AS ABSTRAÇÕES RELEVANTES NAS CAMADAS DE ARQUITETURA CLIENTE-SERVIDOR EM UM SISTEMA DISTRIBUÍDO.

Renato Silva José Camões

Resumo: A descentralização e o aumento de acessos a inúmeros sistemas possibilitaram novos estudos sobre a matéria de sistemas distribuídos. Com o objetivo de compartilhar recursos, surgiram novas arquiteturas e com elas grandes fontes de abstrações. Dentro deste cenário é possível que usuários e programadores façam os seus trabalhos sem ter que se preocupar com um cenário complexo que envolve, arquitetura, a interoperabilidede de plataformas heterogêneas e a comunicação dos processos entre as aplicações. Também não se torna significativo para esses usuários qual unidade de armazenamento está sendo usada para guardar os dados, importando somente, que esses dados estejam disponíveis. Este artigo apresenta uma pesquisa sobre as abstrações relevantes em um sistema distribuído que possibilitam uma transação cliente-servidor de forma transparente e menos complexa.

Palavras chaves: Abstração; Sistema distribuído; Sistema Operacional; Middleware; SGDB.

Abstract: *Decentralization and increased access to systems allowed new studies on the subject of distributed systems. With the purpose of sharing resources, new architectures have emerged and with them great sources of abstractions. In this scenario it is possible for users and developers to do their work without having to worry about a complex scenario that involves architecture, interoperabilidede of heterogeneous platforms and communication processes between applications. It also does not become significant for those users which storage unit is being used to store the data, importing only that these data are available. This article presents a survey of the relevant abstractions in a distributed system that enables a client-server transaction transparent and less complex.*

Keyword: *Abstraction; Distributed system; Operational system; middleware; Database management system.*

Introdução

Quando começou a surgir a computação moderna entre 1945 e 1985 os computadores ainda eram novidade para maioria das empresas e das pessoas. Com a evolução do sistema computacional nos anos 90 esses computadores passaram a ser recursos no auxílio administrativo de algumas empresas, ferramenta de assistência à pesquisas científicas e acadêmicas e uma "caixa" para salvaguarda de arquivos pessoais e corporativos, ainda que de maneira centralizada.

Em decorrência dessa evolução, os itens que faziam parte de um sistema centralizado, tornaram-se distribuídos em rede. As redes locais de computadores (*LANs*) e as redes de longa distância (*WANs*) permitem que centenas ou milhares de máquinas realizem intercâmbio de informações por meio delas.

Dentro desse contexto, Nadiminti, Assunção e Buyya (2006) relatam que *World Wide Web* é usado por milhões de pessoas todos os dias para vários fins incluindo e-mail , ler notícias , download de música, compras on-line ou simplesmente acessando informações sobre qualquer coisa. Usando um navegador *web* padrão , o usuário pode acessar informações armazenadas em servidores situados em qualquer parte do globo. Isto dá o ilusão de que toda esta informação é situada localmente no computador do utilizador . Na realidade , a *web* representa um sistema distribuído enorme que aparece como um único recurso para o usuário disponível com o clique de um botão.

Com a finalidade de propor um maior discernimento sobre esse e outros sistemas distribuídos, é essencial ter informações sobre os diferentes tipos de abstrações que compõem o sistema, quantas instâncias existem, e como os nós se comunicam com outros nós. Por este motivo, tomando-se como base os trabalhos de autores renomados em sistemas distribuídos, o presente trabalho tem por objetivo apresentar as abstrações sobre a configuração e a comunicação nas camadas de arquitetura cliente-servidor de um sistema distribuído.

Sistema Distribuído

Um sistema distribuído é aquele no qual os componentes localizados em computadores interligados em rede se comunicam e coordenam suas ações apenas passando mensagens. (COLOURIS, 2013)

Andrew Tanenbaun (2007) prefere fazer uma caracterização sem ser muito específica, relatando que um sistema distribuído é um conjunto de computadores independentes que se apresenta a seus usuários com um sistema único e coerente.

Além dessas, existem muitas definições de sistemas distribuídos, mas para alguns autores não existe um acordo entre elas e não se tornam satisfatórias. Existe, porém, um equívoco comum entre as pessoas quando se define sistemas distribuídos. Um sistema distribuído não é apenas outro nome para uma rede de computadores. No entanto, deve haver uma distinção. Um sistema distribuído é construído em cima de uma rede e tenta abstrair a existência de vários computadores autônomos, fornecendo os serviços necessários ao usuário ou para as aplicações desse sistema. (NADIMINTI, ASSUNÇAO, BUYYA, 2006)

Torna-se importante ressaltar que todas as definições sobre sistemas distribuídos tem aspectos importantes. Um dos aspectos é que os computadores são protagonistas desse sistema. Outro fator é que os computadores necessitam estar interligados em rede para uma comunicação. Por fim, outro fator essencial é a comunicação e a colaboração realizada entre os componentes e usuários do sistema.

