UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DE CIÊNCIAS EXATAS E ENGENHARIAS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RODRIGO LUTZ

SISTEMA DE GERENCIAMENTO DE REPLICAÇÃO PARA PDV (PONTO DE VENDA) DE MULTI LOJAS

BENTO GONÇALVES

2021

RODRIGO LUTZ

SISTEMA DE GERENCIAMENTO DE REPLICAÇÃO PARA PDV (PONTO DE VENDA) DE MULTI LOJAS

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do Título de Bacharel pela Universidade de Caxias do Sul. Área de concentração: Ciência da Computação.

Orientador: Prof. Dr. Daniel Luis Notari

BENTO GONÇALVES

2021

RODRIGO LUTZ

SISTEMA DE GERENCIAMENTO DE REPLICAÇÃO PARA PDV (PONTO DE VENDA) DE MULTI LOJAS

Trabalho de Conclusão de Curso apresentado como parte dos requisitos para obtenção do Título de Bacharel pela Universidade de Caxias do Sul. Área de concentração: Ciência da Computação.

Aprovado em 02/12/2021

BANCA EXAMINADORA

Prof. Dr. Daniel Luis Notari
Universidade de Caxias do Sul - UCS

Prof. Dra. Helena Graziottin Ribeiro
Universidade de Caxias do Sul - UCS

Prof. Msc. Alexandre Erasmo Krohn Nascimento
Universidade de Caxias do Sul - UCS

Dedico este trabalho a minha família que me incentivou e especialmente a minha esposa que de uma forma e outra, me apoiou para que conseguisse atingir o objetivo final.

RESUMO

A medida que a empresa evolui, o volume de dados cresce a cada dia, e é fundamental para qualquer empresa repensar na sua organização, identificando melhorias necessárias e prevendo a capacidade da empresa de operar de forma ágil e consistente ao longo dos anos. O presente trabalho apresenta o desenvolvimento de um sistema de gerenciamento de replicação de forma assíncrona, utilizando banco de dados *PostgreSQL*. O objetivo é replicar dados de um sistema responsável pelo fechamento das vendas dos consumidores que chegam diretamente na loja para comprar. Todos dados são centralizados em um servidor central e cada computador contém apenas dados necessários para a emissão de uma nota fiscal. Para a elaboração desse sistema, é apresentada de forma resumida o funcionamento das arquiteturas utilizadas para a replicação de dados Bidirecional e Cliente-Servidor. O sistema de gerenciamento de replicação apresentou bons resultados, garantindo a confiabilidade e a consistência dos dados.

Palavras-chave: Gerenciamento de Replicação. Organização. Comunicação Assíncrona. PostgreSQL. Bidirecional. Cliente-Servidor.

ABSTRACT

As a company evolves, the volume of data grows every day, and it is critical for any company to rethink its organization, identifying necessary improvements and anticipating the company's ability to operate in an agile and consistent manner over the years. The present work presents the development of an asynchronous replication management system, using a PostgreSQL database. The objective is to replicate data from a system responsible for closing the sales of consumers who come directly to the store to buy. All data is centralized in a central server and each computer contains only the data needed to issue an invoice. For the elaboration of this system, the architectures used for the replication of Bidirectional and Client-Server data are briefly presented. The replication management system presented good results, ensuring data reliability and consistency.

Keywords: Replication Management. Organization. Asynchronous. PostgreSQL. Bidirectional. Client-Server.

LISTA DE FIGURAS

Figura 1 – Modelo de arquitetura básica para a replicação 13
Figura 2 – Arquitetura Cliente-Servidor 15
Figura 3 – Arquitetura Cliente-Servidor em três camadas 15
Figura 4 – Infraestrutura de um Sistema Distribuído 17
Figura 5 – Modelo *Middleware* 18
Figura 6 – Modelo de arquitetura *Object Request Broker* (ORB) 19
Figura 7 – *Socket* com serviço *Transmission Control Protocol* (TCP) 20
Figura 8 – Modelo *Java* RMI 20
Figura 9 – Arquitetura *Web Services* 21
Figura 10 – Exemplo de comando *GET* de uma arquitetura *Representational State Transfer* (REST) 22
Figura 11 – Método de integração Bidirecional 24
Figura 12 – Comunicação via *Socket* TCP/IP 25
Figura 13 – Organização atual 27
Figura 14 – Sistema Ponto de Venda (PDV) 28
Figura 15 – Sincronização dos dados 29
Figura 16 – Diagrama de caso de uso do Gerenciamento de Replicação 32
Figura 17 – Interface do Gerenciamento de Replicação 33
Figura 18 – Arquitetura Bidirecional 34
Figura 19 – Bidirecional interna na loja 34
Figura 20 – Arquitetura Cliente-Servidor 35
Figura 21 – Diagrama de Classes 37
Figura 22 – Classes utilizadas no projeto 38
Figura 23 – Diagrama de Banco de Dados do sistema PDV 39
Figura 24 – Arquivo de configurações de inicialização 41
Figura 25 – Tabela dados da loja 41
Figura 26 – Tabela informações do sistema PDV 42
Figura 27 – Processo Bidirecional 43
Figura 28 – Processo Cliente-Servidor 44
Figura 29 – Configuração da porta do *Socket* 45
Figura 30 – Tratamento de erros 45
Figura 31 – Telas de *Logs* do *Socket* cliente e servidor 46
Figura 32 – Botão *Start*, *Stop* e *Clear* 46

LISTA DE TABELAS

Tabela 1 – Principais SGBDs e suas características 22

LISTA DE ABREVIATURAS E SIGLAS

CEFET/RJ Centro Federal de Educação Tecnológica do Rio de Janeiro

DII Interface de Invocação Dinâmica

DSI *Dynamic Skeleton Interface*

HTTP *Hypertext Transfer Protocol*

IDE *Integrated Development Environment*

IBM *International Business Machines Corporation*

IP *Internet Protocol*

ORB *Object Request Broker*

NFC Nota Fiscal de Consumidor Eletrônica

MMOGs *Massively Multiplayer Online Games*

PDV Ponto de Venda

REST *Representational State Transfer*

TCP *Transmission Control Protocol*

UDP *User Datagram Protocol*

VPN Rede Virtual Privada

SUMÁRIO

1 INTRODUÇÃO 11
1.1 PROBLEMA DA PESQUISA 12
1.2 OBJETIVO GERAL 12
1.3 ESTRUTURA DO TRABALHO 12

2 REFERENCIAL TEÓRICO 13
2.1 REPLICAÇÃO 13
2.1.1 Replicação Síncrona e Assíncrona 14
2.2 SISTEMA CENTRALIZADO 14
2.2.1 Cliente-Servidor 14
2.3 SISTEMA DISTRIBUÍDO 16
2.3.1 *Middleware* 17
2.3.2 ORB 18
2.3.3 *Sockets* 19
2.3.4 *Java* RMI 20
2.3.5 *Web Services* 20
2.3.6 REST 21
2.4 BANCO DE DADOS 22
2.5 TRABALHOS RELACIONADOS 23
2.5.1 Ministério de Defesa do Reino Unido 23
2.5.2 *Massively Multiplayer Online Games* (MMOGs) 24
2.5.3 Um sistema de comunicação via *Socket* 24
2.6 CONSIDERAÇÕES DO CAPÍTULO 26

3 ORGANIZAÇÃO ATUAL 27
3.1 CONSIDERAÇÕES DO CAPÍTULO 29

4 APLICAÇÃO 30
4.1 REQUISITOS DA APLICAÇÃO 30
4.2 INTERFACE 32
4.3 ARQUITETURA DE SOFTWARE 33
4.4 DIAGRAMA DE CLASSES 36
4.5 MODELO DE DADOS 38
4.6 CONSIDERAÇÕES DO CAPÍTULO 39

5 TESTE DA APLICAÇÃO 40
5.1 PRÉ-CONFIGURAÇÃO 40

5.2 INSTRUÇÃO 42
5.3 PROGRESSO 43
5.4 REGISTRO 45
5.5 AMBIENTE DE TESTE 47

6 CONCLUSÃO 48

REFERÊNCIAS 49

1 INTRODUÇÃO

Devido à constante mudança e transformação de novas tecnologias que atinge todas as atividades baseadas em informação, Costa & Rosa (2020) afirmam que uma grande dificuldade é manter a capacitação competitiva da empresa no mercado. Portanto, o gerenciamento e o controle sobre as informações se tornou indispensável para a organização. A tecnologia se transformou em um elemento chave, e tem contribuído no processamento de diferenciação no mercado.

De acordo com Batista (2017), a procura por mecanismos que auxiliam os profissionais a definir a melhor estratégia se tornou essencial para as empresas que exigem ter todas as informações mais condizentes possível com o acontecimento. Em pouco tempo, as informações se tornam inapropriadas para ser válidas para uma decisão tardia.

Segundo Falsarella & Jannuzzi (2017), a construção de um plano estratégico e a qualidade das informações adquirida para esse propósito são elementos fundamentais para se posicionar na frente de seus concorrentes nesse mercado competitivo. Dessa forma, a agilidade nas demandas do mercado contribui para a sobrevivência das organizações.

Veras (2015) aponta que as partes de uma infraestrutura da empresa precisam ser pensadas para garantir ganhos de escala e otimizações de recursos. Portanto, esse esforço permite obter a flexibilidade necessária para atingir seus objetivos.

> Para a implementação destas ferramentas fica evidente a importância de se conhecer a empresa, através de sua estrutura, recursos e processos. A inserção da empresa no mercado, seus objetivos, metas e estratégias devem ser levantados para verificar em que medida elas buscam ou necessitam de vantagens competitivas (MORAES *et al.*, 2018).

Moura *et al.* (2018) identificou que foi possível obter vantagens com investimento em tecnologia para o sucesso na estratégia empresarial, devido a redução do tempo de execução das tarefas, melhor desempenho na performance e otimização dos processos. Caso contrário, a empresa deixa passar várias oportunidades de modernizar seus processos.

De acordo com Cunha (2020), é necessário investir para ter um bom sistema de gerenciamento de replicações de dados da empresa, pois a falta de dados leva a desorganização e ao desconhecimento das informações relevantes. É mais importante investir em tecnologia correta e conveniente com as necessidades da organização empresarial.

1.1 PROBLEMA DA PESQUISA

O estudo desse trabalho é baseado em uma empresa de varejo, a qual atualmente conta com quatorze lojas espalhadas pelo Brasil e dez lojas no exterior localizadas no Chile, Colômbia e Peru. A maioria das lojas se encontram dentro de grandes centros comerciais como *Shopping Centers*. Ao longo do desenvolvimento desse trabalho, iremos identificá-la como a empresa *STORE*.

A administração das lojas é realizada pelo departamento central da empresa *STORE*. Atualmente, o sistema realiza o acesso diretamente no banco de dados do servidor localizado em cada loja. Dessa forma, o sistema retorna informações sobre uma única loja. Este departamento exige que sejam centralizadas as informações de todas as lojas.

1.2 OBJETIVO GERAL

Este trabalho tem como objetivo criar um sistema de gerenciamento de replicação para PDV (Ponto de Venda), utilizando replicação de dados assíncrona através do uso das arquiteturas Cliente-Servidor e Bidirecional.

1.3 ESTRUTURA DO TRABALHO

A estrutura do texto segue da seguinte forma: o Capítulo 2 explica os conceitos referente a replicação, sistema centralizado, distribuído, gerenciamento de banco de dados e aborda alguns trabalhos relacionados. No capítulo 3 demonstra a organização atual do sistema PDV das lojas. No capítulo 4 apresenta a aplicação que foi desenvolvida para este trabalho, utilizando conceitos de replicação de dados. No capítulo 5 mostra os testes da aplicação e suas principais funcionalidades. Por último, o capítulo 6 realiza o desfecho conclusivo do trabalho.

