado para cada cliente.

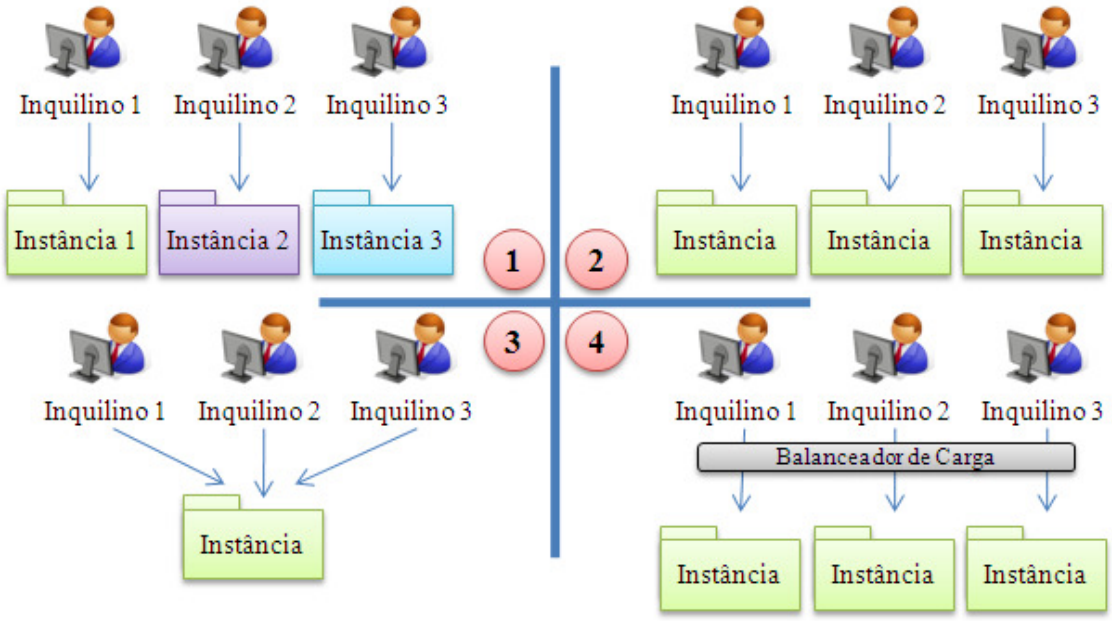

Figura 3.1 – Níveis de Maturidade

No nível 2, utiliza-se o alocamento de recursos, porém cada cliente possui a sua base de dados separada dos demais, e existem atributos de configuração e isolamento físico ou virtual. No nível 3, o ambiente é para múltiplos clientes sendo compartilhado com atributos de configuração. Por fim, no nível 4, se faz necessário o balanceamento de carga, estando presente as características anteriores.

Segundo [CHONG, 2006], os departamentos de TI têm experimentado grandes evoluções nas últimas décadas, sendo que estas tem influenciado de forma direta os modelos de negócio, por isto, o conceito sobre níveis de maturidade relacionado a SaaS vem ajudar a entender essas transformações descrevendo como os recursos tecnológicos podem beneficiar as empresas. A seguir são descritos cada um destes níveis de maturidade.

Nível 1 – Personalizado/ad hoc: cada cliente possui sua própria e específica versão da aplicação, hospedado e sendo executado nos servidores dedicados no

provedor ou em partições separadas.

O atendimento é diferenciado, porém os custos são maiores visto que não existe o compartilhamento da estrutura e a customização única é onerosa. Assemelha-se ao modelo ASP - Application Service Provider.

Nível 2 – Configurável: as instâncias continuam separadas como no nível anterior, a diferença está no fato de que o código fonte é compartilhado e o provedor fornece aos clientes opções de configuração onde a edição de atributos modifica aspectos do software.

Esta configurabilidade não requer serviços de customização, pois neste nível, as alterações no código são realizadas em sua parte comum, sendo preservados os atributos de configuração, podendo a atualização ser distribuída de forma rápida e ao mesmo tempo para todos os clientes, já que todos irão receber exatamente a mesma atualização.

Nível 3 - Configurável e Múltiplos Clientes: o provedor executa uma única instância que serve a todos os clientes. Obviamente o tratamento de metadados, manutenção de configuração e modelagem do banco de dados são essenciais neste nível. Devido a característica de vários clientes existe um compartilhamento total dos recursos.

O custo neste nível é dividido entre os clientes, sendo que mais pessoas podem ser atendidas. Políticas de acesso, autenticação e segurança passam a ser requisitos básicos, devendo o provedor garantir que os dados de um cliente sejam mantidos separados dos demais e que não exista a possibilidade de uma instância da aplicação ser compartilhada entre vários clientes.

Nível 4 - Escalável, Configurável e Multi Clientes: o provedor atende vários clientes com instâncias idênticas, sendo os dados dos clientes mantidos separados através de metadados configuráveis e personalizados para cada cliente.

Devido o aspecto de escalabilidade, para o provedor, o trabalho com o hardware é mais complexo, pois precisa estar preparado para as futuras e repentinas demandas. Na prática, conforme cheguem as demandas dos usuários, as requisições são delegadas a um ou outro servidor, conforme o nível de carga que

exista no momento, de forma a maximizar o tempo de resposta.

Neste nível, a escalabilidade é garantida, porque o número de servidores e instâncias pode ser modificado conforme a demanda, sem a necessidade de modificações na aplicação.

A maturidade de um SaaS se apresenta na forma de uma seqüência contínua de eventos, onde de um lado estão códigos e dados separados e, de outro lado, códigos e dados compartilhados. A continuidade do software depende muito de perto das necessidades dos clientes.

3.4.2 Ciclo de Desenvolvimento

Conforme [MELO, 2007], as características que definem o modelo SaaS estão fundamentadas nos princípios básicos de licenciamento, localização e gerenciamento do ciclo de vida da aplicação.

O valor do licenciamento no modelo SaaS é semelhante quando comparado ao modelo tradicional, porém neste o cliente ainda terá que investir em customização para acompanhar os avanços tecnológicos como a customização, hardware, implementação, entre outros, enquanto que no modelo SaaS o pagamento é pela utilização. A expectativa é que os provedores de SaaS forneçam cada vez mais aplicações com custos de assinaturas menores.

O princípio de localização relaciona-se ao local onde o software está hospedado e o meio pelo qual ele é acessado. Nesse sentido, no Brasil, ainda não se tem uma conexão de banda e velocidade necessárias para um desempenho satisfatório e que se possa desconsiderar a latência da rede, e não se pode ignorar as consequências relacionadas com a lentidão de conexão em uma aplicação.

Em SaaS os softwares são hospedados por terceiros e acessados através da Internet; uma alternativa que contribui para amenizar a questão da distância da rede e largura de banda, é a utilização de “appliance”, que são aplicações pré-configuradas e instaladas no cliente, onde os dados são processados e colocados em cache para diminuir o número de micro-transações e aumentar a velocidade de acesso do usuário [MELO, 2007].

Em relação ao fator gerenciamento do ciclo de vida da aplicação, as

empresas precisam desenvolver mais o setor de gerenciamento da informação para manter e operar as aplicações de SaaS. Além disto, precisam conhecer o ambiente do cliente e prezar por ações operacionais no sentido de resolver problemas de segurança, performance, disponibilidade, entre outros.

O objetivo é que realmente os fornecedores de SaaS cuidem da gerência de TI, sendo que a execução de tarefas de gerência bem como suas responsabilidades intrinsicas não sejam pertinentes aos consumidores de SaaS.

3.4.3 Características Específicas

São descritos a seguir algumas características que identificam um SaaS.

3.4.3.1 Arquitetura Multi-Cliente

Esta arquitetura é o modelo utilizado para o compartilhamento do processamento das aplicações e recursos de armazenamento de dados em um ambiente um-para-muitos.

Para o fornecimento de SaaS, essa arquitetura é considerada muito eficiente, pois possibilita que os provedores de software alcancem a economia de escala ao gerenciar os custos de infra-estrutura servindo a muitos consumidores, de grande ou pequeno porte.

Nesta arquitetura, todas as partes compartilham uma estrutura comum e uma base de código que é mantida centralizada, devido estarem na mesma infra-estrutura os fornecedores podem inovar mais rapidamente, liberando novas versões, e com isto economizar um valioso tempo de desenvolvimento que anteriormente era gasto com a manutenção de inúmeras versões com código desatualizado.

3.4.3.2 Aplicações configuráveis

Como aplicativos SaaS são dirigidos a uma arquitetura multi-cliente, normalmente não aceitam customizações, de forma que, ao contrário do modelo tradicional, não é possível um cliente alterar o código fonte da aplicação, o esquema do banco de dados ou as interfaces gráficas.

Porém, aplicações SaaS são projetadas para suportar configurações, que são

um conjunto de opções, parâmetros, que afetam as funcionalidades e aparência da aplicação. Cada cliente pode ter seus próprios parâmetros para as opções de configuração.

Aplicações SaaS podem permitir que um cliente forneça através de uma interface seu logotipo e ainda um conjunto de cores embora não seja permitido alterar o layout das páginas para algo diferente do que foi projetado, sendo que estas especificidades são preservadas quando ocorrem atualizações.

3.4.3.3 Rápido Desenvolvimento

Considerando que não há preocupações com instalação, configuração e manutenção do ambiente de desenvolvimento, os desenvolvedores podem se preocupar unicamente com os detalhes das regras de negócios, diminuindo o tempo para desenvolvimento do software e aumentando por consequência sua produtividade.

3.4.3.4 Rápida Atualização

Segundo [Creese, 2010], aplicações SaaS são atualizadas mais frequentemente que os softwares tradicionais, muitas vezes semanalmente ou mensalmente.

Isto acontece porque a hospedagem é centralizada, então quaisquer atividades de atualização se tornam visíveis, e de forma transparente, a todos os clientes sem haver necessidade de reinstalação específica no cliente.

Como a configuração é única, os testes do desenvolvimento também são mais rápidos. Além disto, o provedor da solução pode ter acesso ao comportamento do usuário dentro da aplicação, através de uma web analítica, o que torna mais fácil identificar áreas com necessidade de melhorias.

3.4.3.5 Protocolos de Integração Abertos

Visto que aplicações SaaS não podem acessar os sistemas internos de uma empresa, elas precisam oferecer protocolos de integração bem como APIs que possibilitem conexões através da rede.

A ubiquidade de aplicações SaaS e de outros serviços da internet, além da

padronização de APIs, aumentou o desenvolvimento de mashups, que são aplicações que combinam dados, apresentação e funcionalidades de múltiplos serviços criando então um serviço composto, fato este que aumenta a diferença para os softwares licenciados visto que estes não são facilmente integrados fora do firewall da empresa.

3.4.3.6 Funcionalidades Colaborativas

Certamente inspirado no sucesso das redes socias, aplicações SaaS podem prover características que permitam os usuários colaborarem e compartilharem informações.

Estas funcionalidades também podem ser utilizadas para identificar novas idéias para melhorar o software. Uma colaboração, implícita ou explícita, entre usuário de clientes distintos só é possível a partir de um software com hospedagem centralizada.

3.4.3.7 Disponibilidade

Software como serviço permite o acesso e utilização dos softwares disponíveis no mercado através de uma rede remota ou conexão à Internet.

Isso significa que o software não está instalado no computador do assinante, mas sim no servidor do provedor de SaaS. Isso também faz com que o software esteja disponível para o assinante a qualquer tempo e independentemente de sua localização.

O SaaS ainda deve ser acessível por qualquer tipo de dispositivo computacional, garantindo que todos possam ver as informações ao mesmo tempo, sendo um desafio melhorar o acesso aos dados, de forma a tornar mais fácil o gerenciamento de privilégios e o controle da utilização dos dados.