A comunicação e colaboração entre esses elementos podem estar à qualquer distância. Eles podem estar em continentes separados, no mesmo prédio ou na mesma sala. (COULOURIS, 2013) Uma meta importante de um sistema distribuído é abstrair o fato de que seus processos e recursos estão fisicamente distribuídos por vários computadores. Desta forma, um sistema distribuído que é capaz de se apresentar a usuários e aplicações como se fosse apenas um único sistema de computador é denominado transparente. (TANENBAUN, 2007)

A transparência pode ser abordada como uma das principais abstrações, para um usuário

final ou para um programador de aplicativos, da separação dos componentes em um sistema distribuído, de modo que o sistema seja percebido como um todo, em vez de como uma coleção de componentes independentes. (COULOURIS, 2013)

De fato, para os usuários e operários de um sistema distribuído a abstração usa a estratégia de simplificação, em que detalhes concretos são deixados ambíguos, vagos ou indefinidos; assim uma comunicação efetiva sobre as coisas abstraídas requer uma intuição ou experiência comum entre o comunicador e o recipiente da comunicação. (MARGOLIS, 1955)

Com o propósito de entender as abstrações relevantes em um sistema distribuído esse trabalho utilizou o desenho de sistemas cliente/servidor que utiliza três camadas: um terminal que interage com o usuário, um servidor de aplicativo que contém a regra de negócio (comercial) e um gerenciador de recurso que armazena dados.

Arquitetura cliente/servidor em três camadas

Para Tanenbaun (2007) muitas aplicações clientes servidor podem ser construídas de acordo com três partes diferentes (figura 1): uma parte que manipula a interação com um usuário, uma parte que age sobre um banco de dados ou sistema de arquivos e uma parte intermediária que, em geral, contém a funcionalidade central de uma aplicação.

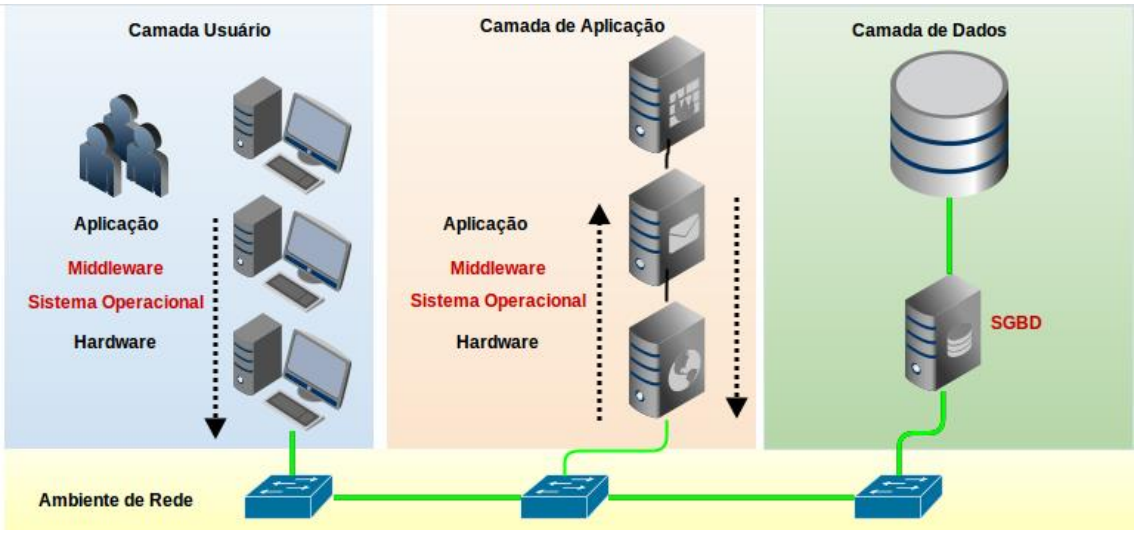

Figura 1 – Arquitetura cliente servidor em 3 camadas
Fonte: elaborado pelo autor

O nível de interface de usuário consiste em programas que permitem aos usuários finais interagir com aplicações por meio da GUI, que por sua vez é implementada por um programa mais simples caracterizando essa camada como *thin client*. "cliente magro". Em sua maioria a camada de usuário possui um computador que exibe a aplicação *front-end* em um *Web Browser*, ou em uma *GUI* específica. Já a camada lógica dos processos de negócio (camada intermediária) está em execução em servidores, integrada com a camada de clientes. Este é o

lugar onde transacção está em execução, ou onde os dados digitalizados são processados. (GLOOR, 2000) Por fim, a terceira camada provê os serviços de banco de dados e pode ser otimizado sem utilizar as linguagens proprietárias dos fabricantes. Um componente que manipula os dados garante que os dados estejam consistentes pelo ambiente distribuído, pelo uso de replicação, travamento de tabelas, e checagem de consistência. (DE OLIVEIRA, 2003)