2 REFERENCIAL TEÓRICO

Neste capítulo são abordados aspectos relacionados a replicações de dados, sistema distribuído e sistema centralizado, apresentando um referencial teórico dos conceitos referentes ao tema. Na sequência, o capítulo também apresentará a análise dos trabalhos relacionados.

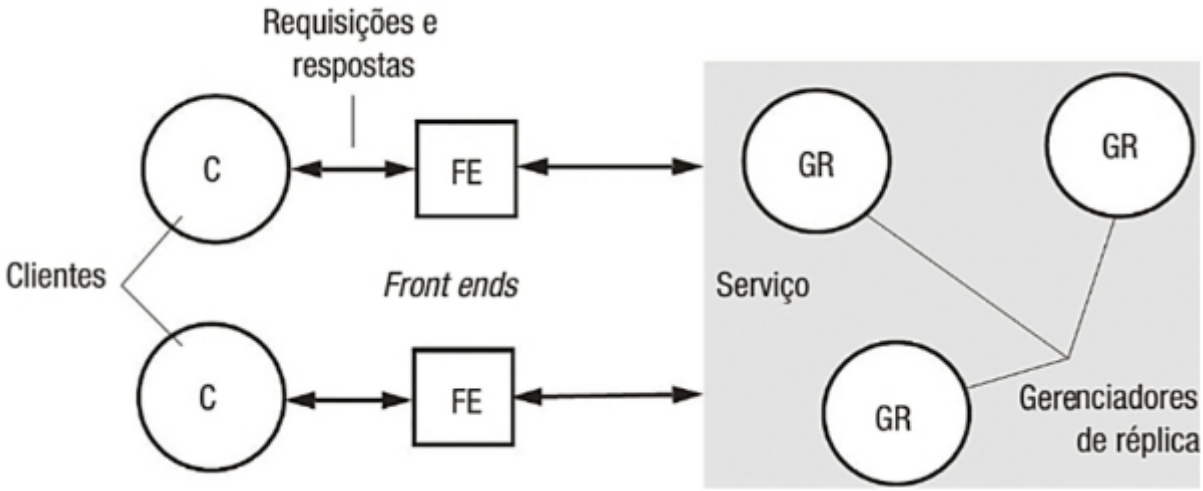
2.1 REPLICAÇÃO

De acordo com Neto (2015), a replicação é a cópia dos dados de um sistema para o outro, independente de discos. O resultado final da replicação é o conjunto de dados consistentes e iguais em discos localizadas em lugares diferentes. Além disso, em função da constante mudança imposta pelo ambiente, a solução de replicação deve ser flexível para se ajustar nas modificações.

Segundo Monteiro *et al.* (2020), na replicação de dados há chamadas de microsserviços, que replicam rotinas com pequenas tarefas. É fundamental para melhorar o desempenho, aumentar a disponibilidade de serviços e a tolerância a falhas.

Coulouris (2014) aponta que a replicação de dados deve manter suas réplicas consistentes, pois em caso de ocorrer alguma falha durante o processo é preciso desfazer a operação. Isso traz mais complexidade para a aplicação, principalmente para os sistemas tolerantes a falhas. O modelo de uma arquitetura de replicação é apresentada na Figura 1, que dispõe gerenciadores de réplicas, enviando requisições e respostas aos clientes através de *Front ends*.

Figura 1 – Modelo de arquitetura básica para a replicação



Fonte: Coulouris (2014).

2.1.1 Replicação Síncrona e Assíncrona

Existem duas maneiras de se realizar replicações, a forma síncrona e assíncrona. Segundo Cawende (2018), a principal diferença entre elas é como são gravadas as réplicas de dados. Na replicação síncrona, o sistema só fica disponível quando todos os dados são sincronizados, dessa forma, ocorre em tempo real. Por outro lado, a replicação assíncrona permite agendar o processo de replicação de tempos em tempos, por exemplo de minuto a minuto e comumente é um processo mais demorado.

2.2 SISTEMA CENTRALIZADO

Segundo Coulouris (2014), umas das vantagens de um sistema centralizado é que permite ter uma visão global das informações do sistema. Na medida em que o sistema aumenta surgem problemas, como o gargalo, ou seja, muitos usuários solicitando dados ao mesmo tempo.

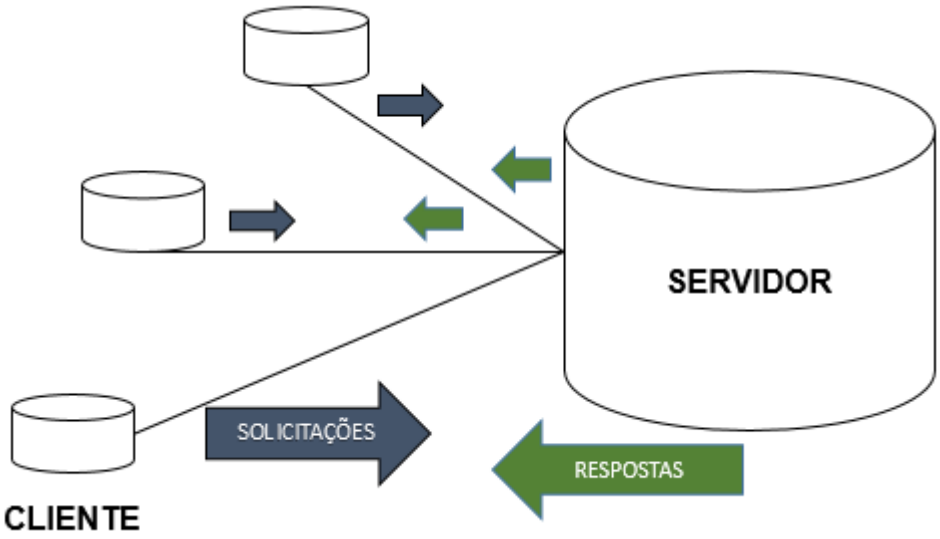
De acordo com Pichetti (2020), a possibilidade de vários usuários acessando simultaneamente o banco de dados, é um mecanismo importante, pois facilita o compartilhamento de dados. Na próxima seção será abordada uma arquitetura de gerenciamento de um sistema centralizado.

2.2.1 Cliente-Servidor

É um modelo computacional que desencadeia clientes e servidores, desenvolvida para lidar com grandes números de equipamentos como computadores, servidores de arquivos, banco de dados, web, correio eletrônico, estações de trabalho e outros softwares que estão interligados na rede da empresa. Um cliente é uma máquina que oferece interface com o usuário e processamento atual. Quando um cliente solicita uma funcionalidade que não existe no computador próprio, é conectado com o servidor centralizado que disponibiliza a funcionalidade requisitada. Um servidor é um sistema que oferece um determinado serviço aos clientes, como acesso aos arquivos, impressões, arquivamento e acesso a banco de dados (ELMASRI; NAVATHE, 2018).

A Figura 2 apresenta uma representação gráfica de uma arquitetura Cliente-Servidor, onde clientes solicitam dados para o servidor.

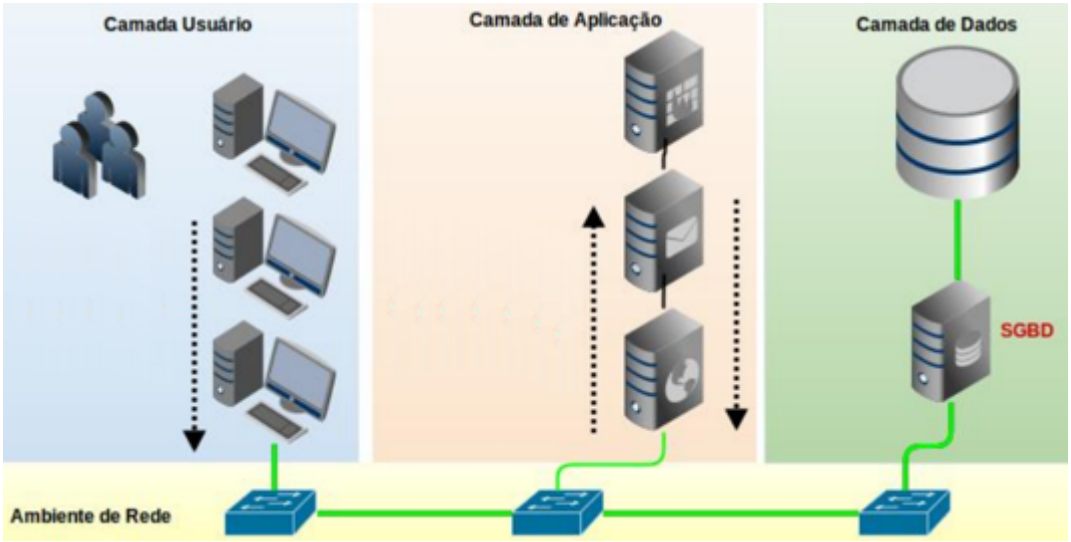
Figura 2 – Arquitetura Cliente-Servidor



Fonte: Desenvolvido pelo autor (2021), baseado em Lizama *et al.* (2016).

De acordo com Camões (2016), quando o cliente realiza uma conexão diretamente com o servidor, é chamada de aplicação em duas camadas, dessa forma, realiza o acesso diretamente no servidor. Muitas aplicações podem ser construídas com uma camada intermediária, que manipula a interação com o usuário e uma parte sobre o Banco de Dados, chamada de aplicação em três camadas. A Figura 3 representa uma arquitetura em três camadas.

Figura 3 – Arquitetura Cliente-Servidor em três camadas



Fonte: Camões (2016).

2.3 SISTEMA DISTRIBUÍDO

Segundo Monteiro *et al.* (2020), a definição de um sistema distribuído pode ser visto como um conjunto de computadores que estão conectados em uma rede, formando uma visão de um sistema único. Quando ocorre algo que não funciona como deveria ou alguma falha no sistema, é importante ressaltar que umas das principais características de um sistema distribuído é a transparência, ou seja, o operador não fica sabendo se ocorreu algum erro. Em contrapartida, o sistema de gerenciamento precisa notificar o administrador para que tome providencias para os dados serem entregues ao destino.

> Os sistemas de informação distribuídos têm como desafio lidar com a interoperabilidade, que é a característica que permite que um sistema se comunique com outro de forma transparente. Este tipo de sistema normalmente é acessado por meio de um navegador ou aplicativo que se comunica com um provedor. Além disso, diversos tipos de dispositivos podem se conectar à internet e acessar esse serviço. A interoperabilidade é cada vez mais necessária no meio corporativo (MONTEIRO *et al.*, 2020, p.30).

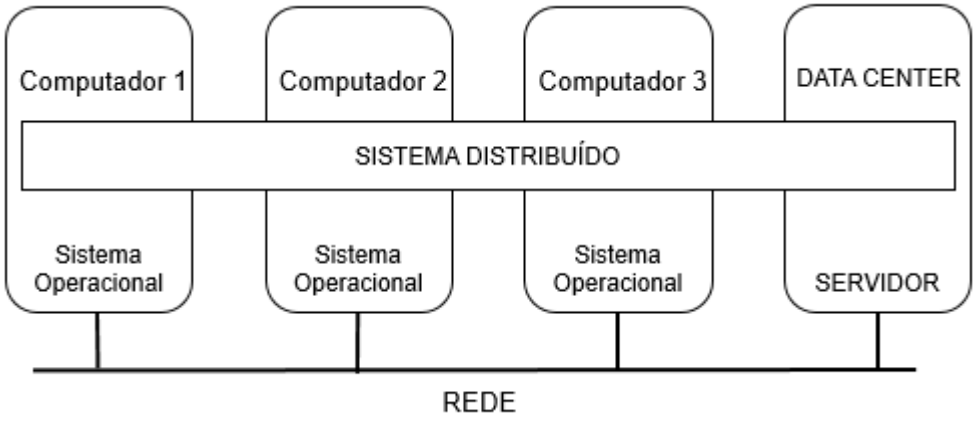
De acordo com Steen & Tanenbaum (2017), sistemas distribuídos fornecem uma visão de sistema único, neste contexto, os usuários finais não percebem que estão lidando com o fato de que os processos, dados e controles, estão dispersos entre uma rede interligada de computadores.