Deve-se levar em consideração a possibilidade de ocorrências como incêndio, roubo, ataques de hackers, pois o provedor deve estar tecnologicamente preparado para fazer com que o sistema volte a funcionar o mais breve possível a fim de atender o contrato firmado com o cliente.

3.4.3.8 Licenças

Os SaaS possuem um modelo de licenciamento baseado em assinatura, ao contrário de uma taxa única para uso perpétuo, como é feito com o softwares tradicionais.

Este modelo utiliza uma abordagem simples, baseada em tempo de uso, sendo cobrado somente aquilo que foi utilizado pelo usuário, como é feito, por exemplo, pelas companhias de energia elétrica.

3.4.3.9 Gerenciamento

As aplicações SaaS são completamente gerenciadas pelo fornecedor, e acordos SLA, regem a qualidade, disponibilidade e suporte que o provedor deve prover para o cliente.

Algumas formas de gerenciamento essências para SaaS incluem o provisionamento, funções de configuração, controle de pagamento, monitoração e suporte.

3.4.3.10 Escalabilidade

A escalabilidade é um atributo desejável em qualquer aplicação, e representa a habilidade de manipular uma quantidade crescente de serviço de forma uniforme, ou o fato de estar preparado para expandir ou retrair o sistema de acordo com o aumento do tráfego, sem modificar sua arquitetura [BONDI, 2001], sendo esta característica crucial para os sistemas a longo prazo.

Quando um sistema não é escalável, o custo adicional para lidar com o tráfego dos dados é muito elevado, além da alta probabilidade de se perder clientes por causa da má qualidade do serviço.

3.4.3.11 Modelo de Dados Extensível

Certamente, clientes distintos possuem diferentes tipos de necessidades, e o atendimento a estas especificidades pode ser feito adequadamente com a utilização de um modelo de dados extensível.

A implementação deste modelo é uma parte muito delicada no

desenvolvimento de um SaaS, visto que tudo o que se torna configurável automaticamente deixa de ser customizado, por isto necessita de metadados para lhe dar suporte, que também precisam ser armazenados em uma base de dados.

Esta situação exige do desenvolvedor a implementação de mecanismos que possibilitem aos clientes estender o modelo original de dados para garantir a satisfação de suas necessidades e isto sem afetar o modelo utilizado pelos demais clientes. Este é um dos grandes desafios de SaaS, fornecer uma arquitetura que atenda necessidades diversificadas dos clientes.

Em suma, a extensibilidade dos dados é um recurso do sistema que vai permitir ao cliente inserir campos customizados na aplicação, que não fazem originalmente parte do sistema e serão válidos apenas para um cliente específico.

3.5 VANTAGENS E LIMITAÇÕES DE UM SAAS

Descreve-se nesta seção aspectos considerados como vantagens e desvantagens da utilização de um solução SaaS.

3.5.1 Vantagens

Um **menor custo inicial**, comparando-se aos softwares tradicionais, pois não existem taxas de licença, sendo cobrado apenas pelo uso do SaaS sendo que o provedor é o responsável por gerenciar toda a infra-estrutura de TI, bem como seus custos e ainda os gerentes humanos.

Obter um software seguro com as especificidades necessárias e com garantia de estabilidade das operações, certamente é muito mais caro do que utilizar os serviços de um software compartilhado com outras empresas que tenham interesses semelhantes.

Desta forma, uma empresa passa a dividir sistematicamente os custos com outras empresas envolvidas ao invés de arcar com os elevados custos unitários. Ou seja, do ponto de vista macroeconômico, esse argumento leva à uma economia de escala. Este é um forte argumento de venda, uma das razões pelas quais o modelo de SaaS vem crescendo no mercado.

É apresentado ao cliente um **tempo reduzido para disponibilização da aplicação**. Considerando que um sistema de porte médio se desenvolve normalmente em seis meses e que eventualmente pode demorar décadas, neste modelo de distribuição, ao contrário de customizar o software, o cliente apenas vai configurar o aplicativo com os atributos disponibilizados pelo desenvolvedor.

Considerando a infra-estrutura tecnológica, no modelo tradicional, a comum extrapolação no tempo de implementação normalmente garante que, quando o projeto atingir seu final, poderá estar obsoleto tecnologicamente, porque facilmente um dos componentes utilizados pode ter sido modificado por outra tecnologia mais recente. Existe ainda o caso onde, por questões financeiras, os administradores optam por uma solução tecnológica menos atualizada. Diferentemente, no modelo SaaS, o cliente normalmente fará uso das versões mais recentes e tecnologias mais atualizadas, mesmo porque os custos relacionados são divididos entre todos os clientes.

As **atualizações são transparentes** e de única responsabilidade do provedor, não existem arquivos para download ou para serem instalados. Além disto, o provedor gerencia a disponibilidade e o cliente não tem que se preocupar em adicionar hardware ou largura de banda com o crescimento de usuários.

Segundo [Amorim, 2008], Os usuários esperam que os serviços estejam disponíveis e sejam melhorados com o tempo sem a necessidade de versões, instalações ou upgrades para o usuário se preocupar. Estes aplicativos são chamados pelos seus próprios fornecedores como beta e esta tendência é comumente denominada de beta perpétuo, que quer dizer que, a todo o momento o software está evoluindo, novas funcionalidades são inseridas e os bugs são corrigidos. Tudo ocorrendo de maneira totalmente transparente para o usuário.

É realizado um **melhor gerenciamento da aplicação**, em termos de suporte, isto devido a centralização do ambiente, porque é mais fácil administrar um conjunto de recursos, todos em um mesmo ambiente, normalmente um Data Center, do que espalhados em vários locais.

Os aplicativos SaaS possuem **alto nível de disponibilidade**, pois estão disponíveis a partir de quaisquer computadores ou dispositivos computacionais, a

qualquer hora e em todos os lugares, por isto SaaS tende a ter altos índices de adoção, com pouco esforço para aprendizagem, dado que já existe uma familiarização geral com o uso da internet.

Acontece uma **evolução constante do provedor**, pois podem escalar sua infra-estrutura de forma indefinida para atender a demanda de um cliente, além de poder oferecer funcionalidades para atender necessidades específicas, agregando integração a sistemas ERP, *Enterprise Resource Planning*, legados ou sistemas de produtividade, isto sem interromper o fornecimento do SaaS.

3.5.2 Limitações

Existem também desvantagens ou pelo menos fatores que podem dissuadir uma adoção generalizada de uma solução de SaaS.

Todas as operações possuem o pressuposto de serem realizadas através da internet. Infelizmente, não se pode garantir um serviço com um grau de confiabilidade próximo dos cem por cento. Algumas aplicações não se podem dar ao luxo de ficarem indisponíveis porque aconteceu algum problema com a internet do provedor e normalmente quando isto acontece se dá em momentos críticos.

A velocidade das operações é outro fator determinante. Hoje em dia estamos cada vez mais próximos de velocidades que facilitam a realização de operações mais pesadas de forma suave, mas ainda assim trata-se de uma comunicação via internet, onde a largura de banda varia grandemente entre as localidades.

Existem questões óbvias associadas as realidades empresariais, tais como, se o fornecedor do serviço desaparecer inesperadamente? Será extremamente complicado resolver uma situação destas, principalmente se a utilização do serviço for mais para intensiva do que esporádica e o provedor simplesmente comunicar que no prazo de 30 dias fechará as portas.

A integração de diversos SaaS e/ou ferramentas e serviços de terceiros pode ser controversa, pois pode muito bem ser um fator diferenciador entre o serviço A e o serviço B, mas também pode ser a fonte de problemas. Um serviço que integra com a sua oferta mais 2 ou 3 ferramentas não essenciais mas ainda assim de utilidade complementar, consegue ganhar vantagem competitiva, porém ganha igualmente a

instabilidade somada de todas as partes que o compõem. É um jogo delicado e perigoso, tanto para fornecedor quanto para cliente.

A segurança é um item fundamental que deve ser bem trabalhado em qualquer software comercial. Porém, em uma aplicação desenvolvida no modelo SaaS, existem alguns problemas que não se encontram no modelo tradicional, como a segurança dos dados.

O provedor precisará dar garantias que os dados não serão perdidos ou violados, portanto aspectos relacionados a políticas de backup e segurança, em suas diversas possibilidades, devem ser bem elaboradas afim de garantir as cláusulas do SLA. Apesar do software ser o mesmo para todos os clientes, os dados de cada cliente são únicos e particulares.

O quesito confiança também é de suma importância, visto que os clientes colocam nas mãos dos fornecedores toda uma credibilidade, sem saber se a qualidade dos serviços é compatível. Podem ser utilizadas auditorias realizadas por auditores independentes, terceirizados, para constatar se os serviços prestados atendem aos requisitos especificados em contrato.

Outra questão refere-se à escalabilidade, isto é, a capacidade de manter o desempenho de todos os elementos especificados como requisitos da aplicação, incluindo a segurança, na medida em que o volume de dados evolui e que o número de usuários aumenta.

No modelo SaaS, é muito maior a tendência que problemas aconteçam em menos tempo, porque dependendo do software, pode-se passar de cinco mil para cinqüenta mil usuários em tempo relativamente pequeno.

3.6 ASPECTOS FINANCEIROS DO SAAS

Não se pode descartar a avaliação financeira quando se pensa em qualquer tipo de investimento, logo este contexto também é muito importante e deve ser avaliado de forma consciente, considerando-se exatamente quanto se gasta e o que se obtêm de retorno com a adoção de um determinado modelo de software.

A fonte de renda de empresas de software tradicional inclui os direitos

autorais sobre o software, as atualizações do software, manutenção e outras taxas de serviços. Porém o ideal de transformar o software em serviço, cria uma separação da propriedade do software e o direito de uso [TURNER, 2003], e este fato possui grande influência sobre o modelo de negócio de software tradicional.

Em [CUSUMANO, 2004] já se observava que o declínio das vendas de licenças tradicionais em relação aos serviços. Segundo [CUSUMANO, 2008], a venda de produtos poderia até continuar crescendo, mas a taxa de crescimento da venda de serviços é muito maior.

3.6.1 A Teoria da Cauda Longa

A economia de mercado é alcançada quando uma empresa que fornece produtos consegue vender a menor quantidade de tipos de produto para o maior número possível de consumidores. Em um processo industrial, quanto maior for a variedade de cores e tipos de matéria-prima utilizada, maior será o custo associado ao produto, que nem sempre pode ser diretamente repassado ao consumidor.

Com a evolução da tecnologia aplicada aos negócios, surgiu uma nova orientação relacionada a postura em relação ao mercado que propõe uma dinâmica onde o conceito de massa transforma-se em atendimento personalizado, que carece de novas ferramentas.

Segundo [ANDERSON, 2006], a era de consumidores em rede, onde todo o processo é digital, imprime uma forte mudança na economia de distribuição, visto a absorção de quase tudo pela internet, onde a mesma é transformada em loja, teatro e mecanismo de difusão, obtêm valores de custos por uma fração mínima do custo tradicional.

A internet, conforme [ANDERSON, 2006], é a responsável por um novo universo, no qual novas tecnologias causam impacto nos negócios atuais, onde a tendência é que o modelo de mercado onde os consumidores eram direcionados por listas de sucesso pré-fabricadas, seja desestruturado e gradualmente perca espaço devido a introdução de novas tecnologias que possibilitam os usuários a trafegar entre os segmentos pretendidos identificando o que realmente satisfaz as suas necessidades, sem considerar os modelos impostos.

Percebe-se que os provedores de soluções de SaaS defrontam-se com uma curva de negócio semelhante ao da cauda longa. Na figura 3.2 a parte amarela representa o lado tradicional, onde se trabalha vendendo muito a poucos clientes. A medida que o custo do produto pode ser reduzido, seu preço final também é, logo, mais clientes irão poder comprar, representando os clientes típicos pela cor cinza, então consequentemente, com uma redução de custos ainda maior, começa-se um conquistar gradual de clientes, referenciado pela cor verde.