Além disso a camada cliente interage por meio de uma aplicação que é implementada na máquina do usuário. Duas abordagens tornam-se importantes. A primeira é de um sistema operacional que dá suporte na implementação dessas aplicações, abstraindo toda gerência dos recursos do computador. A segunda trata da abstração entre a comunicação dos processos do cliente e os processos do negócio, que interage com a base de dados.

Abstrações na camada cliente (Sistema Operacional)

Conforme relatam Avi Silberschatz, Peter Baer Galvin e Greg Gagne (2008) um sistema distribuído é uma coleção de processadores que não compartilham memória ou um relógio. Na verdade, cada processador possui sua própria memória local. Os processadores se comunicam uns com os outros por meio de diversas redes de comunicação, como barramentos de alta velocidade e linhas telefônicas. Desta forma, ressaltam a importância dos recursos locais que dão início as solicitações dos clientes.

Possivelmente o início da comunicação se faz necessário uma interação entre máquina e homem. Surge então o fator de abstração que é o sistema operacional. Um sistema operacional é um programa que atua como intermediário entre o usuário de um computador e o hardware do computador. A finalidade do sistema operacional é prover um ambiente no qual um usuário possa executar programas de uma forma conveniente e eficiente.(SILBERSCHATZ et al, 2008)

Com a finalidade de atingir os objetivos propostos de forma conveniente e eficiente o sistema operacional oferece diversos tipos de serviços. Todo sistema operacional oferece meios para que um programa seja carregado na memória principal e executado. Talvez o serviço mais importante oferecido seja o que permite a utilização de arquivos e diretórios. Também o acesso aos periféricos é feito através do sistema operacional. À medida que diversos usuários compartilham o computador, passa a ser interessante saber quanto de quais recursos cada usuário necessita. Diversas informações sobre o estado do sistema são mantidas. (OLIVEIRA, CARISSIMI, TOSCANI, 2001)

Para garantir que todos esses serviços possam ser oferecidos o sistema operacional é dividido em gerencias que possibilitam as abstrações de usuários e programadores. Ressaltam-se três principais gerências do sistema operacional: gerência de processos; gerência de memória; e sistema de arquivos.

1 - A gerência de processos é responsável por criar e remover processos de usuário e de sistema, suspender e retornar os processos, prover mecanismos para o sincronismo de processos, prover mecanismos para a comunicação entre processos e prover mecanismos para o tratamento de *deadlook*. (SILBERSCHATZ et al, 2008)

2 – A gerência de memória tem como função gerir a memória de modo eficiente: manter o controle de quais partes da memória estão em uso e quais não estão, alocando a memória aos

processos quando eles precisam e liberando-a quando esses processos terminam. (TANENBAUN, 2007)

3 – A gerência sobre os arquivos vai tratar do armazenamento e a recuperação de informações que são atividades essenciais para qualquer tipo de aplicação. Apesar de ser a parte mais visível de um sistema operacional, pois a manipulação de arquivos é uma atividade frequentemente realizada pelos usuários, deve ocorrer de maneira uniforme, independente dos diferentes dispositivos de armazenamento. Portanto, torna-se necessário aspectos como identificação, organização, compartilhamento, métodos de acesso, proteção e operações de entrada e saída. (MACHADO, MAIA, 2004)

É importante perceber que essas são algumas atividades de um sistema operacional. Existe ainda os controles de entrada e saída, a segurança do sistema que permiti a autenticação e a integridade das informações. Esse trabalho destaca essas três porque o sistema computacional moderno consiste em pelo menos um processador, memória principal e discos. Só isso significa muito para desenvolvedores e usuários que não precisam manter o controle de todos esses componentes, deixando que o sistema operacional se encarregue disso.

Em suma um programa não faz nada a menos que suas instruções sejam executadas por uma *CPU*. Por isso a gerência de processos é importante. Uma atividade transparente para desenvolvedores e usuários, por exemplo, é a realização da alternância da *CPU* entre processos que acontece por meio de algorítimos de escalonamento. É através desse escalonamento que o processo, ou processos, mudam e estado de forma imperceptível para o usuário.