Além de ser um sistema transparente, para Lage & Alves (2015), é um sistema que permite processar dados de forma separada, em diferentes servidores, processando dados simultaneamente, proporcionando maior portabilidade, mobilidade e interatividade.

Outra observação importante: Sistemas distribuídos estão interligados entre vários nós independentes, cada um tem sua própria noção de tempo. Por razões de eficiência e redundância, relógios do mundo real são extremamente necessários para sincronizar dados entre si (STEEN; TANENBAUM, 2017).

A Figura 4, mostra a infraestrutura de um sistema distribuído, apresentando um conjunto computadores e servidor interligados na rede que demonstra um sistema único.

Figura 4 – Infraestrutura de um Sistema Distribuído



Fonte: Desenvolvido pelo autor (2021) baseado em Monteiro *et al.* (2020).

Para auxiliar no desenvolvimento de aplicativos distribuídos, os sistemas distribuídos são organizados para ter uma camada separada de software, o uso de *Middleware*, que será apresentada na próxima seção. Outras soluções para o desenvolvimento de aplicações distribuídas, também serão apresentadas nas seções seguintes.

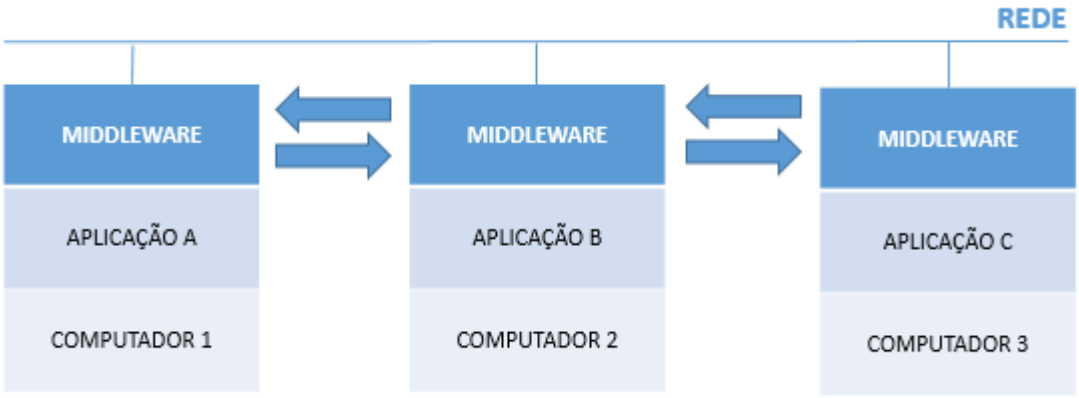
2.3.1 *Middleware*

Uma solução para a execução de aplicações distribuídas é a categoria de softwares *Middleware*, conhecido como Mediador, caracteriza uma camada de software que possibilita a interoperabilidade entre as aplicações distribuídas. Segundo Monteiro *et al.* (2020), utiliza recursos remotos para se conectar com aplicações, dados e usuários. Oferece serviços como controle de acesso, segurança, acesso ao banco de dados, instalações e manutenções remotas.

A camada é responsável pela funcionalidade do sistema e opera em modo Bidirecional. Comporta-se como uma interface entre a camada de sistemas e de aplicações (SCHENFELD *et al.*, 2016).

De acordo com Etzkon (2017), *Middleware* é usado como uma forma de reduzir a complexidade do sistema para os programadores que é um dos principais objetivos na utilização dessa camada intermediária. Os computadores podem ser de diferentes Sistemas Operacionais para a troca de mensagens. Foram originados através de *Socket*, no entanto tem grandes desvantagens em comparação com novas tecnologias de *Middleware* mais modernas, pois exigem conhecimentos em rede e também como armazenamento de informações são armazenados nos computadores. A Figura 5 demonstra como é o modelo de interação de um sistema *Middleware*.

Figura 5 – Modelo *Middleware*



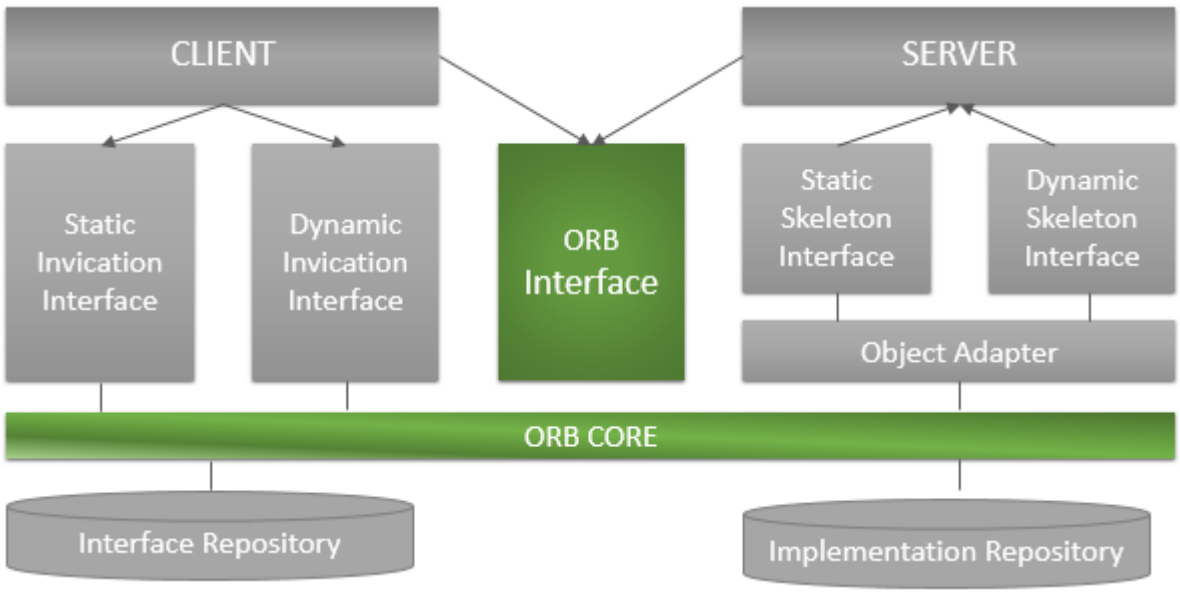
Fonte: Desenvolvido pelo autor (2021) baseado em Steen & Tanenbaum (2017).

2.3.2 ORB

É um tipo de *Middleware* que define um padrão de implementação de interatividade dos componentes de um sistema construído com base em objetos, segundo Mochalov, Bratchenko & Yakovlev (2018), corresponde a um barramento lógico chamado *Object Request Broker* (ORB). Gera diferentes mecanismos para executar e implementar como a seleção de componentes de programa para um aplicativo desenvolvido com base em objetos, para realizar a troca de dados e integração de recursos. Neste contexto, o processo do cliente solicita uma requisição e a ORB é encarregado a encontrar as informações do objeto para retornar os dados solicitados. A definição e implementação de ORBs teve bastante desenvolvimento durante o auge da utilização da arquitetura CORBA e outras arquiteturas e implementações fortemente baseadas em objetos.

A Figura 6, apresenta o modelo de arquitetura ORB, destacando as unidade lógicas ORB. A interface cliente invoca uma operação na implementação de um objeto, acessando os serviços remotos e deve ser transparente para o chamador. Quando um cliente invoca uma operação, o ORB é responsável por encontrar o objeto implementação, entregando a solicitação ao objeto e retornando qualquer resposta para o chamador. A Interface de Invocação Dinâmica (DII) permite que um cliente acesse diretamente os mecanismos de solicitação subjacentes fornecidos por um ORB. *Dynamic Skeleton Interface* (DSI) é o servidor analógico do lado ao DII do lado do cliente. O DSI permite um ORB para entregar solicitações para uma implementação de objeto que não tem conhecimento em tempo de compilação do tipo de objeto que está implementando (HUSEINI; MEMETI, 2019).

Figura 6 – Modelo de arquitetura ORB



Fonte: Desenvolvido pelo autor (2021) baseado em Akhter *et al.* (2021).

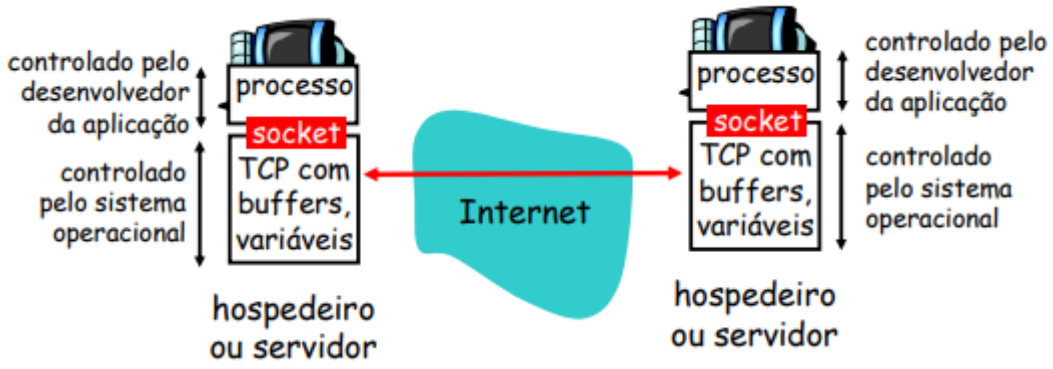
2.3.3 *Sockets*

De acordo com Steen & Tanenbaum (2017), muitos sistemas distribuídos são construídos através de um modelo simples orientado a mensagens, oferecida pela camada de transporte. É utilizado um conjunto de protocolos de mensagens por meio de *Socket*. Dessa forma, facilita a portabilidade de um aplicativo para uma máquina diferente. Para aplicativos maiores, esse tipo de troca de mensagens pode significar um esforço de implementação relativamente grande.

A programação de *Socket* com TCP, é um modelo orientado a mensagem, segundo do ponto de vista de Teixeira & Catanduva (2018), "TCP oferece transferência de *Bytes* confiável, em ordem (*Pipe*) entre cliente e servidor". Alguns fundamentos de funcionamento devem ocorrer, como por exemplo, o servidor precisa estar rodando antes que o cliente lhe solicite algo, cliente e servidor devem ter um *Socket* (porta) para receber e enviar segmentos e por fim, o cliente deve saber o endereço IP do servidor e o número da porta do *Socket*.

Os *Sockets* são a base de implementação de serviços de troca de mensagens mais evoluídos, como o próprio RMI, logo, a combinação de *Threads* e *Socket* é perfeita para implementação de um chat. A Figura 7 ilustra a comunicação TCP utilizando *Sockets*.

Figura 7 – *Socket* com serviço TCP

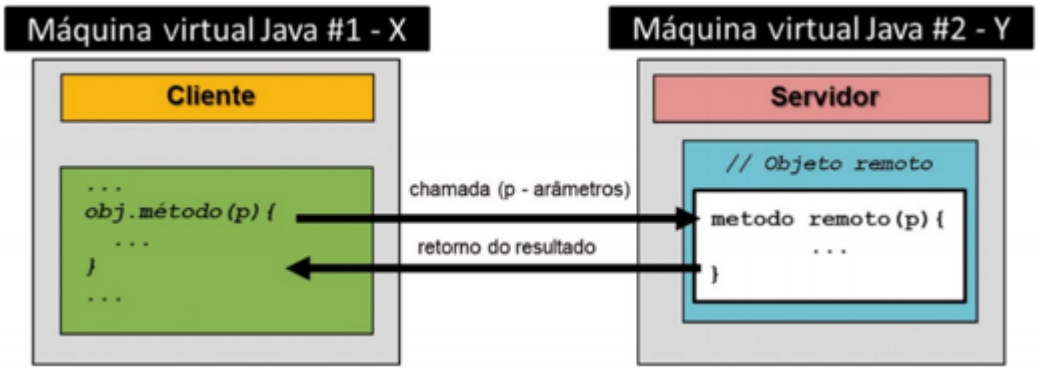


Fonte: Teixeira & Catanduva (2018).