O que a teoria da cauda longa revela é que não existe limite à direita da figura, logo, ao contrário de se ter algumas poucas empresas que vendem milhões de unidades de softwares, lucrando com economia de escala, é possível, no modelo SaaS, uma empresa gerenciar, muitos softwares para os quais só se tinha poucos clientes interessados, obtendo lucro ao mesmo tempo em que diminui os custos para os usuários.

Com esta nova arquitetura de negócios, substitui-se a realidade de algumas empresas que vendem milhões de unidades de software por milhares de empresas que gerenciam alguns softwares como serviço, porém, na soma acumulada dos produtos por segmentos, o volume gerado pode superar o volume dos produtos de massa e consequentemente o faturamento e o lucro serem maiores.

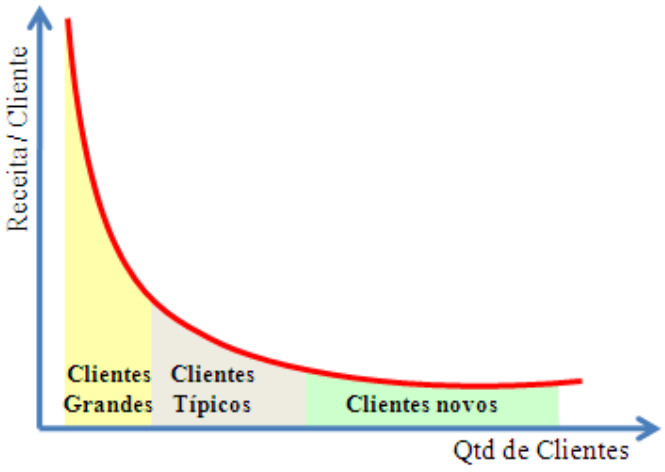

Figura 3.2 – Teoria da Cauda Longa - alterado

Com o oferecimento de softwares com menor custo, existe uma tendência de adoção por uma quantidade maior de clientes, sendo que este número pode ser

ilimitado, dado que a curva não toca o eixo X.

Desta forma, os provedores de SaaS podem oferecer recursos por um valor global menor em relação aos fornecedores tradicionais, não apenas pelo custo mas também pelo fato de os clientes não precisarem investir como antes em infra-estrutura relacionada a TI. Então percebe-se que os provedores de SaaS passam a ter acesso a uma faixa inteiramente nova de clientes que antes não conseguiam alcançar por ser economicamente inviável.

Dois elementos são apontados por [ANDERSON, 2006] como mais importantes para o sucesso no mercado de cauda longa: tornar o produto disponível e criar mecanismos de busca eficientes para que os consumidores consigam chegar até estes produtos.

3.6.2 Estrutura de Custos

A utilização da estrutura de SaaS apenas para distribuir uma aplicação na rede não é o melhor motivo para a adoção deste modelo, deve haver uma estratégia comercial para garantir que haverá receita.

Obviamente é necessário haver público que demande pelo serviço, portanto o cliente precisa fazer um estudo dos gastos envolvidos e estimativas de uso das funcionalidades que serão ofertadas.

Um SaaS propicia para as empresas a oportunidade que elas não corram os riscos da compra de software e transfiram o departamento de TI de um centro de custo estático para um custo dinâmico, como uma parte da empresa que produz valor [CARRARO, 2006].

Quando se compra a licença de um software, no custo total de propriedade, Total Cost of Ownership - TCO, o valor da licença representa a menor parte, isto porque para implementar e customizar o software é necessário a obtenção de hardwares com desempenho compatível com o software que será executado.

Além disso, existem os custos com pessoal, relacionados com a manutenção e operação do software bem como o treinamento dos usuários, que na verdade, representam um risco significativo para a organização, fazendo com que este software fique fora do alcance de algumas empresas que poderiam obter dele

grandes benefícios.

Na figura 3.3 é apresentado um dos argumentos de vendas utilizados por uma empresa de software, a Salesforce, onde percebe-se uma comparação relacionando os custos com softwares.

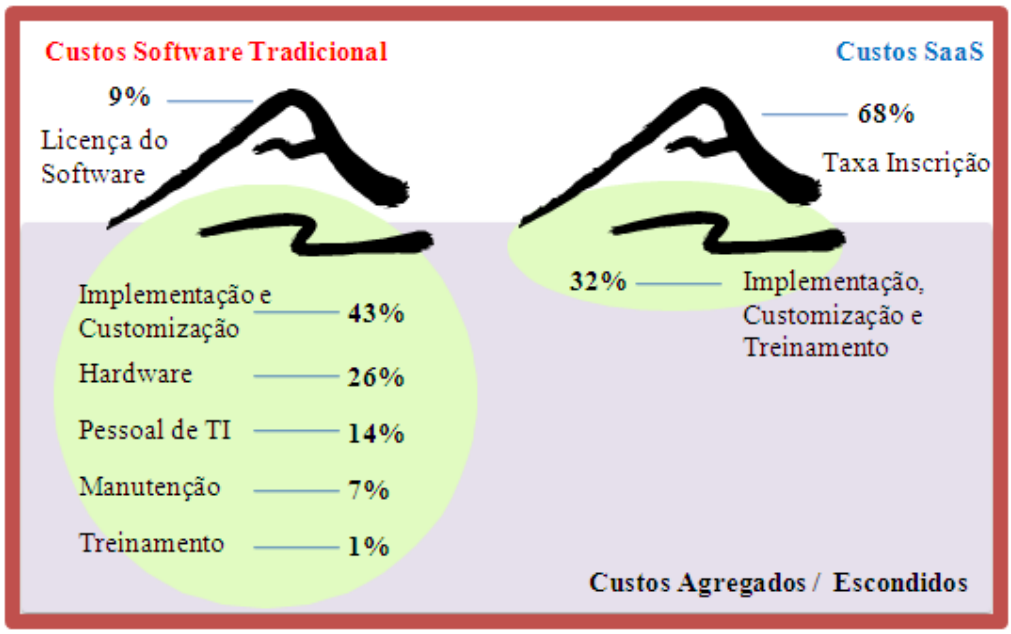

Figura 3.3 – Estrutura de Custos

As áreas verdes na figura 3.3, que estão relacionadas à criação e manutenção de software, desaparecem no modelo de SaaS, isto porque grande parte desses fatores são de exclusiva responsabilidade do provedor.

Os custos implícitos são menores e, ao mesmo tempo, o valor que se paga pela licença é semelhante. Para quem está operando neste modelo, lado direito da figura, perceberá que, a longo prazo o modelo é mais viável do que o modelo tradicional.

3.7 CONCLUSÃO

De forma semelhante a conceituação de nuvens computacionais, ainda não existe um consenso na literatura sobre a definição de SaaS.

Neste capítulo foi desenvolvida uma pesquisa bibliográfica abrangente sobre

este tema, com o propósito de oferecer um referencial teórico que inclui diversas áreas ligadas ao tema, relacionando funcionalidades e aspectos importantes de um SaaS.

O objetivo foi descrever características especificadas na literatura que são esperadas em um SaaS, seu ciclo de vida, as vantagens e limitações de sua utilização, para com isto possibilitar o entendimento de porque esta forma de entrega de software tem alcançado cada vez mais adeptos ao longo dos anos.

Como a literatura aponta SaaS como tendência para os próximos anos, esta forma de entrega de aplicações, bem como a solução de seus desafios, se mostra relevante e com grande possibilidade de aceitação no meio científico e de negócios.

4. SLA

A computação em nuvem, de forma um tanto quanto generalizada, utiliza o conceito de acordos em nível de serviço (Service Level Agreements – SLA) para controlar o uso de recursos computacionais advindos de um provedor.

4.1 INTRODUÇÃO

As propostas de SLA apresentam muitas diferenças, mas, segundo [SCHNJAKIN, 2010] é possível identificar uma estrutura geral para um SLA, que apresenta informações sobre as partes, parâmetros do SLA, métricas para calcular os parâmetros do SLA, algoritmo para calcular os parâmetros do SLA, objetivo de nível de serviço - SLO e ações a serem tomadas em caso de violação do acordo.

Normalmente as estratégias de gerenciamento SLA estão baseadas no entendimento de duas fases distintas, onde uma se relaciona com a negociação do contrato e a outra está focada na monitoração em tempo real de sua execução.

Desta forma, pode-se dizer que um gerenciamento SLA deve incluir a definição do contrato, a partir de um esquema básico com parâmetros de QoS, Quality of Service – Qualidade de Serviço, a negociação do SLA, o acompanhamento do SLA e ainda a execução do SLA, isto tudo de acordo com uma política definida entre as partes.

Neste contexto, o ponto principal é construir uma nova camada acima da nuvem computacional que seja capaz de criar um mecanismo de negociação entre provedores e consumidores de serviços.

O objetivo básico de uma nuvem computacional é compartilhar recursos, fato este suportado devido a natureza intrínseca de um ambiente com infra-estrutura compartilhada. Desta forma, acordos de nível de serviço, são oferecidos pelos provedores de serviço ao longo da nuvem.

Segundo [ENTRIALGO, 2011], questões de qualidade de serviço podem ser abordadas sob vários pontos de vista, tais como prestar um serviço sujeito a restrições de desempenho ou como descobrir e selecionar dinamicamente um

serviço com requisitos de desempenho, portanto medições, monitoramento e relatórios de desempenho da nuvem são baseados na capacidade dos usuários consumirem os recursos disponíveis. Sendo que é importante que os provedores ofereçam SLAs baseados em desempenho para os usuários [SCHAD, 2010].

Um grande desafio para o paradigma de nuvens computacionais, relacionado com SLA, é determinar a causa da interrupção de um serviço, dado a natureza complexa do ambiente.

A negociação automática de atributos de QoS ainda não está devidamente equacionada no contexto de empresarial. Vários esquemas de SLA e protocolos de negociação já foram propostos, dentre eles, o WS-Agreement, [ANDRIEUX, 2007], e o WSLA [KELLER, 2003]. Estes mecanismos, de modo geral, tratam apenas a negociação de um serviço simples com a básica troca de mensagens. A negociação automática de múltiplos serviços e em várias etapas ainda carece de amadurecimento.

A execução e o acompanhamento SLA também possuem uma função de supervisor, com a intenção de verificar se as condições definidas no contrato, para todos os serviços em execução estão sendo cumpridas. Podem ser utilizados mecanismos de notificação para avisar que uma situação anormal esta acontecendo e ainda disparar ações corretivas de acordo com as políticas definidas.

4.2 REQUISITOS SLA

A seguir, podem-se identificar alguns requisitos importantes para o cenário de SLA, bem como possíveis soluções.

Na figura 4.1, pode-se observar um diagrama com estes requisitos e suas relações, sendo que são especificadas três fases distintas, a primeira, Fase de Definição do Contrato SLA, pode ser dividida na especificação do modelo, sua publicação e descobrimento. A Fase de Negociação do SLA envolve duas etapas, a negociação propriamente dita e a otimização do SLA. Por fim, a Fase de Execução do SLA é representada por quatro atividades, o acompanhamento, a avaliação, a re-negociação e a contabilidade do SLA.