Imagine se cada programa entrasse na *CPU* da forma que quisesse? Por este motivo, o processo torna-se uma abstração de um programa em execução. Processos são uma das mais antigas e importantes abstrações que o sistema operacional oferece. Dentre elas, pode-se dizer que ele mantêm a capacidade de operações (pseudo)concorrentes, mesmo quando há apenas uma *CPU* disponível. Sem abstração de processos, a ciência da computação moderna não existiria. (TANENBAUN, 2007)

Além da gerência da *CPU* ressalta-se que em um ambiente distribuído múltiplas aplicações estarão na memória simultaneamente na esperança de que não ocorra interferência bilateral. Para isso a gerência de memória preocupa-se em realizar a proteção da memória (para que um processo não invada o endereço de outro processo) e a realocação (desocupar endereços não utilizados e ocupá-los com as novas instruções). O sistema operacional faz a abstração para memória, assim como o conceito de processo cria um tipo de *CPU* abstrata para executar programas, o espaço de endereçamento cria um tipo de memória abstrata para abrigá-los.

Já o sistema de arquivos tem a característica de prover mecanismos para garantir que os arquivos possam ser operados somente por aqueles processos que obtiveram a devida autorização do sistema operacional. O conceito de arquivo, muito mais útil que o simples espaço em disco, é uma abstração criada pelo sistema operacional. Nesse caso, temos o sistema operacional criando um recurso lógico a partir dos recursos físicos existentes no sistema computacional, propiciando a utilização menos complexa para os usuários.(OLIVEIRA, CARISSIMI, TOSCANI, 2001)

É nessa condição que o sistema operacional torna possível a interação entre um terminal e um usuário. Por meio do sistema operacional a complexidade de escalonar e sincronizar processos, e evitar *deadlock* entre eles é exequível. Através do sistema operacional é descomplicado alocar memória, realizar paginação, segmentação e *swapping* que possibilitam vários programas distribuídos compartilhar de um recurso caro e escasso em máquinas. Só com a capacidade de abstração do sistema operacional é possível realizar a troca de arquivos de diferentes sistemas e com diferentes tipos de armazenamento, acesso, nomeação e segurança.

Por consequência, a camada cliente, (instalações, nós, computadores, máquinas ou hosts) requerente de recursos, com uma interface que geralmente é o sistema operacional, fica encarregada de realizar a apresentação, ou seja, solicita um serviço, através de um programa, que faz uma chamada ao sistema operacional, que por sua vez processa a requisição (local), que posteriormente é enviada para um servidor (remoto) e permite que o cliente solicitante fique livre para realizar outras tarefas.

Assim sendo a requisição será processada na próxima camada nominada de camada de aplicação. O servidor de aplicação (designado *middleware*), se encarrega de prover o recurso.

Abstrações na camada de aplicação (*middleware*)

O software de computador correu tradicionalmente em sistemas isolados, onde a interface com o usuário, o aplicativo (negócio) e o processamento de dados, eram persistentes e residiam em um computador, com periféricos ligados a ele por barramentos ou cabos. Poucos sistemas interessantes, no entanto, ainda são concebidos desta forma. Em vez disso, a maioria de software de computador hoje é executado em sistemas distribuído, onde a apresentação interativa, processamento de negócios e recursos de dados residem vagamente acoplados em nós de computação e níveis de serviços ligados entre si por redes. (SCHMIDT, 2009)

Tanenbaun (2007) relata que para que a transparência e outras abstrações aconteçam, deve-se separar as aplicações das plataformas subjacentes provendo uma camada intermediária (*middleware*). O autor explica que o *middleware* é uma aplicação que reside logicamente, na maioria das vezes, na camada de aplicação, mas que contém muitos protocolos de uso geral que justificam suas próprias camadas, independentemente de outras aplicações mais específicas.

O *middleware* foi uma contribuição fundamental para o desenvolvimento do sistema distribuído. O surgimento da computação de objetos distribuídos (DOC) *middleware* no final de 1980 e início de 1990 representava a confluência das principais informações tecnologias. (SCHMIDT, 2009)

Para Philip Bernstein (1996) o *middleware* , se senta ''no meio'' acima do sistema operacional e software de rede e abaixo aplicativos específicos da indústria. Vendedores estão oferecendo serviços de sistema distribuído que têm interfaces padrão de programação e protocolos para ajudar a resolver problemas de heterogeneidade e computação distribuída. A programação padrão de interfaces facilitam com que às aplicações executem em uma variedade de plataformas, dando ao cliente e ao fornecedor de independência e abstração.

Para o autor a maioria dos serviços de *middleware* são distribuídos. Isto é, um serviço de middleware geralmente inclui uma parte no cliente, que suporta a Interface de Programação de Aplicação *API* do serviço em execução no aplicativo (local), e uma parte servidor, que suporta funções principais do serviço e pode executar em um sistema diferente (remoto). Pode haver múltiplas implementações de cada parte.