2.3.4 *Java* RMI

Java RMI pode ser implementado em diferentes linguagens. Segundo Padole (2017), é baseada em uma solução *Java* que oferece mecanismos para o desenvolvimento de aplicações distribuídas, cliente e servidor. O servidor oferece os serviços e o cliente, por sua vez, utiliza esses serviços conforme a sua necessidade. De acordo com a funcionalidade do *Java* RMI, a chamada do método ocorre no endereço de memória que reside em um computador diferente, ou seja, referências de memória não compartilhadas. Esta comunicação remota entre as aplicações utilizando *Java* RMI, é possível devido a dois objetos *Stub* e *Skeleton*, que são responsáveis pela comunicação com a rede. O modelo *Java* RMI, é apresentado na Figura 8.

Figura 8 – Modelo *Java* RMI



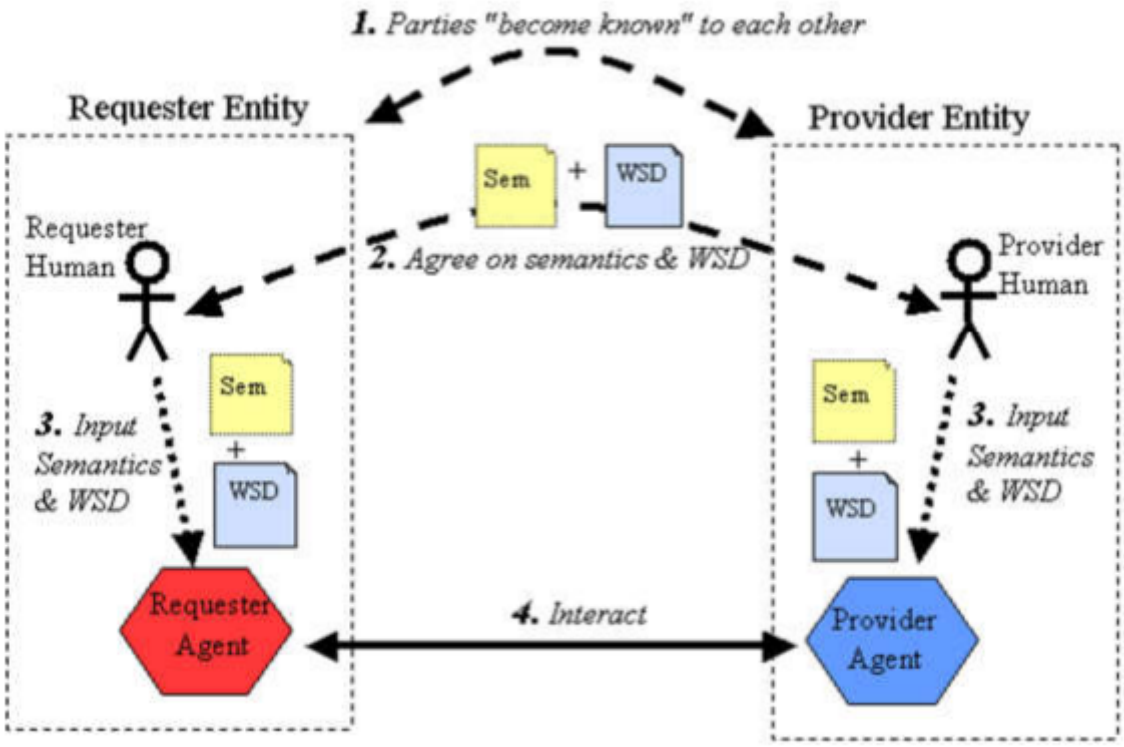
Fonte: Monteiro *et al.* (2020).

2.3.5 *Web Services*

Web Services são serviços fornecidos e consumidos através da *internet*. Segundo Junior (2017, p.4), " um serviço de software projetado para suportar a interação interoperável máquina-máquina sobre uma rede". Com o avanço da tecnologia e a da Internet, os sistemas precisam lidar com o problema da interoperabilidade via internet, que é a capacidade de transmitir dados

de forma transparente para outro sistema. Para ilustrar a arquitetura *Web Services*, a Figura 9, são ilustrados os principais passos e as responsabilidades.

Figura 9 – Arquitetura *Web Services*



Fonte: Junior (2017).

Elementos *Requester* e *Provider* fazem o reconhecimento, ambos, concordam com a descrição do processo e a semântica através da qual irão interagir, em seguida, são compreendidas pelos agentes *Requester* e *Provider*, e por fim, trocam utilizando um protocolo específico, realizando as tarefas solicitadas (JUNIOR, 2017).

Podemos citar uma forma de implementação de *Web Services* através dos princípios da arquitetura REST, apresentada na próxima seção.

2.3.6 REST

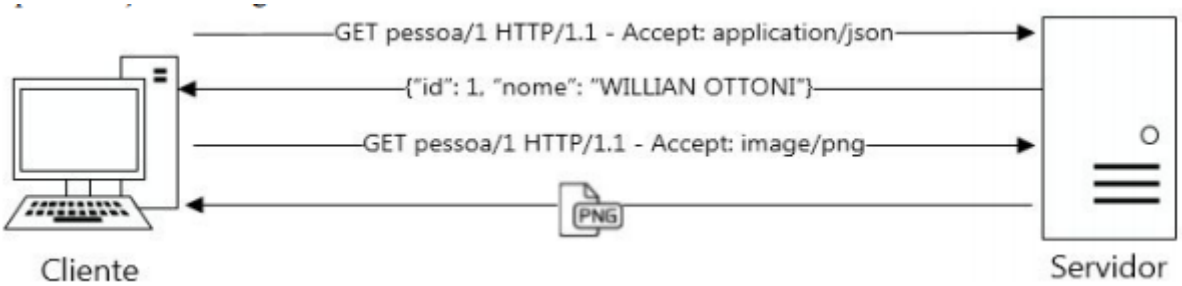
O surgimento da arquitetura *Representational State Transfer* (REST), para implementações de serviços *Web*, de acordo com Marques (2018), permitiu um modo mais simples e flexível para a comunicação entre cliente e servidor. O REST oferece uma coleção de comandos necessários para se desenvolver um serviço escalável, consistente e com alta performance.

De acordo com os desenvolvedores que utilizam REST, essa arquitetura leva como principal vantagem a facilidade no desenvolvimento, aproveitamento da infraestrutura *web* que existe hoje e com um esforço pequeno de aprendizado. É de fácil entendimento, porque o REST

usa o padrão de chamadas *GET*, *PUT*, *POST* e *DELETE* e a estrutura de retorno pode ser definida pelo desenvolvedor em vários formatos e pode utilizar qualquer navegador (FERREIRA; KNOP, 2017).

Serviços RESTful possuem comportamento REST, usa o mesmo design e seu comportamento muda de acordo com a maneira que é consumido (LECHETA, 2015). A Figura 10 mostra a utilização do recurso do *Hypertext Transfer Protocol* (HTTP) para enviar um comando *GET* para o servidor.

Figura 10 – Exemplo de comando *GET* de uma arquitetura REST



Fonte: Ferreira & Knop (2017).

2.4 BANCO DE DADOS

A maior parte dos projetos de sistemas necessitam de um banco de dados onde guardar as informações do sistema PDV para posteriormente serem replicadas. Existem vários bancos de dados com arquiteturas diferentes disponíveis no mercado, porém, alguns são comercializados e outros são gratuitos.

Através da Tabela 1, podemos analisar as características de alguns Sistemas de Gerenciamento de Banco de Dados (SGBD) disponíveis no mercado.

Tabela 1 – Principais SGBDs e suas características

SGBD	Plataforma	Tipo de licença	Arquitetura	Modelo
Cassandra	Multiplataforma	Gratuita	Distribuída	Família de colunas
Firebird	Multiplataforma	Gratuita	Cliente-Servidor	Relacional
IBM DB2	Multiplataforma	Paga	Cliente-Servidor	Relacional
Interbase	Multiplataforma	Paga	Cliente-Servidor	Relacional
MariaDB	Multiplataforma	Gratuita	Cliente-Servidor	Relacional
SQL Server	Multiplataforma	Paga e Gratuita	Cliente-Servidor	Relacional
MongoDB	Multiplataforma	Gratuita	Distribuída	Documentos
MySQL	Multiplataforma	Paga e Gratuita	Cliente-Servidor	Relacional
Oracle	Multiplataforma	Paga e Gratuita	Cliente-Servidor	Relacional
PostgreSQL	Multiplataforma	Gratuita	Cliente-Servidor	Objeto-Relacional
Redis	Multiplataforma	Gratuita	Distribuída	Chave-valor

Fonte: Pichetti (2020).

De acordo com Pichetti (2020) os SGBDs como *Oracle*, *MySQL*, *PostgreSQL* são os mais utilizados. Alguns SGBDs são de código aberto como *MySQL*, *PostgreSQL* e *Cassandra*. Os SGBDs como *IBM DB2* e *Interbase* são bancos de dados comerciais. O modelo de arquitetura relacional e arquitetura Cliente-Servidor esta presente em quase todos SGBDs, sendo que todos são multiplataforma.

Conforme citado por Steen & Tanenbaum (2017), em muitos ambientes Cliente-Servidor, a máquina cliente é uma estação de trabalho ou um PC do usuário conectado por meio de uma rede a um banco de dados. Grande parte da aplicação está executando na máquina cliente, mas por outro lado, todas as operações com arquivos ou entradas em um banco de dados, posteriormente vão para o servidor. Em uma arquitetura distribuída, todos os bancos de dados contém dados de todos os clientes.

Há alguns SGBDs que são também distribuídos, segundo Macák (2018) *MySQL* possui replicação entre *Clusters* e foi projetado para ser altamente disponível, permitindo fazer atualizações, upgrades de servidores e backups, sem tempo de inatividade. Já o *PostgreSQL* quando é integrado a aplicação *PgCluster*, executa *Cluster* e distribui dados de uma forma transparente.

Existem arquiteturas como mestre-mestre ou mestre-escravo, onde existem vários servidores que atendem como mestres (suportam atualizações) e também como escravos (suportam apenas consultas). Segundo Rockenbach *et al.* (2018), a replicação mestre-mestre mantém informações sincronizadas independentemente do servidor e com alto nível de acesso de leitura, pode ter baixa performance quando é replicação síncrona e a replicação mestre-escravo envia todas as alterações para todos os servidores escravos.

2.5 TRABALHOS RELACIONADOS

Essa seção apresenta alguns trabalhos correlatos com a pesquisa atual do trabalho, mostrando exemplos motivacionais de sistemas distribuídos e comunicações entre dispositivos que foram aplicadas, ilustrando seus objetivos e aplicações associadas a eles.