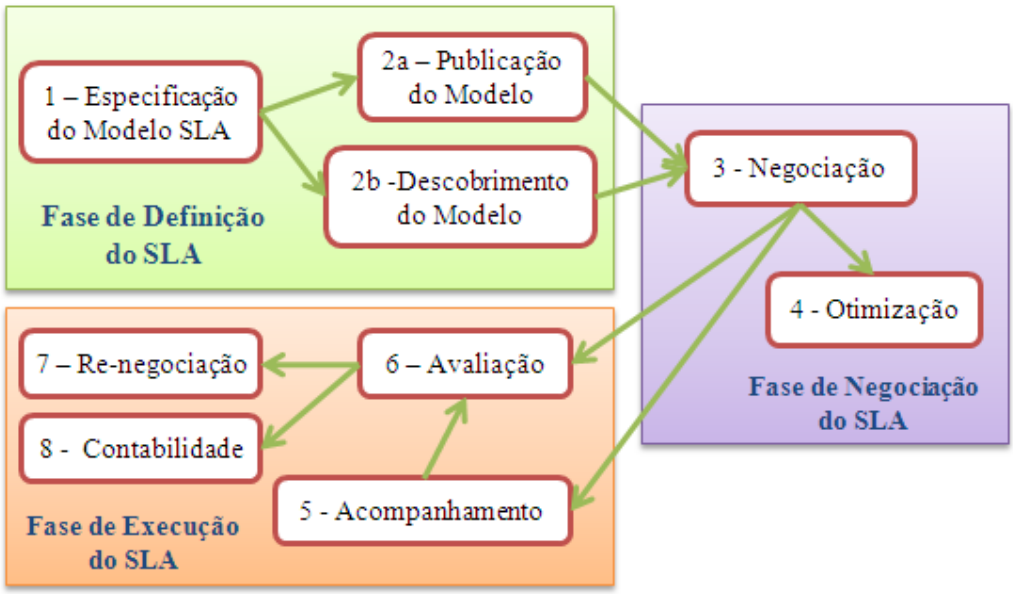

Figura 4.1 – Requisitos SLA

As subseções a seguir, descrevem cada etapa das três fases dos requisitos de um SLA.

4.2.1 Especificação do Modelo SLA

Quando um novo recurso é disponibilizado por um provedor de serviço, ele deve incluir também um modelo de SLA que descreva um tipo de contrato compatível com o uso dos recursos para regular sua utilização por todos os envolvidos no ambiente de negócio, incluindo penalidades pelo não cumprimento das regras.

Pode ser difícil definir um modelo, porém este desafio pode começar a ruir com a utilização, a partir de adaptações realizadas em modelos utilizados anteriormente.

4.2.2 Publicação e Descobrimento do Modelo SLA

Uma vez que um recurso foi disponibilizado, o ideal é que a maior quantidade possível de usuários tenham conhecimento deste fato. Este anúncio deve apresentar a capacidade dos recursos oferecidos e deve ser encaminhado para um grande público.

Um ambiente de publicação / descobrimento deve ser provido de um agente

atuando como ponte entre os usuários potenciais e o provedor da oferta, afim de permitir que possíveis parceiros estabeleçam comunicação.

4.2.3 Negociação

O cliente e o provedor de serviço, que são as partes envolvidas no processo, devem aceitar os termos do SLA que os vincula.

Eles também devem detalhar as responsabilidades de cada parte bem como as conseqüências advindas da violação das normas. Esta negociação pode consumir grande quantidade de tempo representando um longo processo.

A disponibilização de uma interface onde seja possível debater pontos conflitantes do SLA pode gerar um rápido acordo, beneficiando mais rapidamente a todos.

4.2.4 Otimização

Quando um provedor aceita uma requisição, ele precisa colocá-la em uma fila e utilizar uma política de agendamento para definir a ordem que vai atender as solicitações.

Apesar disto, o provedor também precisa considerar como otimizar a utilização dos recursos e como preservar os parâmetros de QoS que são garantidos pelo SLA.

Neste cenário, é muito importante considerar a possibilidade da chegada de novas requisições e suas respectivas prioridades, mesmo já estando executando outras tarefas, para atendê-las com os recursos que satisfaçam os requisitos da forma mais adequada.

4.2.5 Acompanhamento

Considerando o fato que um provedor iniciou a provisão de acesso aos recursos, ele deve monitorar a operacionalização destes recursos.

As informações monitoradas podem ser utilizadas para verificar se os atributos de QoS definidos no SLA estão sendo respeitados.

4.2.6 Avaliação

Os gerentes e usuários, não possuem um interesse focado exclusivamente em saber apenas se uma tarefa está sendo executada corretamente. Outras informações como violação de contrato ou estatísticas globais também são relevantes para a verificação do SLA.

Uma avaliação consiste em um processo que analisa informações pré-determinadas que foram monitoradas e registradas em momentos anteriores. A identificação de informações relevantes é importante para retratar com fidelidade o processo.

4.2.7 Renegociação

A qualquer momento, uma das partes do contrato pode querer alterar as políticas de uso dos recursos, normalmente para estar em conformidade com alguma exigência externa advinda de mudanças no contexto.

Uma vez que a premissa de alterações sempre vai existir enquanto o sistema estiver executando, é importante considerar este aspecto, mas apesar do ambiente sofrer alterações o comportamento dos processos deve permanecer inalterado. Ou seja, é preciso garantir que a após qualquer migração, adição ou remoção de recursos o sistema continuará funcionando corretamente.

4.2.8 Contabilidade

A utilização dos recursos obviamente deve gerar uma lista descrevendo quais foram utilizados, em que medida e por quanto tempo, bem como relacionando os valores acertados pelo uso de cada um deles de acordo com as definições estabelecidas no SLA.

Isto não representa o faturamento propriamente dito, mas é a base para elaboração do prospecto financeiro, que inclusive pode ser desfavorável ao provedor em caso do não cumprimento dos requisitos de QoS.

4.3 FUNCIONALIDADES DE UM SLA

Uma funcionalidade comum em um SLA deve descrever uma capacidade de atender

a um ou vários requisitos.

4.3.1 Otimização da Seleção de Recursos

Este aspecto aborda a necessidade de um algoritmo de otimização integrado ao alocador de recursos, com a intenção de encontrar o tempo de alocação mais adequado, considerando as definições do SLA.

Esta funcionalidade deve levar a uma otimização da execução de uma instância de uma aplicação, tarefa ou serviço baseado nos parâmetros SLA de forma a maximizar a probabilidade de satisfação do SLA.

As requisições, submetidas a um SLA, são processadas para selecionar o melhor host, entre todos os disponíveis. A definição de melhor para um host depende do estado de uma série de variáveis no sistema, tais como, os recursos disponíveis, os recursos que são necessários para satisfazer os requisitos do SLA e ainda os objetivos de otimização.

Alguns objetivos estão diretamente relacionados ao conhecimento dos requisitos em um SLA, como a minimização do tempo de conclusão, a minimização de custos, maximização da probabilidade de sucesso, etc) enquanto outros objetivos estão relacionados com o estado do sistema, por exemplo, o balanceamento da carga de trabalho.

4.3.2 Acompanhamento em Tempo Real

Esta funcionalidade consiste em um processo de monitoração da execução de uma instância de um serviço relacionando-a com as definições de um SLA de forma a concluir se o contrato está sendo respeitado ou não.

Durante a execução, se algum parâmetro associado aos SLOs, Service Level Objectives ou Objetivos de Nível de Serviço, do SLA, que são efetivamente os tópicos a serem medidos dentro do SLA, atingir um valor limite, identifica-se uma ameaça de violação e ações de recuperação podem ser ativadas a fim de preservar o SLA ou até mesmo minimizar as conseqüências de uma violação efetiva do contrato.

Entre as ações de recuperação, pode-se visualizar a re-alocação das tarefas

de uma aplicação em recursos disponíveis ou ainda a aquisição de recursos adicionais.

Uma opção interessante seria a de enviar alertas de ameaça para o provedor para lhe capacitar a tomar medidas para tentar impedir a violação. Outra opção para esta funcionalidade seria a possibilidade de oferecer informações para o usuário que assinou o SLA, informando-o sobre o status atual do SLA, tornando o processo mais transparente, isto porque qualquer ferramentas de monitoramento estará junto com os recursos que estão em posse do provedor.

Uma alternativa para dirimir qualquer suspeita seria a utilização de um módulo de terceiros para monitorar o processo, porém isto pode ir contra as práticas de negócio do provedor.

4.3.3 Negociação do Contrato

O cliente e o provedor precisam concordar com as condições do SLA, antes de assiná-lo, e este ponto de comum acordo está baseado nas solicitações do usuário e no posicionamento do provedor.

Esta funcionalidade diz respeito ao procedimento das partes, consumidores e provedores, concordando com os termos de um SLA que regem o intercâmbio de recursos. As partes tentam chegar a um acordo baseado em um consenso depois de compartilhar suas idéias e objetivos.

Um aspecto comum em acordos comerciais é que o provedor normalmente não publica ofertas, ele espera o consumidor enviar uma proposta. Para tanto o cliente precisa conhecer o que o provedor está pronto a oferecer, o que normalmente acontece através de propagandas genéricas feitas pelo provedor.

Desta forma, um provedor publica um modelo que descreve o serviço e os seus termos, incluindo parâmetros de QoS e compensações possíveis em caso de violação, seria uma proposta básica de serviços. Estes modelos dispõem de campos em branco ou editáveis, sendo destinadas a relacionar as necessidades específicas do usuário.

Em posse do modelo, o cliente preenche-o com valores que descrevem o seu planejamento para o uso dos recursos disponíveis. Novos termos podem ser

adicionados, outros podem ser removidos ou alterados, gerando um novo documento que deve ser encaminhado ao provedor.

Nesta ocasião, o provedor, considerando sua disponibilidade de recursos e políticas internas, envia ao cliente uma cotação com valores considerados, por ele, pertinentes para a provisão dos serviços conforme as especificidades relacionadas no documento enviado pelo cliente.

O cliente se estiver satisfeito com a cotação, assina o documento e envia para o provedor como uma proposta. Esta proposta representa um acordo que o usuário está pronto para cumprir. O provedor é livre para rejeitar ou aceitar, no último caso, a proposta torna-se um SLA oficialmente assinado por ambas as partes, e começa a ser um documento legal válido.

Caso o cliente não fique satisfeito com a cotação, o processo pode ser repetido, até chegar-se a um acordo ou então desconsiderar a contratação do serviço com aquele provedor.

Durante a negociação as partes podem modificar livremente os termos em relação a taxas, parâmetros de QoS, disponibilidade de recurso, etc.

Uma vez que o contrato foi concordado e assinado, ainda assim a necessidade de mudança precisa ser considerada, sendo necessária uma renegociação. Neste caso, o mesmo procedimento pode ser utilizado embora tenha-se que considerar a existência do primeiro contrato, que já existem recursos alocados e ainda que dependendo da alteração podem surgir dificuldades técnicas no lado do provedor.

4.3.4 Publicação e Descoberta de Serviços

Um espaço virtual comum ou um mercado de serviços, é necessário para que provedores e clientes possam se comunicar.

Os primeiros devem anunciar os seus serviços, através de um mecanismo de publicação baseado em seus modelos de SLA, enquanto os últimos precisam encontrar quem possa satisfazer as suas necessidades, possivelmente utilizando um mecanismo de busca.

Disponibilizar serviços para terceiros, obviamente envolve a desafiante tarefa de comunicar a sua existência, fornecendo meios para encontrá-los. Portanto, técnicas para a publicação de serviços, ou seja, anúncios para uma comunidade, precisam ser estabelecidas.

Descobrir serviços representa um processo de localização de provedores com o objetivo de obter-se documentos com a descrição dos serviço que tenham sido disponibilizados.

Para implementar uma funcionalidade relacionada com a publicação e descoberta de serviços, minimamente são necessários alguns componentes infra-estruturais, tais como um mecanismo de registro no espaço virtual de serviços, onde um provedor de serviços publica o seu serviço juntamente com os preços de cada serviço.

A criação e registro de corretores de recursos, que são responsáveis por descobrir os recursos e serviços com atributos específicos que atendem os consumidores com seus requisitos de QoS, também se faz necessário.

E ainda os consumidores de serviços também precisam de um mecanismo de registro neste espaço virtual, para o caso dele mesmo querer pesquisar em busca de serviços que atendam suas necessidades ou atribuir essa tarefa a um corretor de recursos que fiscaliza o espaço.