Ademais a maioria dos serviços de *middleware* pode rodar em múltiplas plataformas, aumentando assim a cobertura de aplicações que dependem desses serviços. Se o serviço é distribuído, isso também melhora a interoperabilidade, possibilitando que aplicações em diferentes plataformas possam usar o serviço para se comunicar e / ou trocar dados. Para ter uma boa cobertura, os serviços de *middleware* são normalmente programados para ser portátil significando a portabilidade para outra plataforma com um esforço modesto e previsível. (BERNSTEIN, 1993)

O que ocorre é que muitos sistemas são baseados em troca explícita de mensagens entre processos. Quando um processo na máquina "A" (cliente) chama um procedimento na máquina "B" (servidor), o processo chamador em "A" é suspenso, e a execução do procedimento chamado ocorre em "B". Informações podem ser transportadas do chamador para que foi chamado nos parâmetros e podem voltar no resultado do procedimento. (TANEMBAUM, 2007)

De certo a abstração ocorre porque absolutamente nada da troca de mensagens é visível para o programador ou para o usuário. A ideia é fazer com que uma chamada de procedimento remoto pareça o mais possível uma chamada local. Em outras palavras ocorre uma abstração no qual o procedimento de chamada não deve estar ciente de que o procedimento chamado está executando em uma máquina diferente ou vice-versa.

Ao longo dos anos, tecnologias de *middleware* como *RPC* (chamada de procedimento remoto) e Java *RMI* (invocação de método remoto) tem fornecido os meios para construir sistemas baseados em componentes distribuídos. Estas tecnologias permitem que os sistemas possam interagir ao nível do componente, por proporcionar uma abstração de protocolos e software que oferece interoperabilidade necessária para componentes desenvolvidos na programação de diferente línguas para a troca mensagens. (NADIMINTI, ASSUNÇAO, BUYYA, 2006)

A *RPC* e a Java *RMI* são tecnologias que focam em interfaces, que são contratos entre clientes e servidores que definem a localização, meios independentes para os clientes acessarem a objetos fornecidos por um servidor. O *middleware* também definem os protocolos de comunicação e objeto modelos de informação para permitir interoperabilidade entre aplicações heterogêneas escritos em várias línguas que funcionam em várias plataformas. (SCHMIDT, 2009)

A semântica dessas tecnologias permitem que um cliente chame um procedimento ou objeto em um *host* remoto da mesma forma como chamaria um procedimento ou objeto local. Esses sistemas de *middleware* escondem os detalhes que permitem a comunicação, oferecendo um *stub*[1] no lado do cliente (figura 2). Normalmente, existe um *stub* separado para cada procedimento ou objeto remoto. Quando um o cliente chama um procedimento remoto, o *middleware* chama o *stub* apropriado, passando-lhe os parâmetros fornecidos ao

[1] São partes (fragmentos) de algoritmos que provêm a abstração de uma chamada (local) de procedimento (método) fazendo a ligação deste com o mecanismo de comunicação

procedimento remoto. Esse *stub* localiza a porta no servidor e empacota os parâmetros. O empacotamento de parâmetros envolve a inclusão dos parâmetros em uma forma que possa ser transmitida por uma rede. O *stub* transmite, então, uma mensagem ao servidor usando a passagem de mensagens. Um stub semelhante, no servidor, recebe essa mensagem e chama o procedimento no servidor. Se for preciso, valores de retorno são passados de volta ao cliente, usando a mesma técnica.

Imagine se o cliente tivesse que realizar todo esse serviço, ou seja, criar uma mensagem traduzida para o outro sistema, na busca de um arquivo ou de um serviço remoto? Independente da tecnologia o que é realmente relevante é que o *middleware* surgiu como uma parte crítica da a infra-estrutura de tecnologia da informação. Sem ele seria complexo mascarar a heterogeneidade das aplicações que são feitas de várias partes distribuídas em execução em diferentes redes de computadores e sistemas de telecomunicações. Além disso, sistemas de middleware fornecer serviços de valor acrescentado tais como serviços de nomeação e de transação, como bem como ferramentas e APIs que oferecem interfaces de nível elevado e uniforme para os desenvolvedores de aplicativos para que os aplicativos possam ser facilmente construídos. (MAHMOUD, 2004)