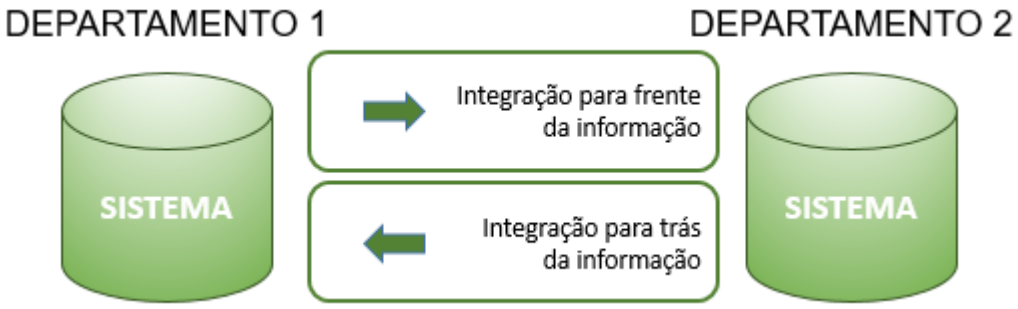
2.5.1 Ministério de Defesa do Reino Unido

Até a década de 1990, cada departamento do Ministério tinha seu próprio sistema com seu próprio banco de dados e as informações não eram replicadas, o compartilhamento entre sistemas era muito difícil. As informações eram introduzidas manualmente, múltiplas vezes, em diferentes sistemas. Isso consumia tempo e era ineficiente. Muitas vezes, o gerente não conseguia obter dados necessários para tomar decisões corretamente. O sistema foi desenvolvido pelo Exército e foram realizadas conexões entre bancos de dados pela rede. Essa integração permitiu transferir informações entre os bancos de dados, eliminando a necessidade de entrada de dados manuais no sistema, porque, depois que a informação já esta lá, o sistema envia imediatamente aos outros bancos de dados. Foi possível compartilhar informações entre diferentes

departamentos, mas também aumentou drasticamente a qualidade da informação (BALTZAN; PHILLIPS, 2012).

Construir integrações bidirecionais, oferece flexibilidade para criar, atualizar e deletar informações em qualquer um dos sistemas. De qualquer modo, implementar esse sistema gera mudanças no sistema e em todos outros sistemas. A Figura 11, apresenta o método de integração para frente e para trás.

Figura 11 – Método de integração Bidirecional



Fonte: Desenvolvido pelo autor (2021) baseado em Baltzan & Phillips (2012).

2.5.2 *Massively Multiplayer Online Games* (MMOGs)

A *Massively Multiplayer Online Games* (MMOGs) oferece jogos online com vários jogadores simultâneos, como o *EverQuest II*, da *Sony*, e o *EVE Online*, da empresa finlandesa *CCP Games*. O número de jogadores está aumentando cada vez mais e o sistema deve ser capaz de suportar mais de 50.000 usuários simultâneos (COULOURIS, 2014) .

Devido a necessidade de tempos de respostas mais rápidos, a propagação de eventos em tempo real para vários jogadores e a manutenção em um mundo compartilhado, representa um grande desafio para a engenharia MMOGs (COULOURIS, 2014) .

Para o projeto MMOGs, foram adotados 2 modelos. Especialmente para o jogo *EVE Online*, utiliza-se uma arquitetura Cliente-Servidor, onde a única cópia é mantida em um servidor centralizado e são acessadas por programas clientes, dos equipamentos dos jogadores como por exemplo, os consoles. Outros jogos MMOGs, adotaram arquiteturas mais distribuídas, onde é particionado um número muito grande de servidores e que estão geograficamente distribuídos, portanto o usuário se aloca em um servidor com base na proximidade geográfica, pois ocorrem latência da rede até o local do jogador (COULOURIS, 2014).

2.5.3 Um sistema de comunicação via *Socket*

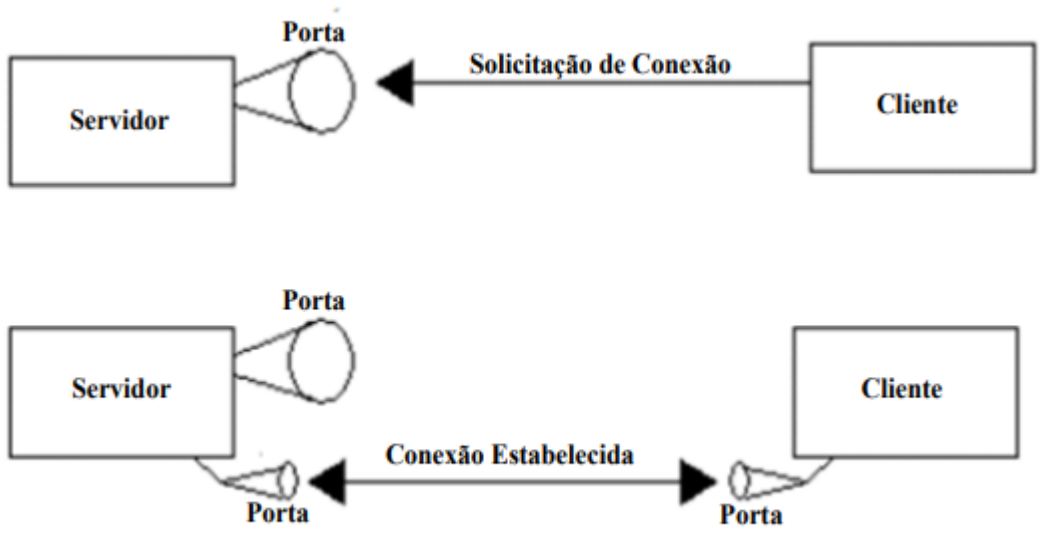
O fator principal desse trabalho relacionado, é enfatizar o uso de *Socket* para a comunicação e transferência de informações entre qualquer dispositivo. Utilizado em caso de neces-

sidade de comunicação entre dispositivos bem específicos, que não utilizam interfaces de mais alto nível.

Foi desenvolvido em um projeto acadêmico no Centro Federal de Educação Tecnológica do Rio de Janeiro (CEFET/RJ), um sistema de comunicação via *Socket* para controle de um robô de inspeção. A primeira camada de aplicação foi implementada na linguagem *Java* e permite operar e controlar o robô remotamente, através do protocolo TCP e *User Datagram Protocol* (UDP), juntamente com o protocolo *Internet Protocol* (IP). Já a segunda camada de aplicação foi realizada a comunicação via *Socket*, transmitindo um conjunto de instruções para os movimentos do robô, do servidor para o cliente (RIBEIRO *et al.*, 2017).

Ao mesmo tempo que um dos *Sockets* solicita a conexão com o servidor, o outro envia a a autorização para a conexão com o cliente, garantindo que a conexão seja estabelecida para que o robô seja controlado. Esse esquema de funcionamento é apresentado na Figura 12 e representa o modo de comunicação por TCP, utilizando *Sockets* (RIBEIRO *et al.*, 2017).

Figura 12 – Comunicação via *Socket* TCP/IP



Fonte: Ribeiro *et al.* (2017).

2.6 CONSIDERAÇÕES DO CAPÍTULO

Implementar um sistema distribuído é um objetivo importante para tornar mais fácil aos usuários e aplicativos, acessar dados e compartilhar recursos remotos, como por exemplo, o servidor. Apesar da replicação aumentar o processamento de operações de leitura e escrita, aumenta também a disponibilidade de dados, por haver mais de uma alternativa para acessar os dados. De acordo a demanda do negócio, a replicação é classificada como síncrona ou assíncrona. Neste contexto, foram citadas as tecnologias para mover ou buscar dados, tratando os principais conceitos sobre *Middleware*, *software* que possibilita a troca informações entre aplicações distribuídas. ORB, facilita a comunicação entre sistemas através de objetos. *Sockets*, protocolo orientado a mensagens, *Java* RMI oferecem mecanismos para o desenvolvimento de aplicações distribuídas. *Web Services*, tecnologia mais usada atualmente na *Web* para a comunicação de dados entre sistemas. REST, utiliza um modo de desenvolvimento mais simples para a comunicação entre cliente e servidor via *Web Services*.

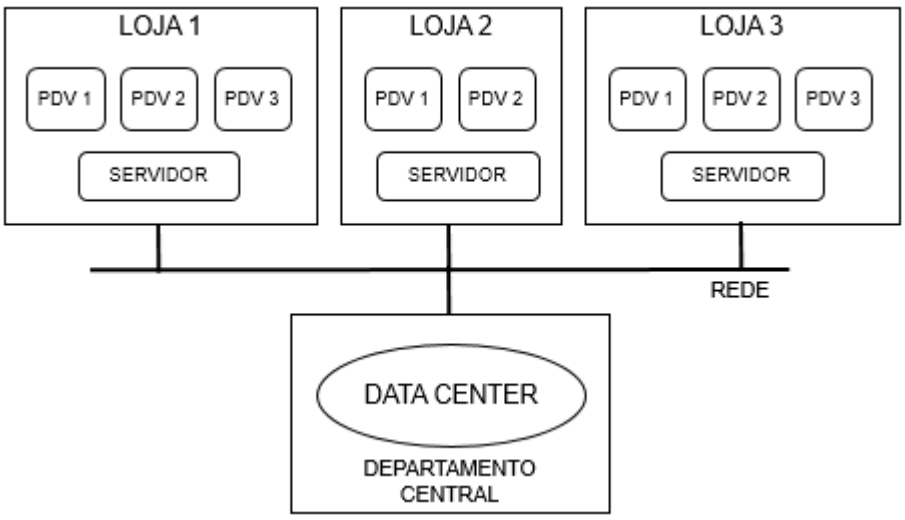
No contexto dos trabalhos relacionados, o sistema do Ministério de Defesa do Reino Unido basicamente é uma arquitetura Bidirecional, mantendo dados sincronizados em todos SGBDs. Essa funcionalidade é relevante neste trabalho para sincronizar informações entre os sistemas PDVs, localizados na mesma loja. Já a arquitetura Cliente-Servidor, adotado no primeiro modelo do projeto MMOGs, mantém dados de todos os clientes em um servidor centralizado, como por exemplo o *Data Center* da empresa *STORE*. O sistema de comunicação via *Socket*, utilizada no projeto acadêmico para controlar um robô remotamente, é responsável pela transferência de informações entre os equipamentos. Portanto os trabalhos relacionados citados, empregam as tecnologias que são utilizadas no desenvolvimento da aplicação.

No próximo capítulo serão abordado os detalhes da organização atual da empresa *STORE* para aplicar o conhecimento da situação atual e posteriormente, apresentar a aplicação desenvolvida para este trabalho.

3 ORGANIZAÇÃO ATUAL

A empresa *STORE* possui um departamento central, onde dispõe um Data Center da *International Business Machines Corporation* (IBM) *Informix*, que gerencia as informações de todas as lojas. Cada loja possui um servidor também da IBM *Informix*, para armazenar os dados que são processados pelos sistemas PDVs. A comunicação entre o departamento central e as lojas é realizada através de links dedicados de alta velocidade. A Figura 13 demonstra a organização atual da empresa.

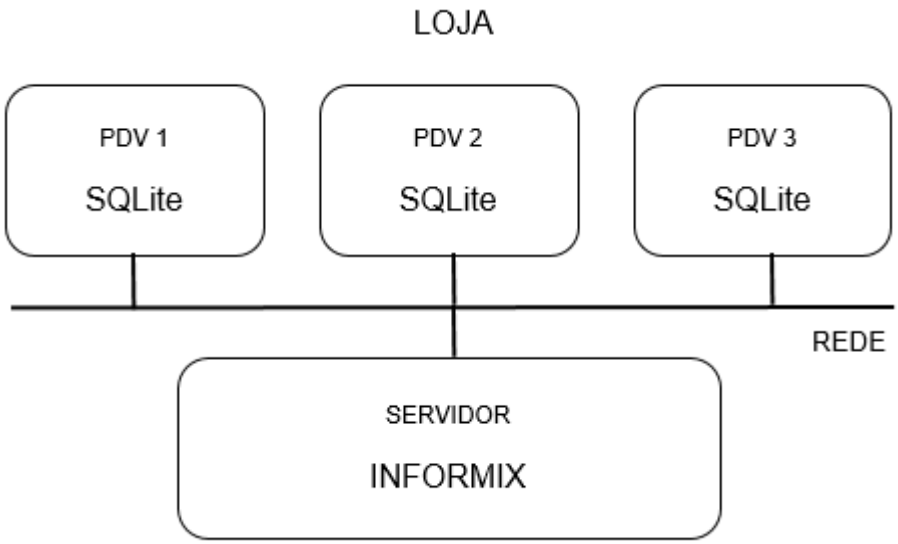
Figura 13 – Organização atual



Fonte: Desenvolvido pelo autor (2021).