Essas tarefas envolvem vários aspectos, todos alicerçados na existência do mercado de serviços, que pode ser reunir provedores e consumidores em potencial. Este mercado precisa estabelecer procedimentos para a publicidade dos serviços, como a definição de formatos comuns para modelos de SLA e descrição dos serviços, mas também deve prover ferramentas de mineração de dados para facilitar a pesquisa.

4.3.5 Criação de Modelos SLA

Os provedores de serviço podem ter dificuldade para descrever seus recursos de forma clara e segura, incluindo aspectos jurídicos e de usabilidade.

Esta funcionalidade está relacionada a criação e definição de modelos SLA que serão publicados por um provedor. Estes modelos são componentes muito

importantes na negociação do SLA e para a descoberta de serviços do provedor, tanto que os modelos são as ferramentas que dão suporte a publicidade dos serviços para o mundo exterior e a negociação sempre começa com um cliente solicitando um modelo de SLA para um provedor.

De modo geral, um modelo SLA deve ser facilmente compreendido por si só, porém sem perder sua expressividade, sendo que isto permitirá a qualquer cliente entender exatamente o que está especificado no modelo além de facilitar a criação dos modelos pelo provedor.

Embora possa ser tentador adaptar os modelos de SLA existentes para atender às necessidades do cliente, é importante que sejam consideradas as necessidades do provedor de serviços como as mais importantes, em caso de conflito, as necessidades do prestador de serviços deve ter preferência sobre as necessidades do cliente de serviços.

A criação de modelos SLA não é uma tarefa trivial e provedores de serviço sem experiência precisam de assistência para especificar os modelos SLA para os serviços que prestam, por isso não é suficiente fornecer somente a definição do que o modelo deve conter.

Parece razoável fornecer componentes para aliviar o fardo de criação de um modelo SLA, sendo que isto pode ser realizado fornecendo-se uma interface simples para a criação de um framework para modelos SLA que podem então ser refinados pelo provedor.

O uso de modelos SLA deve sempre estar em sintonia com os aspectos legais, o que é crítico para ser utilizado no domínio do negócio.

4.3.6 Contabilidade do SLA

Os provedores precisam fazer um resumo dos recursos usados em uma base de dados do usuário para compará-lo com as condições acordadas no SLA.

Os dados monitorados devem ser utilizados e analisados, sendo a contabilidade feita regularmente, incluindo as possíveis sanções para as violações do SLA.

Esta funcionalidade abrange o processo de cobrança dos consumidores pela utilização dos serviços do provedor de acordo com os preços acordados entre as duas partes que estão definidos no SLA.

Para que a cobrança dos clientes seja feita através do SLA, o modelo de SLA deve ser concebido com este propósito. De forma mais especifica, algumas métricas do SLA devem representar os termos dos preços, descrevendo como o consumidor vai ser cobrado para o uso de um recurso específico.

As métricas que representam condições de preço devem conter informações como o preço pelo uso da métrica, a moeda e o período pelo qual o preço é válido. O projeto do modelo de SLA deve ser suficientemente flexível para permitir diferentes tipos de métricas e encargos. Custos adicionais poderão ser cobrados dado o uso cumulativo de uma métrica ou para o aumento da disponibilidade de uma métrica ao longo do período de faturamento.

O componente de contabilidade deve recolher periodicamente informações sobre o uso, que é fornecido pelo componente de acompanhamento, para então calcular as taxas que devem ser cobradas do cliente.

A qualquer momento o cliente deve ser capaz de visualizar o uso de recursos que ele fez, juntamente com seus respectivos encargos.

Também é possível definir sanções que serão impostas ao consumidor, quando um de seus aplicativos causarem uma violação de SLA. Nesse caso, a taxa de penalização relacionada com a violação de uma restrição também deve ser incluída no modelo SLA como um dos seus termos. Sendo que o componente de acompanhamento deve enviar notificações para o módulo de contabilidade cada vez que uma violação ocorre, de modo que este possa calcular a taxa, com base no SLA, para então notificar o consumidor.

4.4 TÓPICOS DE UM SLA

Um SLAs tem que descrever as condições comuns do negócio, características como parâmetros de desempenho e disponibilidade, métricas de avaliação, contabilidade e questões jurídicas, além dos prazos de contrato [MALKOWSKI, 2010].

A forma exata de um SLA depende de uma série de fatores, incluindo se o SLA é um contrato separado e único, ou faz parte, de um contrato maior. Normalmente, as negociações são facilitadas, se anexos para novos serviços puderem ser adicionados ao SLA sem ter que haver uma re-negociação do corpo principal.

A figura 4.2 apresenta um protótipo de um modelo SLA, sendo que um resumo dos principais tópicos que devem ser incluídos em um SLA é relatado em [Open Group, 2004] e são apresentados nas subseções a seguir, sendo que estes tópicos em sua maioria estão inseridos no quadro Termos da figura 4.2, na forma de especificidades dos serviços ou na descrição de garantias.

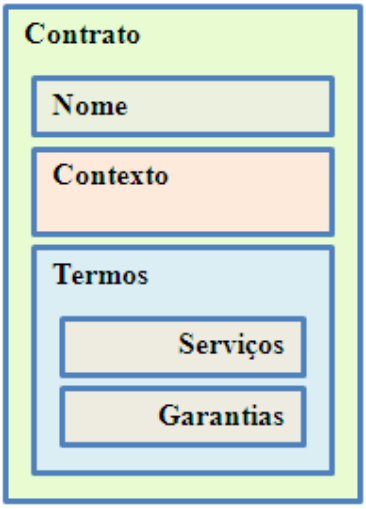

Figura 4.2 – Modelo SLA

4.4.1 Introdução

Esta seção deve documentar as partes interessadas, o provedor e o consumidor, que estão de acordo com SLA. A introdução deve conter também uma breve panorâmica sobre a necessidade do SLA e a aplicação ou serviços a que ela se destina.

4.4.2 Requisitos do Cliente

A documentação de como o cliente quer usar um serviço que esteja disponível deve constar nesta seção, por exemplo, se um tempo de resposta menor que um segundo é um exigência para uma determinada operação.

Também se faz necessário a compreensão dos valores de pico e a duração de quaisquer surtos de transações que sejam esperados, podendo ser necessário determinar como o serviço deve responder quando, no caso exemplificado, o tempo da transação é ultrapassado.

4.4.3 Visão Geral do Serviço

Esta seção deverá descrever o serviço, incluindo a localização física e lógica das interfaces entre as duas partes, quem é o proprietário de cada parte da interface, o número de localização e quaisquer outras informações que descrevam o produto ou serviço de forma adequada.

4.4.4 Prazo

O período durante o qual o SLA é válido deverá estar especificado nesta seção, de forma ideal informando uma data de início e fim da vigência.

4.4.5 Responsabilidades

As responsabilidades devem estar especificadas nesta seção, tanto do cliente, para que o provedor garanta a conformidade, quanto a responsabilidade do provedor para com o cliente. As expectativas de ambos os lados também podem ser detalhados nesta seção.

4.4.6 Detalhes do Serviço

Esta seção deve descrever a parametrização do serviço em termos de Key Performance Indicator (KPI) e como eles deverão ser comunicados ao cliente. Isso provavelmente irá assumir a forma de uma tabela que deve mostrar claramente os níveis de desempenho aceitáveis e de condições fora da especificação.

4.4.7 Exceções

É provável que exceções precisem ser incluídas e claramente documentadas no SLA. Se for o caso, esta seção deve atender a esta necessidade.

4.4.8 Amostragem e Relatórios

Será necessário definir com que freqüência os KPIs do SLA são medidos como

parâmetros de conformidade e quantas vezes eles são recolhidos na forma de um relatório ou para calcular a Key Quality Indicator (KQI) da aplicação .

Relatórios de não-conformidade, dos requisitos fora de especificação podem exigir uma freqüência diferente, instantaneamente, dentro de uma hora, etc, ignorando desta forma o processo de verificação normal. O método de relatar não-conformidades também pode ser diferente e deve ser documentado.

A notificação de eventos assíncronos como alarmes, alertas, podem também ser diferentes e devem ser respondidos. Pode ainda ser necessário estabelecer qual a frequência destes eventos assíncronos podem ser toleradas pelo cliente ou fornecedor.

4.4.9 Sanções

As sanções pelo não atendimento dos requisitos devem ser detalhadas e elas podem variar dependendo do tipo de notificação, podendo ser:

- Perda de taxas
- Reembolso de honorários
- A compensação por lucros cessantes
- Rescisão
- A combinação dos anteriores

Invocar as cláusulas penais não significa necessariamente no ganho de grandes benefícios, considerando-se que o dano já está feito e qualquer sanção compulsória não é suficiente para compensar, mesmo que parcialmente, a perda de um negócio ou danos causado à imagem da marca.

4.4.10 Resolução de Disputas

Esta seção deve documentar como as diferenças de opinião sobre o SLA, seus relatórios ou questões de desempenho são resolvidos. Pode ser necessário fornecer os dados de contato para o caso de situações que não puderem ser resolvidas facilmente.

4.4.11 Solicitações de Mudança

Deve constar nesta seção o detalhamento do processo de como solicitações de mudança no SLA serão feitas e manipuladas, com o detalhamento de quaisquer despesas.

A freqüência de solicitações de mudanças deve ser detalhada, estabelecendo-se os termos para não haver excessos que venham onerar o custo total de tais pedidos.

Avisos para os períodos de solicitações de mudanças devem ser documentados. O desempenho destas solicitações de mudança pode estar sujeitos a penalizações.

4.4.12 Rescisão do SLA

É possível que ambas as partes, em algum momento, queiram rescindir o SLA por um motivo especial. Estas razões precisam ser documentadas, bem como o período de aviso prévio para a rescisão e quaisquer custos associados devem ser detalhados.

O período de aviso pode variar se for do provedor para o cliente em relação a quando for do cliente paro o fornecedor.

Outro fator que pode ser considerado aqui é o fato de que uma das partes pode ser adquirida por uma terceira parte, de tal forma que as exigências do serviço podem ser muito diferentes. Será preciso determinado se o SLA será encerrado, se deve continuar como está, ou se será renegociado.

4.4.13 Legislação

Devem ser especificadas nesta seção quais as leis que devem ser consideradas para o SLA, bem como em qual jurisdição, qualquer violação do contrato, deve ser julgada.

4.4.14 Confidencialidade

Esta seção deverá especificar e destacar todos os aspectos do SLA, se existirem, tais como desempenho, relatórios e dados que sejam confidenciais.

4.4.15 Garantias

As áreas que são abrangidas por quaisquer condições de garantia devem ser detalhadas. Sempre que existirem garantias para os recursos de qualquer serviço e como estas garantias afetam o SLA, devem estar descritas neste tópico.

4.4.16 Indenizações e Limitações de Responsabilidade

É importante compreender quem é o responsável em resultado do fracasso do SLA, se é o provedor ou o cliente.

4.4.17 Assinaturas

O SLA deve ser datado e assinado pelos representantes responsáveis de ambas as organizações.

4.5 ARQUITETURA SLA

A figura 4.3 ilustra uma arquitetura proposta por [ZHANG, 2010] para um sistema SLA. As funções de gerenciamento estão divididas em dois módulos principais de acordo com as fases antes da execução e em tempo de execução que a gerência de SLA deve considerar.

Figura 4.3 – Arquitetura SLA

São descritos o módulo de gestão dos registros e pesquisas do SLA e o módulo de gerenciamento de tempo de execução na figura 4.3.

O primeiro módulo está focado no registro e na busca de serviços que não envolvem nenhum parâmetro de tempo de execução, ou seja, ele não é responsável por garantir que a função de acompanhamento defina corretamente se o serviço disponível está em conformidade com o que foi previamente acordado.