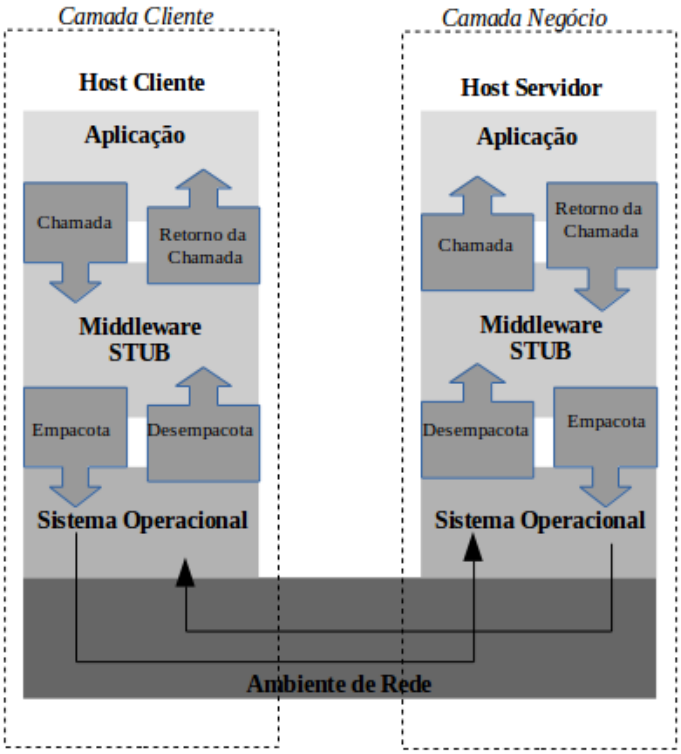

Figura 2 – A comunicação em camadas entre aplicação cliente-servidor

Fonte: elaborado pelo autor

Sobretudo cabe ressaltar que o sistema operacional também está presente na camada de aplicação realizando as abstrações importantes que foram mencionadas na camada cliente. Ressalta-se ainda que todo o acesso do cliente ao banco de dados, é feito de acordo com as regras contidas na camada de aplicações. O cliente não tem acesso direto ao banco de dados,

sem antes passar pelo servidor de aplicações. Com isso é necessário também entender as abstrações que ocorrem na terceira camada.

Abstrações na camada de dados (SGBD)

Conforme já citado neste trabalho o modelo cliente/servidor, nasceu para descentralizar os dados e recursos de processamento, em oposição ao modelo centralizado utilizado pelos mainframes. Um programa era instalado no cliente (com auxilio do *middleware*) e fazia acesso a um servidor de aplicação no qual também residia o banco de dados. Com a evolução, o banco de dados ficou isolado e o seu o acesso passou a ser realizado através das regras contidas no servidor de aplicações.

Emerge então o Sistema de Gerenciamento de Banco de Dados (SGBD) com o objetivo de segregar o que é vizualizado na camada do usuário dos detalhes internos da camada de banco de dados. O propósito é promover a abstração de armazenagem e manipulação dados e também favorecer a independência dos dados em relação às aplicações.

O SGDB deve assegurar um cenário abstrato do banco de dados para o usuário e para as aplicações (figura 3), ou seja, para estes não se torna significativo qual unidade de armazenamento está sendo usada para guardar os dados, importando somente, que esses dados estejam disponíveis no momento necessário. (SILBERSCHATZ, KORTH, SUDARSHAN, 2006)

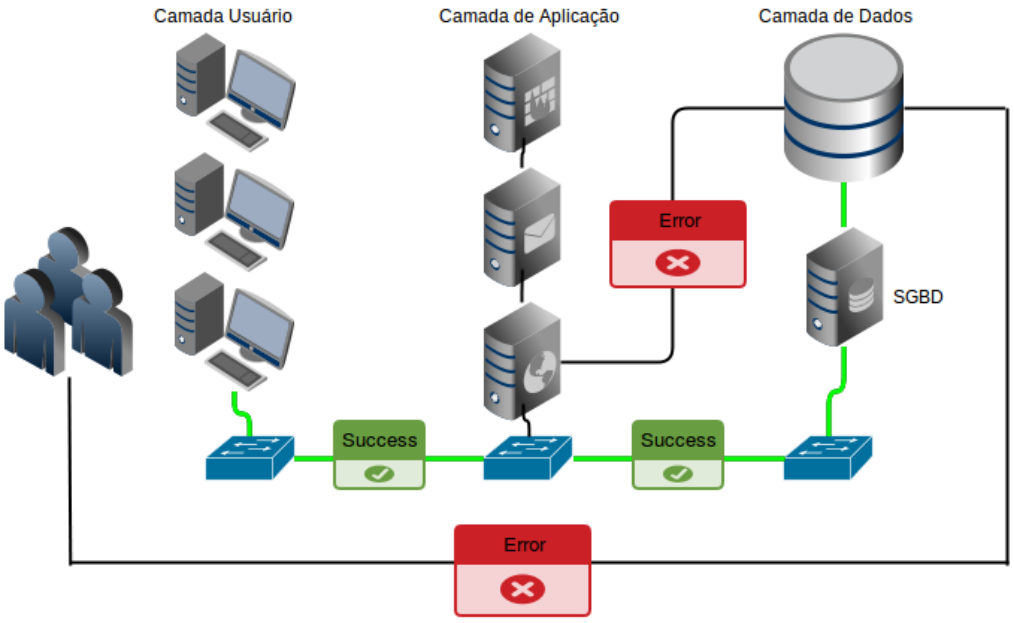

Figura 3 - Segregação da camada de usuáro da camada de dados

Fonte: elaborado pelo autor

Como resultado, sob um ponto de vista de mais alto nível, um sistema desse tipo pode ser considerado como tendo uma estrutura muito simples, consistindo em um servidor, também chamado *back end*, e um conjunto de clientes, também chamados *front ends.* (DATES, 2004)