O sistema PDV é integrado com banco de dados *SQLite*. Caso houver alguma parada crítica ou manutenção do servidor interno, é possível utilizar o sistema sem conexão com o servidor, evitando assim a interrupção do serviço de geração da Nota Fiscal de Consumidor Eletrônica (NFC). Após o servidor se estabilizar, é realizada a sincronização dos dados. Atualmente o sistema é desenvolvido em uma linguagem de programação *Delphi*. Figura 14 apresenta a infraestrutura interna de uma loja.

Figura 14 – Sistema PDV



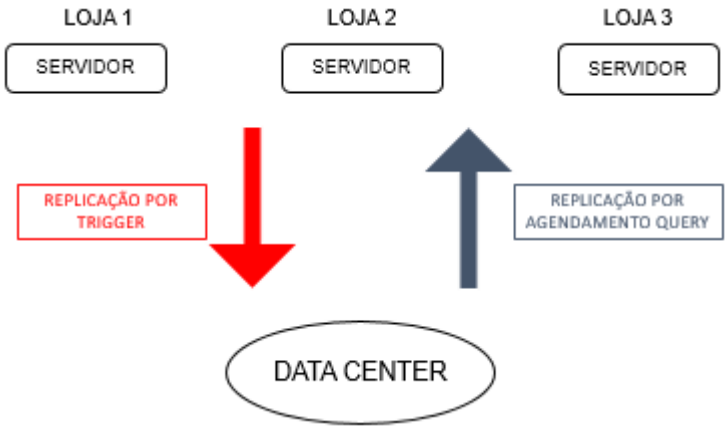
Fonte: Desenvolvido pelo autor (2021).

Há duas maneiras distintas para a replicação de dados, por *Triggers* ou por agendamento de *Query*. As informações das lojas são replicadas via *Trigger* para o *Data Center* que podem ser chamadas também como gatilhos ou disparadores. A *Trigger* é um tipo de procedimento que é executado sempre quando há uma tentativa de modificar os dados na tabela, é executado automaticamente assim que é disparada, não podendo ser ignorado. Um dos principais problemas de se utilizar *Triggers* é quando ocorre alguma falha de conexão ou indisponibilidade de rede, os dados não são sincronizados. Em cada tabela deve ser configurada as *Triggers*, caso a tabela não tiver os dados não são replicados.

Por outro lado, na replicação por agendamento de *Query*, as operações de inclusão, alteração e exclusão no banco de dados geram uma *Query* a qual em seguida é enviada para fila de replicações que devem ser executadas em todos os servidores interligados na rede. A fila serve para executar as *Query* no banco de dados na ordem que vão chegando, portanto se alguma *Query* ocorrer algum problema na atualização, não é mais atualizada até que o administrador do sistema realiza o ajuste para liberar a fila de replicações.

A Figura 15 demonstra como é realizada a sincronização dos dados entre o *Data Center* e os servidores das lojas.

Figura 15 – Sincronização dos dados



Fonte: Desenvolvido pelo autor (2021).

3.1 CONSIDERAÇÕES DO CAPÍTULO

De acordo com a empresa *STORE*, a replicação de dados ainda não esta totalmente consistente, ainda faltam melhorias a serem desenvolvidas para a replicação manter integridade dos dados.

No próximo capítulo, será demostrado o desenvolvimento da aplicação que é o sistema de gerenciamento de replicação, os requisitos necessários para a implantação, a interface, a arquitetura de *software*, o diagrama de classes e o modelo de dados que integram todo o sistema de gerenciamento de replicação.

4 APLICAÇÃO

O problema se originou devido ao crescimento do número de lojas, a empresa *STORE* compreendeu a importância de centralizar os dados de sua organização para a administração central. Desse forma, é possível realizar consultas e atualizações dos dados com mais agilidade. Outro problema é manter os dados consistentes, constantemente é necessário verificar se os dados estão sincronizados, gerando retrabalho e costuma ser desgastante em termos de recursos.

Usando os conceitos em replicações de dados, o propósito deste trabalho é desenvolver um sistema de gerenciamento de replicação no qual são replicados dados de um sistema PDV, com capacidade de verificar se os dados estão consistentes.

Este capítulo apresenta a aplicação, explicando as principais características e suas funcionalidades do sistema de gerenciamento de replicação através de requisitos da aplicação, interface, arquitetura de *software*, diagrama de classes e modelo de dados.

4.1 REQUISITOS DA APLICAÇÃO

Para iniciar os requisitos do sistema de gerenciamento de replicação, é necessário substituir o banco de dados *SQLite*, utilizado somente em ambientes monousuário. Desse modo, somente um usuário pode estar logado no sistema. Portanto, para outros sistemas PDVs compartilharem dados entre si, o modelo proposto nesse trabalho é utilizar um banco de dados que permite acesso em ambientes multiusuários.

O SGBD selecionado para este trabalho é o *PostgreSQL*. De acordo com Carvalho (2017) é um forte sistema gerenciador de banco de dados objeto-relacional de código aberto e gratuito. Tem fácil integração com *Delphi* que é utilizado no desenvolvimento do sistema PDV atual. É um SGBD que tem capacidade de replicação e é distribuído, porém o SGBD do *Data Center* continuará sendo o *Informix*. Portanto, o sistema de gerenciamento de replicação pode lidar com qualquer SGBD. Não foram utilizados os recursos distribuídos do *PostgreSQL* devido a diferença de arquitetura *Informix* que não é compatível, em vista disso, serão utilizados *Sockets*.

Cada sistema PDV contém seu próprio banco de dados com capacidade de replicação. As consultas e as atualizações de dados são realizadas diretamente no próprio banco de dados do sistema PDV. Os computadores internos da loja devem estar interligados na rede local e na Rede Virtual Privada (VPN) para a conexão com o servidor central.

A replicação de dados ocorre após a emissão de uma nota fiscal concluída com sucesso no sistema PDV. Deste modo, os dados são replicados entre os sistemas PDVs da loja e para o servidor central, principalmente, para atualizar as informações do estoque do produto e do

faturamento da loja.

Do outro lado, qualquer alteração no banco de dados do servidor central realizada pela administração, por exemplo, entrada de novos produtos no estoque, atualizações no cadastro do produto e atualizações de tabelas, são replicados para os sistemas PDVs das lojas.

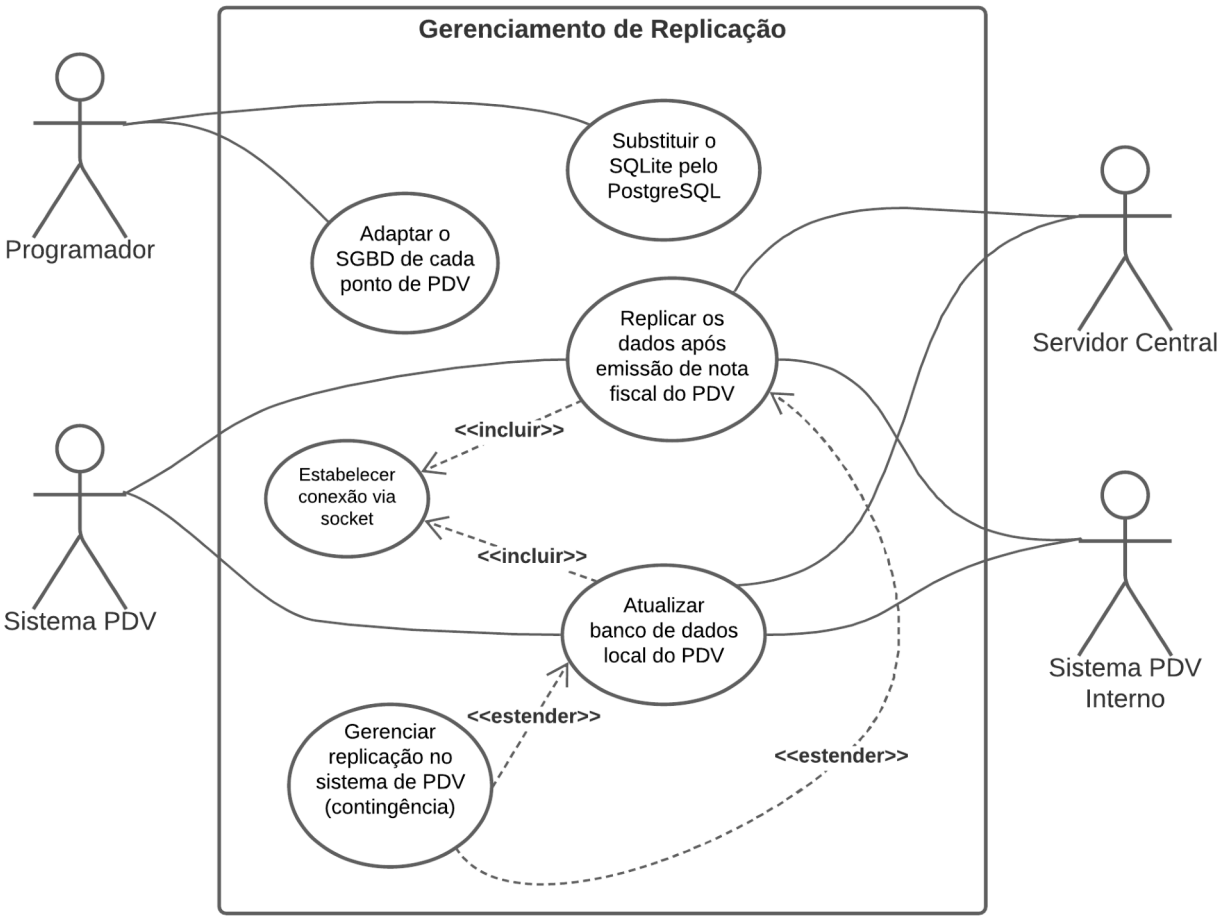
Todo o gerenciamento de replicação de dados é realizada através de duas *Threads*, cliente e servidor. Ambas são executadas em *background* no sistema PDV. Caso o computador esteja desligado por indeterminado tempo, quando for realigado o sistema de gerenciamento de replicação entende qual foi a sua última atualização que foi executada e quais são as atualizações de outros sistemas de gerenciamento de replicação que devem executadas para manter os dados sincronizados e consistentes entre todos os sistemas PDVs da loja e com o servidor central.

As *Threads* executam processos em paralelos sem interromper o processo de venda no sistema PDV. De certa forma, consome mais processamento, portanto, os requisitos mínimos de *hardware* da máquina devem ser reavaliados. A comunicação entre os sistemas de gerenciamento de replicação é via *Sockets*.

É possível também que ocorra uma falha de comunicação, o *link* da internet pode ficar indisponível na rede interna da loja ou com comunicação VPN do servidor central. Portanto, o sistema PDV é capaz de verificar se existe a comunicação para estabelecer uma conexão, considerando que o sistema PDV não pode paralisar, independente da conexão, pois o sistema está preparado para emitir notas em contingência para serem enviadas posteriormente à Sefaz assim que estabelecer a conexão com a internet.

Para formalizar as principais operações do sistema de gerenciamento de replicação, são apresentadas no diagrama de caso de uso da Figura 16.