O segundo módulo tem o objetivo de monitorar e controlar os estados dos serviço em tempo de execução.

4.5.1 Módulo de Registro e Busca do SLA

Segue uma breve descrição dos componentes deste módulo.

- Registro SLA: usado pelos provedores de serviços para registrar seus parâmetros SLA que serão utilizados no sistema de gestão SLA.
- Receptor de Requisições SLA: responsável por receber as indagações sobre os requisitos de QoS dos consumidores dos serviços.
- Analisador SLA: mapeia com precisão os requisitos de QoS para os parâmetros de serviços SLA.
- Ferramentas de implementação SLA: serve para consultar os serviços que cumpram a exigência de QoS no sistema de SLA.
- Biblioteca de dados SLA: para armazenar todas as informações relacionadas aos serviços.
- Contrato SLA da biblioteca: o contrato SLA da base de dados;
- Negociador SLA: é a entidade utilizada para negociação de contratos SLA entre clientes e prestadores de serviços.

4.5.2 Módulo da Gerência dos Serviços em Tempo de Execução

Este módulo é responsável por capturar parâmetros em tempo de execução e mapeá-los em seus parâmetros SLA correspondentes para identificar se existe alguma violação ocorrendo no contrato.

A seguir, uma breve descrição dos componentes deste módulo.

- Monitor SLA em tempo de execução: responsável por obter, em tempo de execução, parâmetros dos serviços, tais como, tempo de solicitação e tempo de resposta do serviço, se a chamada ao serviço foi bem-sucedida e se os serviços foram executados corretamente.
- Log SLA: o banco de dados de registro;
- Estratégia SLA: corresponde aos requisitos para tratamento do SLA, como por exemplo, quanto os provedores de serviços irão pagar se os serviços atrasarem mais que 10 segundos do que foi acertado em contrato.
- Controlador SLA: compara parâmetros SLA computados por meio de parâmetros de serviço em tempo de execução com métricas de SLA da base de dados, para constatar se existem problemas, e assim, tomar as decisões cabíveis a violação.

4.6 CONCLUSÃO

Neste capítulo foram discutidos conceitos sobre acordos em nível de serviço, ou como é comumente conhecido, SLA. Foram apresentados os requisitos para um SLA, especificados algumas funcionalidades e uma arquitetura e por fim descritos tópicos usuais que são comumente encontrados em um SLA.

A utilização de SLA tem como propósitos principais auxiliar no gerenciamento dos recursos disponíveis na nuvem computacional e garantir a satisfação dos requisitos contratados por cada cliente.

5. TRABALHOS RELACIONADOS

5.1 INTRODUÇÃO

São descritos a seguir alguns trabalhos disponíveis na literatura que estão relacionados com o desenvolvimento orientado a modelos e software como serviço.

5.2 DESCRIÇÃO DOS TRABALHOS

O desenvolvimento de software específicos de tecnologia não é viável a longo prazo, visto que com o surgimento acelerado de novas tecnologias, rapidamente estes softwares podem se tornar obsoletos precisando ser substituídos, esta é a consideração de [RITU, 2011].

Os autores ilustram uma abordagem para o desenvolvimento do PIM e PSM que é independente de tecnologias específicas, usando MDA, visando que os avanços tecnológicos não causem impactos nas aplicações, ou que os mesmos sejam mínimos. Os autores entendem que esta abordagem vai aumentar a longevidade e capacidade de reutilização dos serviços desenvolvidos em uma nuvem.

Uma abordagem dirigida a modelos objetivando a introdução rápida de serviços é apresentada por [COCHINWALA, 2005]. Esta proposta considera que a necessidade manual de escrever e integrar códigos personalizados de sistemas legados pode ser substituída por uma interface de mapeamento entre os componentes de serviços e os componentes de gerenciamento do sistema em operação.

Os serviços são representados como modelos dependentes, a partir dos quais mapeamentos para interfaces de gerenciamento de componentes individuais podem ser derivados. Através do uso de XML e tecnologias relacionadas, a abordagem é aplicável a uma ampla variedade de serviços e sistemas.

Uma nova plataforma de aplicativos SaaS baseada na abordagem dirigida a

modelos é apresentada por [XIAOYAN, 2010]. Esta plataforma fornece uma solução conveniente para realizar a personalização e integração de plataformas.

Neste artigo é apresentada uma visão geral do framework, uma série de mecanismos para trabalhar com modelos, templates de modelos e arquivos com a descrição dos modelos. Em seguida, é descrito como todos estes artefatos trabalham juntos em uma arquitetura multi-clientes, e aspectos sobre a personalização e integração de serviços na plataforma SaaS, para então demonstrar como a plataforma trabalha.

Arquitetura geral da plataforma é descrita em detalhes e explicado como a plataforma foi concebida a partir de três aspectos: conhecimento multi-cliente, personalização de serviços e integração de serviços.

Até o momento da publicação, a plataforma SaaS estava parcialmente concluída, porém a escalabilidade e segurança da plataforma não estavam sendo considerados.

Considerando que existe uma falta de metodologias de desenvolvimento precisas, aplicadas a orientação a serviços, [SANZ, 2008], a partir da abordagem de desenvolvimento dirigido a modelos, e mais especificamente em MDA, propõe a definição de um modelo de Arquitetura Orientada a Serviços Software, SOSA, dentro de um framework metodológico dirigido a modelos denominado MIDAS, para o desenvolvimento adaptável e flexível de sistemas modernos.

Foram discutidos os conceitos e elementos que precisam ser definidos para se alcançar a independência de plataforma e um nível de abstração neutro. SOSA é amplamente utilizado como uma maneira de integrar aplicações empresariais e então a existência deste provedores é justificada pela necessidade de projetar as organizações empresariais para o modelo de arquitetura.

Um framework para linha de produção, orientado a serviço e baseado em modelos é apresentado em [XIAO, 2009]. Os autores explicam a arquitetura, os métodos de integração de ferramentas bem como a interação entre elas. O método dirigido a modelo do processo de produção também é descrito.

Foi proposto um algoritmo heurístico de escalonamento que visa a produção

em escala, sendo descrito como criar e executar uma linha de produção e seus princípios teóricos.

5.3 CONCLUSÃO

Pode-se observar que existe certa carência no contexto do desenvolvimento de software automático para nuvens computacionais, sendo provavelmente este aspecto fortalecido pela complexidade de um ambiente em nuvem e com isto corroborando a relevância deste trabalho.

6. APLICAÇÕES E PLATAFORMAS DE TRABALHO AUTOMÁTICAS EM NUVENS COMPUTACIONAIS

6.1 INTRODUÇÃO

Considerando o grande crescimento do paradigma de nuvens computacionais, bem como a necessidade de desenvolvimento de software de forma rápida e preparado para as alterações inevitáveis de tecnologias e plataformas, propõe-se um mecanismo para desenvolvimento de software que utiliza uma abordagem baseada em modelos, a MDA, para o provimento de SaaS ou de plataformas de trabalho, que podem ser definidos como um sistema operacional com aplicações instaladas e disponíveis para utilização.

Neste contexto são propostos dois cenários, o primeiro denominado de Cenário Aplicação, será desenvolvido primeiro e tem o propósito de automatizar o processo de disponibilização de aplicações no ambiente de nuvens.Com a intenção de prover uma maior capacidade de serviços, em paralelo ao Cenário Aplicação será implementado o Cenário Plataforma, que por sua vez tem o propósito de disponibilizar ambientes de trabalho para os clientes, trazendo então maior funcionalidade ao ambiente.

Ambos os cenários estarão disponíveis em um portal web, dado a atual facilidade de acesso a internet, intencionando-se com isto uma maior visibilidade do produto e como conseqüência, uma maior utilização.

6.2 CENÁRIO APLICAÇÃO

Neste primeiro cenário, que pode ser identificado pela figura 6.1, propõe-se a disponibilização de modelos, bem como a descrição dos mesmos e de seus requisitos.

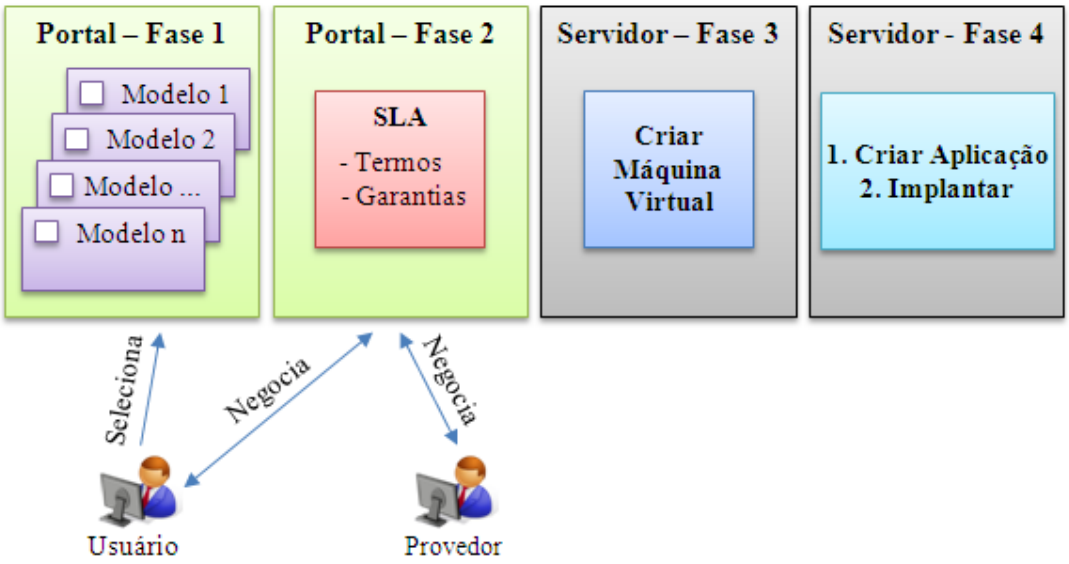

Figura 6.1 – Cenário Aplicação

O processo de utilização deste cenário é composto por quatro fases, sendo que o cliente interage nas Fases 1 e 2 e processos automáticos são responsáveis pelas Fases 3 e 4.

Inicialmente os modelos conterão funcionalidades básicas, possivelmente restringindo-se as operações básicas, conhecidas comumente como CRUD (Create, Retrieve, Update e Delete), estando implícito aos modelos as definições sobre o sistema operacional e banco de dados que serão utilizados, embora pretenda-se que com o amadurecimento da estrutura seja possível fornecer maior flexibilidade ao ambiente.

Os modelos serão criados utilizando a abordagem MDA para que seja possível, sem intervenção humana, a geração automática do código e por conseqüência a disponibilização da aplicação para o cliente.

O funcionamento se dará da seguinte forma:

- **Fase 1**

  O cliente acessa o portal e consulta os modelos disponíveis, observando suas características e funcionalidades para então selecionar, um ou mais que satisfaçam suas necessidades.

- **Fase 2**

Após a seleção dos modelos ser realizada, a mesma é submetida para então ser disponibilizado os termos e garantias de utilização dos produtos, bem como limites (inferior, médio e superior) com a especificação para os atributos e requisitos dos modelos e ainda suas tarifações.

Se os limites disponibilizados forem suficientes para atender as características necessárias ao usuário, o processo continua de forma automática, porém, dado a exigência do usuário, os atributos dos modelos especificados podem ser negociados, através de interface própria, onde maior ou menor quantidade de um parâmetro ou mesmo questão de valores, podem ser tratadas entre o cliente e o provedor, sendo esta etapa sujeita a intervenção humana para aceite ou nas das condições.

Esta negociação, com envio de propostas de ambas as partes pode ser realizada indefinidamente até encontrar-se um ponto comum e o acordo seja efetivado.