O servidor é o próprio SGBD. Conforme Christopher Dates (2004) o SGDB vai realizar as seguintes funções:

1 – Definção de dados: O SGBD deve ser capaz de aceitar definições de dados em formato fonte e convertê-los para o formato objeto apropriado.

2 – Manipulação de dados: O SGBD deve ser capaz de lidar com requisições do usuário para buscar, atualizar ou excluir dados existentes no banco de dados, ou para acrescentar novos dados ao banco de dados.

3 – Otimização e execução: O SGBD deve ser capaz de otmizar as requisições e executar sob o controle do gerenciador em tempo de execução.

4 – Segurança e integridade de dados: O SGBD deve monitorar requisições de usuários e rejeitar toda a tentativa de violar as restrições de segurança e integridade definidas pelo DBA[2]. Essas tarefas podem ser executadas em tempo de compilação ou em tempo de execução, ou ainda em alguma combinação dos dois.

5 – Recuperação de dados e concorrência: Deve impor controle de recuperação que restaure o banco de dados a um estado que se sabe ser correto depois que alguma falha leva a um estado incorreto ou, pelo menos, suspeito. A concorrência se refere ao fato de que os SGBDs em geral permitem que muitas transações tenham acesso ao mesmo banco de dados ao mesmo tempo.

6 – Dicionário de dados: é considerado um banco de dados isolado do sistema e contém "dados sobre dados" também chamado de metadados ou descritores, ou seja, definições de outros objetos do sistema, em vez de somente "dados crus".

7 – Desempenho: critério que vai realizar todas as funções identificadas anteriormente de forma eficiente.

Nota-se que o SGBD é um software capaz de manipular as informações do banco de dados e interagir com o usuário. Diante disto, uma abstração neste modelo é que, o cliente não precisa ter informações sobre o gerenciador de recursos real. Qualquer alteração no banco de dados, por exemplo, pode ser transparente ao usuário, visto que, à aplicação pode ser alterada em seus servidores, mas o cliente não precisa ser modificado.

Esse cenário torna-se mais abstrato quando o suporte completo para banco de dados distribuídos implica em uma única aplicação que deve ser capaz de operar de 'modo transparente' sobre dados diferentes, gerenciados por vários SGBDs diferentes, em execução em uma variedade de máquinas diferentes, admitidos por uma variedade de sistemas operacionais diferentes e conectados entre si por uma variedade de redes de comunicações diferentes – onde 'de modo transparente' significa que a aplicação opera de um ponto de vista lógico como se os dados fossem todos gerenciados por um único SGBD, funcionando em uma única máquina. (DATES, 2004)

Em resumo o objetivo de um SGBD é garantir aos usuários visão abstrata dos dados, ou seja, omite particularidades de como os dados são armazenados e conservados.

2 Pessoa que fornece o suporte técnico necessário para implementar essas decisões estratégicas e de normas com relação aos dados da empresa. (DATES, 2004)

Igualmente a outras camadas a camada de dados também tem um sistema operacional e principalmente um *middleware* de acesso a dados como , *DataJoiner, JBOSS e Java Database Connectivity.*

Outras abstrações importantes

Não fazem parte do escopo deste trabalho as abstrações realizadas na comunicação de rede e também as abstrações de segurança em um sistema cliente/servidor em um sistema distribuído. Porém destaca-se que essas abstrações são muito importantes, porque assim como as abstrações mencionadas anteriormente, tornam o acesso do usuário transparente dando a sensação de um acesso local e simples.

Cabe um outro trabalho para falar sobre os princípios nos quais as redes de computadores são baseados incluindo à organização de protocolos em camadas, comutação de pacotes, roteamento e fluxo de dados. Cabe ainda falar sobre as técnicas de interligação em rede que possibilitam a integração de redes heterogêneas.

Também existe a necessidade de se falar sobre as abstrações de segurança que são dedicadas para garantir a privacidade, a integridade e a disponibilidade dos recursos em sistemas distribuídos. Por isso, outras abstrações devem ocorrer para estabelecer a transparência e confiança entre os usuários.