Figura 16 – Diagrama de caso de uso do Gerenciamento de Replicação

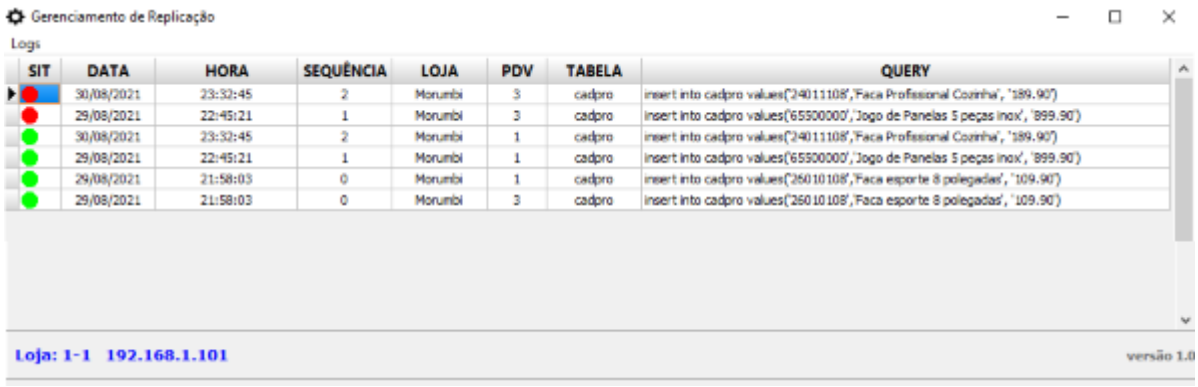


Fonte: Desenvolvido pelo autor (2021).

4.2 INTERFACE

O gerenciamento de replicação é discreto para o usuário, dessa forma, não fica sabendo qual operação o sistema PDV esta processando no momento, somente o administrador ou operador do sistema tem acesso às informações do gerenciador de replicação. A interface contém todas informações da tabela de replicações *movrep*, os atributos da tabela são data, hora, sequência de gravação, loja, número do PDV, tabela que deve ser atualizada e por último a *Query* que contém os dados para serem executadas nas tabelas. Na barra inferior da tela apresenta as informações do código da loja, número do PDV e o IP da máquina. A Figura 17, mostra a interface do gerenciamento de replicação.

Figura 17 – Interface do Gerenciamento de Replicação



Fonte: Desenvolvido pelo autor (2021).

Para um entendimento das principais informações relevantes do sistema de gerenciamento de replicação, seguindo a ordem das colunas da esquerda para a direita, são elas:

1. A situação é apresentada com um indicador na cor verde que indica que o registro foi atualizado e o indicador na cor vermelha que indica que o registro esta pendente para ser atualizado no banco de dados.
2. Data do registro gravado no movimento de replicação.
3. Hora que foi registrado o registro.
4. Sequência do registro, caso houver duplicações com mesma data e hora, a mesma é incrementada, serve para indicar a ordem que o registro deve ser atualizada no banco de dados.
5. Indica de qual loja o registro pertence.
6. Número do PDV de uma determinada loja.
7. Tabela que deve ser atualizada no banco de dados.
8. A *Query* contém dados que permite adicionar, remover e modificar qualquer tipo de dado de um banco de dados.

4.3 ARQUITETURA DE SOFTWARE

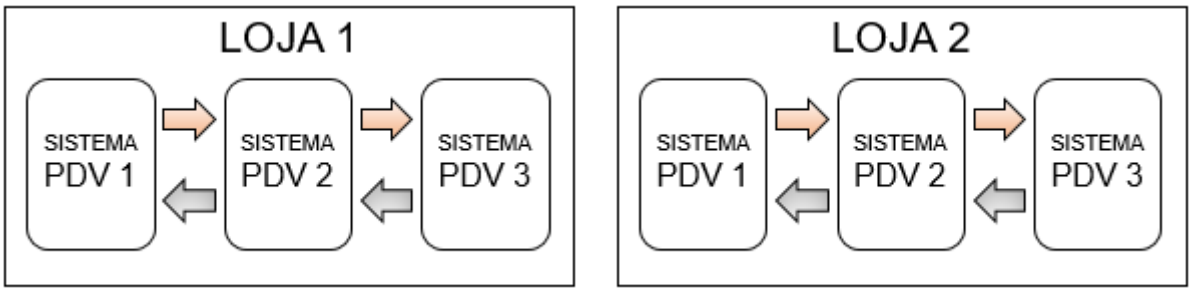
O sistema de PDV utiliza uma replicação baseada em dois modelos: Bidirecional e Cliente-Servidor. É considerada Bidirecional quando a replicação ocorre de um sistema PDV para outro, apresentada na Figura 18. Dessa forma, os sistemas se comunicam entre si.

Figura 18 – Arquitetura Bidirecional

Fonte: Desenvolvido pelo autor (2021).

Porém, a replicação Bidirecional via *Sockets* somente ocorre para os sistemas PDVs instalados em uma única loja, conectadas na mesma rede local conforme representado na Figura 19. Dessa forma, é possível gerenciar as lojas de maneira independente, cada uma com sua regra de negócio.

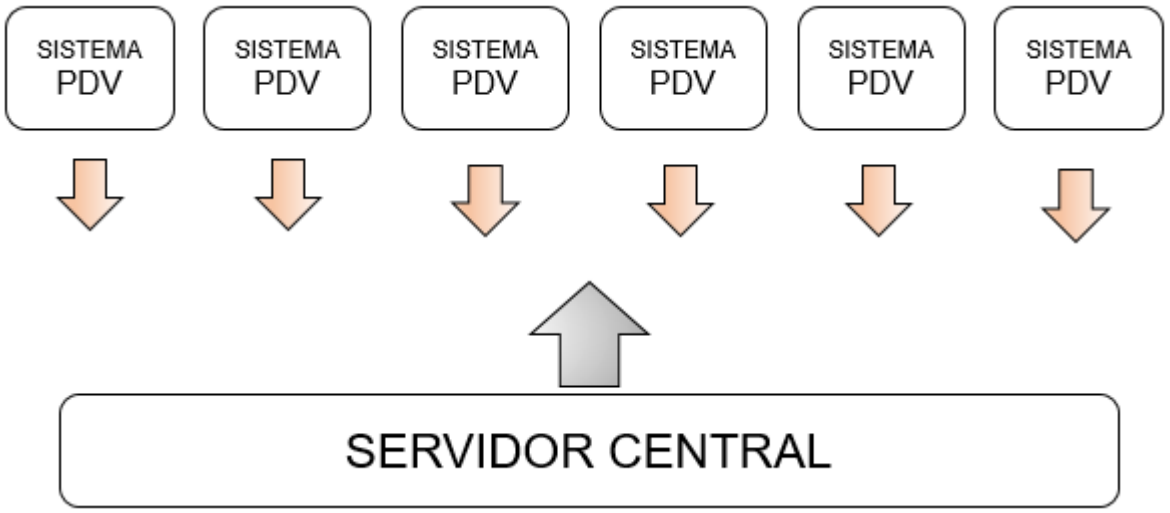
Figura 19 – Bidirecional interna na loja



Fonte: Desenvolvido pelo autor (2021).

Já a comunicação com o servidor central via *Sockets* consiste no modelo Cliente-Servidor, onde os sistemas PDVs são considerados como clientes. Desta forma, o servidor realiza a replicação dos dados para todos os clientes e por outro lado, obtém dados dos clientes para centralizar dados dos sistemas PDVs. A conexão é realizada através de um *link* dedicado de alta velocidade, conectado entre o servidor e as lojas. Essa arquitetura é apresentada na Figura 20.

Figura 20 – Arquitetura Cliente-Servidor



Fonte: Desenvolvido pelo autor (2021).

Os processos do sistema de gerenciamento de replicação são classificadas em instruções numeradas e separadas em duas partes, cliente e servidor. O processo cliente somente solicita dados para o servidor e as instruções são detalhadas:

1. Iniciar o processo de atualizar o servidor com dados do cliente, solicitar para o servidor qual foi a última atualização do cliente que solicitou.
2. Ler a instrução recebida do servidor, comparar o último registro atualizado no cliente e enviar para o servidor todos os registros inseridos posteriormente.
3. Iniciar o processo de atualizar o cliente com dados do servidor, listar todos os PDVs cadastrados na loja e mais o servidor centralizado, exceto o PDV atual, enviar para o servidor o último registro atualizado de cada um dos sistemas.
4. Receber a instrução para incluir no movimento de replicação do cliente a atualização de tabela.

Por outro lado, o processo servidor é responsável apenas para receber as instrução, fazer a leitura, processar e retornar os dados solicitados para o cliente. As instruções são detalhadas:

1. Responsável para retornar a última atualização do cliente que solicitou.
2. Receber a instrução para incluir no movimento de replicação do servidor, a atualização de tabela.
3. Ler a instrução recebida do cliente, comparar o último registro atualizado e enviar todos os registros inseridos posteriormente.

4.4 DIAGRAMA DE CLASSES

No diagrama de classes apresentada na Figura 21, as *Threads* são apresentadas nas classes ThreadServidor e ThreadCliente. O servidor central herda os processos através da classe ThreadServidor e todos os sistemas PDVs seguem a rotina de processamento com ambas as classes ThreadCliente e ThreadServidor. Como a replicação é assíncrona, as *Threads* são configuradas para executar frequentemente de tempos em tempos.

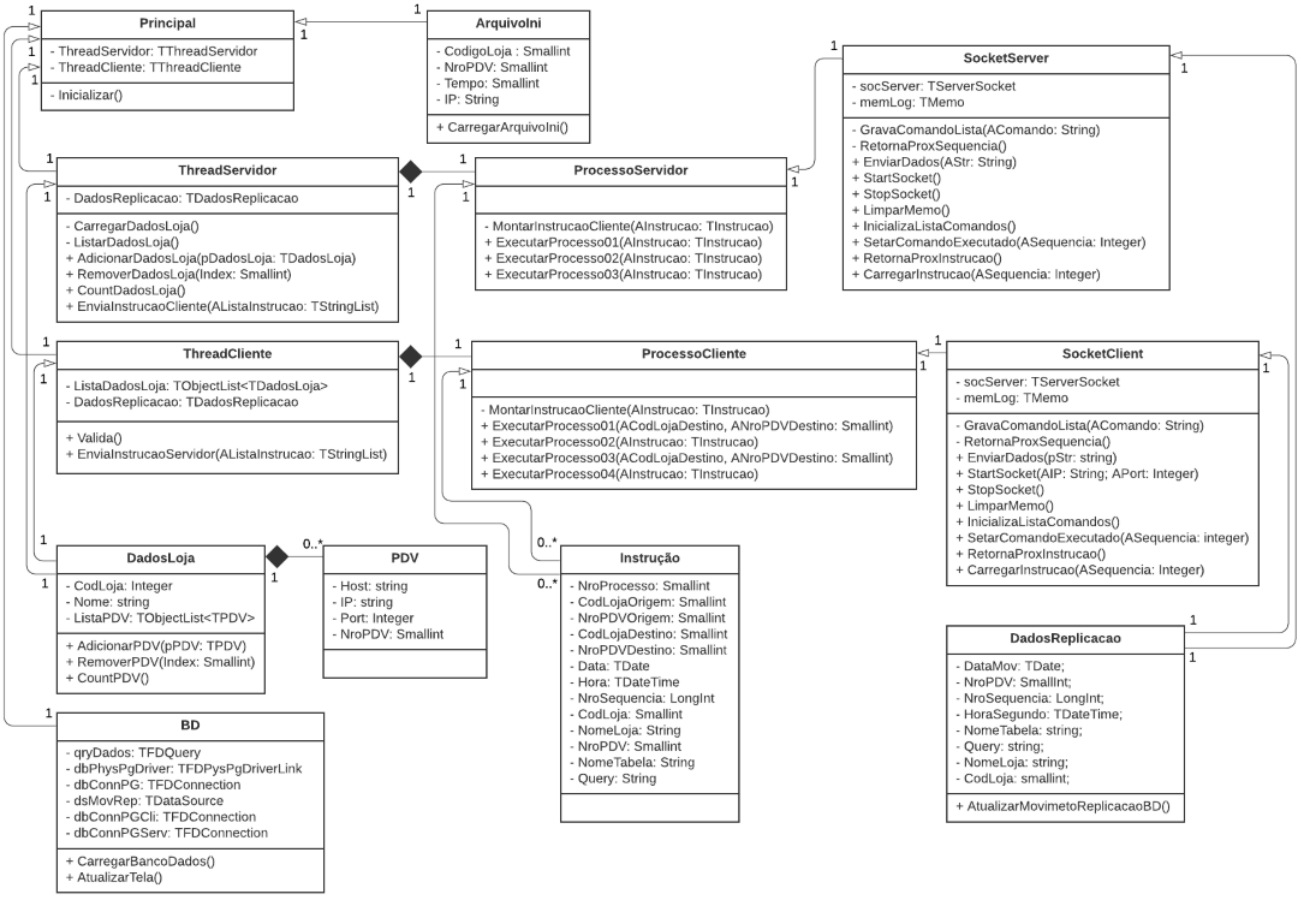
A classe Principal é responsável para dar partida ao sistema de gerenciamento de replicação, onde a classe ArquivoIni faz a leitura de um arquivo de configuração de inicialização com os dados pré configurados pelo próprio administrador do sistema.