- **Fase 3**

Com a efetivação do acordo, é iniciado um processo automático, para que seja criado uma máquina virtual, com os requisitos em conformidade com os modelos escolhidos de forma a apresentar um desempenho adequado e de acordo com as especificações do SLA.

- **Fase 4**

Estando a máquina virtual disponível, inicia-se um processo para gerar a aplicação a partir dos modelos, para por fim, implantar nesta máquina virtual e então a mesma ser disponibilizada ao cliente.

5.3 CENÁRIO PLATAFORMA

A figura 6.2 descreve este segundo cenário. Pode-se perceber que uma grande diferença se apresenta na Fase 1. Isto ocorre porque o propósito deste cenário é disponibilizar para os usuários um ambiente de trabalho.

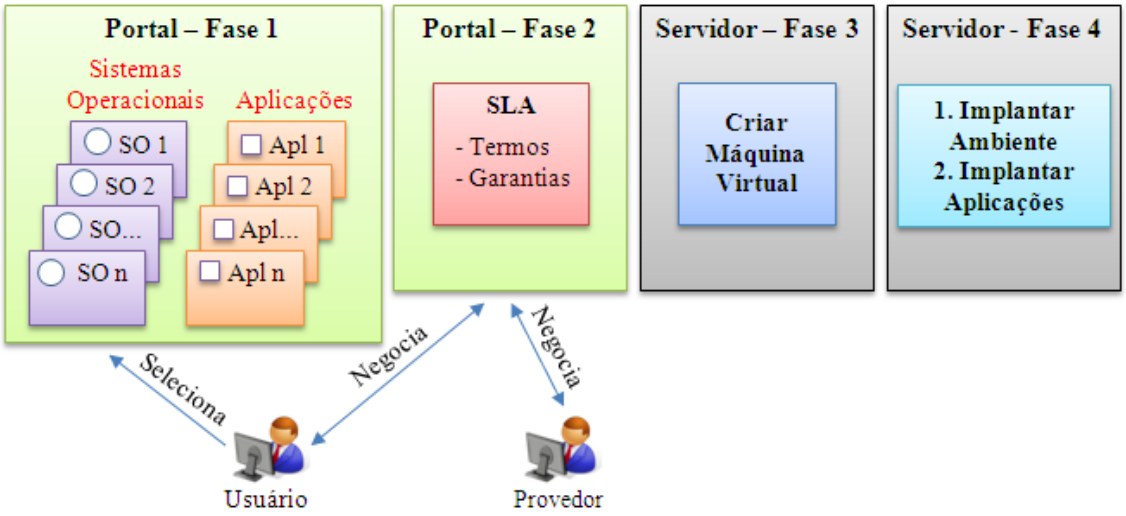

Figura 6.2 – Cenário Plataforma

O funcionamento se dará da seguinte forma:

- **Fase 1**

No portal estarão disponíveis uma lista de sistemas operacionais e aplicações que podem ser instaladas na plataforma em uma nuvem computacional.

O cliente definirá qual o sistema operacional melhor atende suas necessidades e ainda quais aplicações são necessárias e devem estar instaladas na plataforma. Sistemas operacionais e aplicações não contempladas inicialmente poderão ser incluídos posteriormente, de acordo com a demanda, viabilizando maior reutilização do cenário.

- **Fase 2**

Após a definição da plataforma, a escolha é submetida ao portal para então ser apresentado um SLA especificando termos e garantias, viabilizando a negociação, que será feita de forma análoga a Fase 2 do Cenário Aplicação, sendo apenas negociado outro produto.

- **Fase 3**

Através de um processo automático, uma máquina virtual é criada em conformidade com requisitos suficientes para executar o sistema operacional e as aplicações selecionadas sendo também consideradas as especificidades descritas

no SLA.

- **Fase 4**

Nesta etapa, que será realizada sem intervenção humana, são implantados na máquina virtual o sistema operacional e as aplicações que foram definidas pelo cliente.

6.4 CONCLUSÃO

Neste capítulo foi descrito uma proposta para o desenvolvimento automático de software a partir de modelos e ainda a disponibilização, também sem ou com mínima intervenção humana, de ambientes computacionais, sendo ambos no espoco de nuvens computacionais.

7. CONCLUSÃO

Este trabalho apresentou um estudo sobre desenvolvimento de software baseado em modelos, sendo discutidos os desafios pertinentes a construção de sistemas, bem como a adoção de SLA.

Este estudo abordou a especificação MDA do OMG, descrevendo seus fundamentos tecnológicos e conceitos essências, e ainda conceitos sobre software como serviços e contratos em nível de serviço.

7.1 CONCLUSÃO E TRABALHOS FUTUROS

O desenvolvimento de software baseado em modelos tem provado sua eficiência ao longo dos anos, sendo a Modelagem Dirigida a Modelos, MDA, uma iniciativa do Object Management Group, uma proposta para este segmento que se diferencia das demais pela separação entre modelagem e plataforma e ainda geração automática de código a partir do modelo do sistema.

Esta proposta está em fase de amadurecimento onde cada detalhe está sendo estudado de forma a conceber-se a melhor maneira para sua implementação.

A descrição conceitual precisa ser eficientemente embasada, de forma a identificar-se ferramentas e ambientes que sejam mais propícios ao desenvolvimentos das características pretendidas.

Considerando o contexto de nuvens computacionais, pode-se descrever do ponto de vista do usuário, as seguintes contribuições da ferramenta de automação do desenvolvimento de software:

- O usuário não precisa ter conhecimento sobre como o ambiente da nuvem funciona, nem aspectos relacionados a sua manutenção, infra-estrutura ou profissionais especializados.
- A aplicação ou a plataforma de trabalho são especificadas com uma simples seleção de parâmetros, sem a necessidade de instalações, configurações e hardwares robustos no ambiente local do usuário.

- O tempo para disponibilização da aplicação ou ambiente de trabalho é drasticamente reduzido, visto que os modelos para cada funcionalidade já estarão disponíveis e previamente testados, sendo o processo praticamente inteiro realizado de forma automática, exceção feita quando existe a necessidade de negociação de aspectos específicos entre as partes.
- O usuário pode fazer uma previsão clara e objetiva relacionada aos custos com a aplicação ou plataforma, de forma a avaliar com precisão a relação custo / benefício da contratação dos serviços.

Do ponto de vista dos administradores ou provedores de serviços, as seguintes contribuições da ferramenta para automação podem ser destacadas:

- Como o processo apresenta apenas em uma de suas etapas a necessidade de intervenção humana, sendo que esta existe para agregar flexibilidade ao ambiente e pode comumente não ser necessária, os custos relacionados a manutenção e despesas com mão de obra são reduzidos.
- A ferramenta poderá ser utilizada por diversos usuários, o que implica que quanto maior for este número, menor serão os custos relacionados a manutenção da infra-estrutura, que por ser compartilhada por todos os usuários também tem os seus custos divididos entre todos.
- Com a possibilidade de diminuição dos custos, os serviços podem ser oferecidos por um preço menor, com a intenção de alcançar novos clientes, o que traz como conseqüência direta maior lucratividade.
- Em relação ao gerenciamento, as atualizações, novos modelos, novos sistemas operacionais e aplicações são disponibilizados de forma transparente para todos os usuários ao mesmo tempo.
- A estrutura centralizada possibilita atividades de manutenção de forma facilitada, gerenciando-se toda infra-estrutura de um único local.
- Com o advento de muitos usuários utilizando os recursos disponíveis, problemas podem ser localizados e corrigidos rapidamente, bem como

solicitações relevantes de um usuário, podem ser disponibilizadas a todos de forma transparente.

A estrutura centralizada também possibilita a inserção de mecanismos de acompanhamento da usabilidade do ambiente com a intenção de melhorar seu funcionamento, desempenho e identificar novas funcionalidades que podem ser oferecidas, agregando ainda maior relevância ao ambiente.

Como trabalho futuro, porém imediato, pretende-se identificar a ferramenta MDA que será utilizada para criação dos modelos, através de uma avaliação criteriosa comparando-se ferramentas disponíveis no mercado. Em seguida, ambiciona-se criar os modelos e associar seus atributos com um SLA que seja gerenciado automaticamente.

Por fim, com a definição deste primeiro cenário, o trabalho se voltará para o desenvolvimento do segundo, contando com o arcabouço já desenvolvido até então para construir um ambiente de desenvolvimento automático mais completo e flexível.

BIBLIOGRAFIA

[Carr, 2008] CARR, N. Big **Switch: Rewiring the World, from Edison to Google**. Norton & Company, 2008.

[NAUR, 1969] NAUR, P., RANDELL, B. **Software Engineering: Report on a Conference sponsored by the NATO Science Committee**. Scientific Affairs Division, NATO, 1969.

[MILLER, 2001] MILLER, J., MUKERJI., J. **Model Driven Architecture**. Architecture Board ORMSC, 2001.

[OMG, 2003] OMG. **MDA Guide Version 1.0.1**. 2003. Disponível em: < http:// www .omg .org / docs/omg/03-06-01.pdf>. Acesso em: 12 fev 2011.

[OMG, 2010] OMG. **Object Management Architecture**. Disponível em: < http://www.omg.org/oma/>. Acesso em: 15 fev 2011.

[BONVIN, 2011] BONVIN, N., PAPAIOANNOU G., ABERER, K. **Autonomic SLA-driven Provisioning for Cloud Applications**. In 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). 2011.

[CHI, 2011] CHI, Y., MOON, J., HACIGUMUS, H., TATEMURA, J. **Sla-tree: a framework for efficiently supporting sla-based decisions in cloud computing**. In Proceedings of the 14th International Conference on Extending Database Technology, EDBT/ICDT '11, pages 129–140, New York, NY, USA. ACM. 2011.

[La, 2009] La, H., Choi, S., Kim, S. **Technical Challenges and Solution Space for Developing SaaS and Mash-Up Cloud Services**. e-Business Engineering, 2009. ICEBE '09. IEEE International Conference on , vol., no., pp.359-364, 21-23 Oct. 2009.

[Liu, 2010] Liu, F., Guo, W., Zhi, Q., Chou, W. **SaaS Integration for Software Cloud**. Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on , vol., no., pp.402-409, 5-10 July 2010.

[Lin, 2008] Lin, G.; Dasmalchi, G.; Zhu, J. **Cloud Computing and IT as a Service: Opportunities and Challenges**. Web Services, 2008. ICWS '08. IEEE International

Conference on , vol., no., pp.5-5, 23-26 Sept. 2008.

[Dawoud, 2010] Dawoud, W., Takouna, I., Meinel, C. **Infrastructure as a service security: Challenges and solutions**. Informatics and Systems (INFOS), 2010 The 7th International Conference on , vol., no., pp.1-8, 28-30 March 2010.

[Namjoshi, 2009] Namjoshi, J., Gupte, A. **Service Oriented Architecture for Cloud Based Travel Reservation Software as a Service**. Cloud Computing, 2009. CLOUD '09. IEEE International Conference on , vol., no., pp.147-150, 21-25 Sept. 2009.

[Godse, 2009] Godse, M., Mulik, S. **An Approach for Selecting Software-as-a-Service (SaaS) Product**. Cloud Computing, 2009. CLOUD '09. IEEE International Conference on , vol., no., pp.155-158, 21-25 Sept. 2009.

[Pervez, 2010] Pervez, Z., Khattak, A., Sungyoung L., Young-Koo L. **Dual Validation Framework for Multi-Tenant SaaS Architecture**. Future Information Technology (FutureTech), 2010 5th International Conference on , vol., no., pp.1-5, 21-23 May 2010.

[Wu, 2008] Wu C. **Web Services: Software-as-a-Service (SaaS), Communication, and Beyond**. Congress on Services Part II, 2008. SERVICES-2. IEEE , vol., no., pp.1-1, 23-26 Sept. 2008.