Considerações Finais

Torna-se inimaginável o funcionamento de instalações, nós, computadores, máquinas ou *hosts* sem a existência das abstrações realizadas pelo sistema operacional. A gerência do sistema operacional é extremamente relevante para que a máquina funcione de modo profícuo, produtivo e protegido. Sem as abstrações do sistema operacional todo usuário e programador teriam que ter o completo entendimento do hardware, dificultando o uso das aplicações com a necessidade de desvendar e atuar nas complexidades de funcionamento de cada hardware. O sistema operacional surgiu pra realizar abstrações que tornam a interação do usuário com o computador mais inteligível.

Tanto quanto as abstrações do sistema operacional também manifesta-se o *middleware*. O aumento do crescimento em aplicativos baseados em rede, sistemas de comunicações tais como redes de informática e telecomunicações são compostas de um conjunto de dispositivos heterogêneos cujas candidaturas precisam interagir. A importância do *middleware* vai continuar a crescer enquanto os sistemas de computação e comunicações continuarem a ser heterogéneos. Entende-se que o principal objetivo do *middleware* é resolver os problemas de conectividade entre diferentes aplicações, bem como, as inconvenientes interoperabilidades dos sistemas. O *middleware* simplifica a interação e desempenha um papel estratégico na organizações como ambiente distribuído, permitindo também a garantia dos investimentos dos sistemas atuais.

Além disso torna-se importante o sistema de gernciamento de banco de dados para agregar no conjunto de abstrações na arquitetura cliente-servidor em um sistema distribuído. O SGBD, surge para proscrever o trabalho que antigamente um programador de aplicação

realizava controlando o acesso, integridade e redundância dos dados. A identificação, realizada pelo SGBD, pode tratar ou antecipar esses problemas ocasionados nesses serviços evitando a perda de dados gerados e armazenados por organizações de qualquer magnitude. Ressalta-se que as abstrações realizadas pelo SGBD podem proporcionar maior flexibilidade e agilidade e também pode significar menos tempo gasto com a administração do banco de dados.

Por fim, é importante evidenciar que não existe uma camada mais importante do que outra. O que de fato acontece é que ambas as camadas se completam e fornecem o seu serviço de forma abstrata para os usuários e programadores do sistema.

Referências

BERNSTEIN, Philip A. **Middleware an architecture for distributed system services. In: Communications of the ACM**. 1993.

BERNSTEIN, Philip A. **Middleware: a model for distributed system services. Communications of the ACM**, v. 39, n. 2, p. 86-98, 1996.

COULOURIS, George et al. **Sistemas Distribuídos: Conceitos e Projeto**. Bookman Editora, 2013.

DATE, Christopher J. **Introdução a sistemas de bancos de dados**. Elsevier Brasil, 2004.

DE OLIVEIRA, Henrique Eduardo Machado. **Aplicativo Cliente-Servidor multicamadas para controle de uma rede de lojas via WEB utilizando Java**. 2003. Tese de Doutorado. Universidade Federal de Santa Catarina.

GLOOR, Peter. **Making the e-business Transformation. Springer Science & Business Media**, 2012.

MACHADO, Francis Berenger; MAIA, Luiz Paulo. **Arquitetura de sistemas operacionais**. LTC, 2004.

MAHMOUD, Qusay H. (Ed.). **Middleware for communications**. J. Wiley & Sons, 2004.

MARGOLIS, Joseph. **Feeling and Form; a Theory of Art Developed from Philosphy in a New Key**. The Journal of Philosophy, v. 52, n. 11, p. 291-296, 1955.

NADIMINTI, Krishna; DE ASSUNÇAO, Marcos Dias; BUYYA, Rajkumar. **Distributed systems and recent innovations: Challenges and benefits**. InfoNet Magazine, v. 16, n. 3, p. 1-5, 2006.

OLIVEIRA, Rômulo Silva de; CARISSIMI, Alexandre da Silva; TOSCANI, Simão Sirineo. **Sistemas operacionais**. Revista de informática teórica e aplicada. Porto Alegre. Vol. 8, n. 3 (dez. 2001), p. 7-39, 2001.

SILBERSCHATZ, Abraham. **Sistemas operacionais com Java**. Elsevier Brasil, 2008.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S. **Sistema de banco de dados**. Elsevier, 2006.

SCHMIDT, Douglas C. **Software Technologies for Developing Distributed Systems: Objects and Beyond.** Vanderbilt University, 2009.

TANENBAUN, Andrew; STEEN, Maarten. **Sistemas Distribuídos, princípios e paradigmas**. 2ª ed Pearson Prentice Hall, 2007.

VARELA, Matías Leandro. **Conceptos fundamentales de un Middleware y razones de su importância en el mundo de hoy.** UNLAM, Universidad Nacional de La Matanza, Argentina, 2007.