[Das, 2010] Das, C., Mohan, G., Roy, R., Bhattacharya, S . **Quo vadis, SaaS a system dynamics model based enquiry into the SaaS industry**. Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference. pp.732-737, 16-18 April 2010.

[OMG, 2008] OMG. **Common Object Request Broker**. Disponível em: < http://www.omg.org/ spec/CORBA/ >. Acesso em: 12 fev 2011.

[OMG, 2011] OMG. **Unified Modeling Language**. Disponível em: <http://www.uml.org/>. Acesso em: 12 fev 2011.

[FAVRE, 2004] FAVRE, J. **Towards a Basic Theory to Model Driven Engineering**. Workshop in Software Model Engineering (WISME 2004), 2004.

[CALIARI, 2007] CALIARI, G. **Transformações e mapeamentos da MDA e sua**

implementação em três ferramentas. USP. 2007.

[MELLOR, 2004] MELLOR, J. **MDA Distilled – Principles of Model Driven Architecture**. Addison-Wesley. 2004.

[KLEPPE, 2003] KLEPPE, A. WARMER, J. BAST, W. **MDA Explained – The Model Driven Architecture: Practice an Promise**. Addison -Wesley. 2003.

[OMG, 2001] OMG. **Meta Object Facility (MOF) Core Specification – Version 2.0**. 2006. Disponível em: <http://www.omg.org/docs/formal/06-01-01.pdf>. Acesso em: 14 fev 2011.

[FITO, 2010] FITO, O., PRESA, G., GUITART, J. **Sla-driven elastic cloud hosting provider**. Parallel, Distributed, and Network-Based Processing, Euromicro Conference on, 0:111–118. 2010.

[OMG, 1997] OMG. **OMG's MetaObject Facility**. Disponível em: <http://www.omg.org/mof>. Acesso em: 16 fev 2011.

[OMG, 2006a] OMG. **XML Metadata Interchange**. Disponível em: <http://www.omg.org/ spec/ XMI/>. Acesso em: 12 fev 2011.

[Entrialgo et al., 2011] Entrialgo, J., GARCIA, F., GARCIA, J., GARCIA, M., VALLEDOR, P., OBAIDAT, S. **Dynamic adaptation of response-time models for qos management in autonomic systems**. J. Syst. Softw., 84:810–820. 2011.

[OMG, 2006b] OMG. **Common Warehouse Metamodel.** Disponível em: <http://www.omg.org/ spec/CWM/>. Acesso em: 13 fev 2011.

[PRESSMAN, 2001] PRESSMAN, R.S. **Engenharia de Software**. 5.ed. McGraw-Hill, 2001. 843p.

[SELIC, 2003] SELIC, B. **The pragmatics of model-driven development**. IEEE Software, v.20, n.5, p. 19-25. 2003.

[FOWLER, 2004] FOWLER, M. **UML Distilled: A Brief Guide to the Standard Object Modeling Language**. 3a. edição. Addison-Wesley, 2004.

[LIMA, 2007] LIMA, B., SOUSA J., LOPES, D., **Using MDA to Support Hypermedia Document Sharing**. IEEE International Conference on Software Engineering Advances (ICSEA 2007), French Riviera, France, 2007.

[SIRTL, 2008] SIRTL, H. **Software plus Services: New IT- and Business Opportunities by Uniting SaaS, SOA and Web 2.0**. Enterprise Distributed Object Computing Conference, 2008.

[TURNER, 2003] TURNER, M., BUDGEN, D., BRERETON, P**. Turning Software into a Servic**e, Computer, Volume: 36, Issue: 10, Oct. 2003,P38-44.

[CUSUMANO, 2004] Cusumano, M. **The Business of Software**, Free Press/Simon & Schuster, 2004.

[CUSUMANO, 2008] Cusumano, M. **The Changing Software Business: Moving from Products to Services**. Published by the IEEE Computer Society. January 2008

[DAS , 2010] DAS, C.; MOHAN, G.; ROY, R.; BHATTACHARYA, S. **Quo vadis, SaaS a system dynamics model based enquiry into the SaaS industry. Information Management and Engineering**. (ICIME), 2010 The 2nd IEEE International Conference on , vol., no., pp.732-737, 16-18 April 2010.

[NAMJOSHI , 2009] NAMJOSHI, J.; GUPTE, A. **Service Oriented Architecture for Cloud Based Travel Reservation Software as a Service**. Cloud Computing, 2009. CLOUD '09. IEEE International Conference on , vol., no., pp.147-150, 21-25 Sept. 2009.

[SUN, 2007] SUN, W., ZHANG , K., CHEN, S., ZHANG, X., LIANG , H. 2007. **Software as a Service: An Integration Perspective**. In Proceedings of the 5th international Conference on Service-Oriented Computing (Vienna ,Austria, September 17 - 20, 2007).

[CANDAN, 2009] CANDAN, K., WEN-SYAN L; PHAN, T.; MINQI Z. **Frontiers in Information and Software as Services**, Data Engineering, 2009. ICDE '09. IEEE 25th International Conference on , vol., no., pp.1761-1768, March 29 2009-April 2 2009.

[HONG, 2009] HONG, C.; KE, Z.; MING, J.; WEI, G.; JUN, C.; XIN, M. **An End-to-End Methodology and Toolkit for Fine Granularity SaaS-ization**. Cloud Computing, 2009. CLOUD '09. IEEE International Conference on , vol., no., pp.101-108, 21-25 Sept. 2009.

[CHONG, 2006] CHONG,F.;CARRARO,G. **Architecture Strategies for Catching**

the Long Tail. Disponível em: <http://www.cistratech.com/whitepapers/MS_longtailsaas.pdf>. Acesso em: 05 jun 2011.

[SCHNJAKIN, 2010] SCHNJAKIN, M., ALNEMR, R., MEINEL, C. **Contract-based cloud architecture**. In Proceedings of the second international workshop on Cloud data management, CloudDB '10, pages 33–40, New York, NY, USA. ACM.

[ANDERSON, 2006] ANDERSON, C. **The long tail: why the future of business is selling less of more**. Nova York, EUA: Hyperion, 2006. 238 p.

[MELO, 2007] MELO, C., ARCOVERDE, D., MORAES, E., PIMENTEL, J., FREITAS, R. **Software como Serviço: Um Modelo de Negócio Emergente**. Centro de Informática – Universidade Federal de Pernambuco (UFPE). Publicado em 2007. Disponível em: <http://www.cin.ufpe.br/~jhcp/publica/jhcp-saas.pdf>. Acesso em: 30 mai 2011.

[Creese, 2010] CREESE, G. **SaaS vs. Software: The Release Cycle for SaaS Is Usually (Not Always).** Faster.Gartner blog. Gartner, Inc. Disponível em: <http://blogs.gartner.com/guy-creese/2010/05/18/saas-vs-software-the-development-cycle-for-saas-is-usually-not-always-faster/>. Acesso 24 Abr 2011.

[BONDI, 2001] BONDI, A. **Characteristics of scalability and their impact on performance**. Workshop on Software and Performance. Proceedings of the 2nd International Workshop on Software and Performance. Ottawa, Ontario, Canada: 2001. p. 195-203.

[AMORIN, 2008] AMORIN, T. Técnicas **e Oportunidades de Negócio na Web 2.0.** Trabalho de Conclusão de Curso (TCC) - Universidade Federal de Pernambuco – Graduação em Ciência da Computação, Recife. Disponível em: <http://www.cin.ufpe.br/~tg/2007-2/tlba.pdf>. Acesso 9 mai 2011.

[CARRARO, 2006] CARRARO, G., CHONG, F. **Software as a Service (SaaS): An Enterprise Perspective**. Microsoft Corporation. Disponível em: <http://MSDN.microsoft.com/en-us/library/aa905332(loband).aspx>. Acesso em: 15 mai 2011.

[SHU, 2010] SHU Z., Song MEINA, S. **An architecture design of life cycle based SLA management**. Advanced Communication Technology (ICACT), 2010 The 12th International Conference on , vol.2, no., pp.1351-1355, 7-10 Feb. 2010.

[ALHAMAD, 2010] ALHAMAD, M., DILLON, T., CHANG, E. **SLA-Based Trust Model for Cloud Computing.** Network-Based Information Systems (NBiS), 2010 13th International Conference on , vol., no., pp.321-324, 14-16 Sept. 2010.

[ANDRZEJAK, 2010] ANDRZEJAK, A., KONDO, D., SANGHO Yi. **Decision Model for Cloud Computing under SLA Constraints**. Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on , vol., no., pp.257-266, 17-19 Aug. 2010.

[MALKOWSKI, 2010] MALKOWSKI, S., HEDWIG, M., JAYASINGHE, D., PU, C., NEUMANN, D. **Cloudxplor: a tool for configuration planning in clouds based on empirical data**. In Proceedings of the 2010 ACM Symposium on Applied Computing, SAC '10, pages 391–398, New York, NY, USA. ACM.

[MACÍAS, 2010] MACÍAS, M., FITÓ, J., GUITART, J. **Rule-based SLA management for revenue maximisation in Cloud Computing Markets**. Network and Service Management (CNSM), 2010 International Conference on , vol., no., pp.354-357, 25-29 Oct. 2010.

[CORREIA, 2010] CORREIA, A., BRITO, F. **Defining and Observing the Compliance of Service Level Agreements: A Model Driven Approach**. Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the , vol., no., pp.165-170, Sept. 29 2010-Oct. 2 2010.

[ANDRIEUX, 2007] ANDRIEUX, A., et al. **Web Services Agreement Specification.** (WS-Agreement). Disponível em: < http://www.ogf.org/documents/GFD.107.pdf> Acesso 26 Abr 2011.

[CARRARO, 2006] CARRARO, G. **SaaS Simple Maturity Model**. Disponível em: < http://blogs.msdn.com/b/gianpaolo/archive/2006/03/06/544354.aspx>. Acesso 24 Abr 2011.

[KELLER, 2003] KELLER, A., LUDWIG, H. **The wsla framework: Specifying and monitoring service level agreements for web services**. J. Netw. Syst. Manage., 11:57–81.

[Open Group, 2004] The Open Group. **SLA Management Handbook - Enterprise Perspective.** Disponível em: http://www.afutt.org/Qostic/qostic1/SLA-DI-USG-TMF-

060091 SLA_TMForum.pdf. Acesso 24 Abr 2011.

[ZHANG, 2010] Zhang S; Song M. **An architecture design of life cycle based SLA management, Advanced Communication Technology**. (ICACT), 2010 The 12th International Conference on , vol.2, no., pp.1351-1355, 7-10 Feb. 2010.

[RITU, 2011] RITU, S., MANU, S. **Cloud SaaS and Model Driven Architecture**. International Conference on Advanced Computing and Communication Technologies (ACCT 2011).

[COCHINWALA, 2005] COCHINWALA, M., SHIM, H., WULLERT, R. **A model-driven approach to rapid service introduction**. *Integrated Network Management, 2005. IM 2005. 2005 9th IFIP/IEEE International Symposium on* , vol., no., pp. 659- 672, 15-19 May 2005

[XIAOYAN, 2010] XIAOYAN, J., YONG, Z., SHIJUN, L. **A Well-designed SaaS Application Platform Based on Model-driven Approach**. Gcc, pp.276-281, 2010. Ninth International Conference on Grid and Cloud Computing, 2010.

[SANZ, 2008] SANZ, M., ACUNA, J., CUESTA, C., ESPERANZA, M. **Defining Service-Oriented Software Architecture Models for a MDA-based Development Process at the PIM level**. Wicsa, pp.309-312, Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008), 2008.

[XIAO, 2009] Xu Xiao; Sun Hailong; Li Xiang; Zhou Chao. **A Basing on Model-Driven Framework of Service-Oriented Software Production Line.** Computational Intelligence and Design, 2009. ISCID '09. Second International Symposium on , vol.2, no., pp.139-145, 12-14 Dec. 2009.