As *Threads* servidor e cliente recebem a classe DadosLoja que contém informações de acesso dos sistemas PDVs necessário para a comunicação via *Sockets*. As instruções são processadas pelas classes de processamento das *Threads*, portanto utilizam a classe Instrução para montar a estrutura dos dados e pode conter várias instruções. Todos os processos administradas pelas *Threads* são gerenciadas nas classes ProcessoServidor e ProcessoCliente.

A comunicação entre outros sistemas fazem o uso das classes SocketServer e SocketClient, enviam e recebem dados de replicação utilizando a classe DadosReplicacao. Os dados são gravados em uma tabela, formando uma única fila de instruções e em seguida são processadas pelos processos do Cliente e Servidor.

A classe BD contém informações de acesso ao banco de dados *PostgreSQL* para inserir, atualizar e deletar dados. Também faz a atualização das informações na interface do gerenciamento de replicação. A Figura 21 apresenta o diagrama de classes.

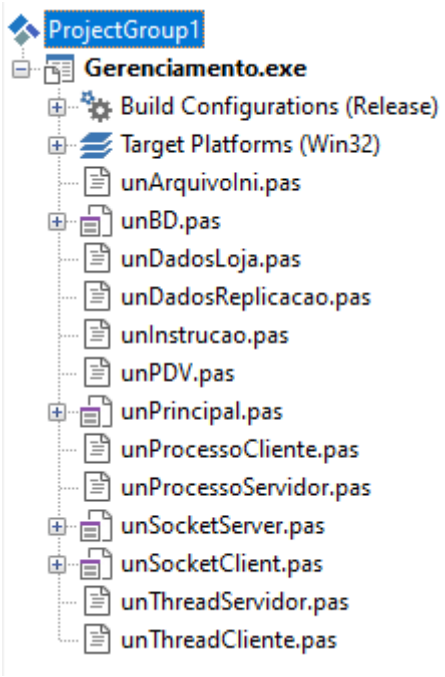
Figura 21 – Diagrama de Classes



Fonte: Desenvolvido pelo autor (2021).

No projeto de desenvolvimento do sistema de gerenciamento de replicação seguem as mesmas informações do diagrama de classes. A Figura 22 mostra as classes utilizadas dentro da *Integrated Development Environment* (IDE) da linguagem *Delphi*.

Figura 22 – Classes utilizadas no projeto



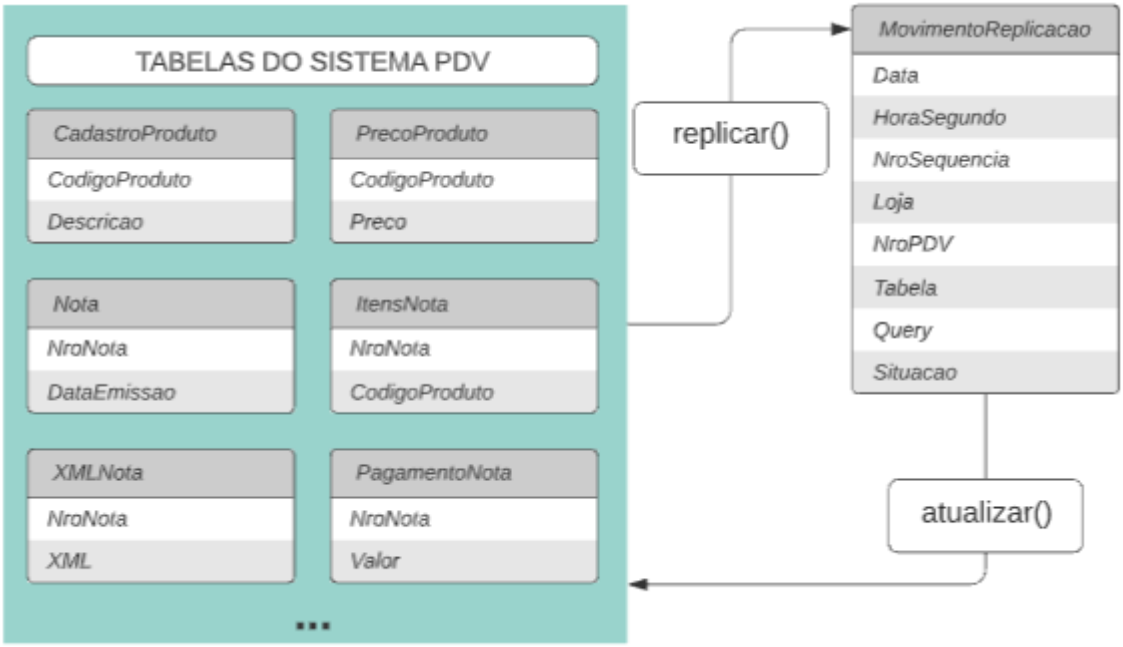
Fonte: Desenvolvido pelo autor (2021).

4.5 MODELO DE DADOS

O modelo de banco de dados elaborado para o gerenciamento de replicações, é mantido em uma única tabela. Qualquer alteração realizada nas tabelas que gerenciam o sistema PDV, é chamada a função Replicar para gravar um registro na tabela de MovimentoReplicacao. As replicações de outros sistemas PDVs e do servidor central são inseridas na mesma tabela. Em seguida a função Atualizar aplica as alterações no banco de dados do próprio sistema PDV. Tanto as funções Replicar e Atualizar são processadas na classe DadosReplicacao apresentada na seção anterior no diagrama de classes.

As tabelas do sistema PDV apresentada na Figura 23 são algumas das principais tabelas que atualizam, porém não fazem parte no processo de replicação, somente serve como exemplo para demonstrar o processo de replicar dados de uma tabela. Os atributos da tabela MovimentoReplicacao são as mesmas que foram detalhadas na interface da Figura 17.

Figura 23 – Diagrama de Banco de Dados do sistema PDV



Fonte: Desenvolvido pelo autor (2021).

4.6 CONSIDERAÇÕES DO CAPÍTULO

A tecnologia que temos a disposição atualmente traz para a empresa uma série de possibilidades de obter novas ferramentas, gerando aumento da produtividade, viabilizando o crescimento da empresa ao longo dos anos e se beneficiando em seus resultados financeiros. Afinal, a empresa *STORE* exige um processo de gestão mais ágil e inteligente, garantindo que os dados sejam consistentes para a tomada de decisão tornar-se mais assertiva.

O aumento de dados gerados pelos sistemas PDVs devido a expansão das lojas da empresa *STORE*, verificou-se a necessidade centralizar os dados para obter o acesso às informações, inicialmente não havia a possibilidade de crescimento acelerado no decorrer dos anos.

No entanto, o presente estudo utilizou conceitos de replicações de dados para garantir a confiabilidade e disponibilidade do sistema a qualquer momento, mesmo quando o sistema permanece por várias horas ou muitos dias sem conexão com a internet. Além disso, manter dados centralizados no servidor central serve também para *backup* dos dados, reduzindo riscos de perda dos dados. O principal diferencial deste sistema de gerenciamento de replicação utiliza duas arquiteturas para o gerenciamento de replicação, Bidirecional e Cliente-Servidor.

Verificou-se que através dessa metodologia, pressupõe-se que é possível obter uma redução de custos em razão de não utilizar o servidor local que esta hospedado na loja para o gerenciamento dos sistemas PDVs. Notou-se que as replicações ocorrem simultaneamente entre os sistemas PDV, devido a independência entre eles na replicação dos dados. Atualmente existe uma única fila de replicação para todos os sistemas PDVs.

5 TESTE DA APLICAÇÃO

Nesta fase iniciou-se o desenvolvimento do gerenciamento de replicação. As funcionalidades do programa foram testados em projetos separados e depois adicionado ao projeto de gerenciamento de replicação. Os primeiros testes ocorreram sem muito sucesso. Para isso, necessitou-se algumas mudanças na estruturação do código, no diagrama de classes e foram separadas algumas funcionalidades do processo cliente e servidor, bem como, a adição de mais uma *Thread* para executar os processos em paralelo e os acessos simultâneos ao banco de dados devido a execução dos processos em paralelo que necessitam buscar informações. No início da solução, foi pensada em apenas uma *Thread*.

Algumas funcionalidades foram criadas para facilitar o gerenciamento, tais como, o arquivo de inicialização, contém a pré-configuração para iniciar o programa e identifica qual é o sistema PDV que vai ser executado. É possível também visualizar os *Logs* das mensagens enviadas e recebidas que é relevante para identificar os erros e os dados que estão trafegando nos *Socket* servidor e cliente.

5.1 PRÉ-CONFIGURAÇÃO

A inicialização do sistema de gerenciamento de replicação se inicia a partir de um arquivo de configurações iniciais. Contém dados como o IP da máquina local, tempo de repetição do processo, código da loja e o número do PDV. O tempo de repetição do processo controla a execução das *Threads*. Para um ambiente de teste controlado foi configurado para repetir o processo a cada 5 segundos, justamente para a *Threads* não ficar consumindo muito recurso de processamento, podendo ser alterada conforme a capacidade de cada máquina. Também é necessário configurar o IP, código da loja e o número do PDV do computador local, dessa forma o sistema entende qual é o sistema de gerenciamento de replicação que esta executando. Essas configurações iniciais são necessárias configurar durante a implantação do sistema PDV na loja. Para o banco de dados *PostgreSQL*, foi instalado e configurado o *pgAdmin 4* versão 5.2, para uso em Desktop. A Figura 24 apresenta o arquivo de configurações de inicialização.

Figura 24 – Arquivo de configurações de inicialização

```
############################################################
## Configurações para o sistema PDV atual do computador ##
############################################################

[configIni]
#IP da máquina local
ip=127.0.0.1

# Tempo de repetição do processo
tempo=5000

[infoPDV]
# OBS: Matriz é 0 e nro PDV 0

# Código da loja
codigoloja=1

# Número do PDV
numero=1
```

Fonte: Desenvolvido pelo autor (2021).

Além das configurações iniciais, é necessário criar todas as tabelas que fazem parte da operação do sistema PDV e também do gerenciamento de replicação. Os dados necessários para carregar nas tabelas são os dados dos servidores para realizar o acesso e as informações do sistema PDV. Essas informações são essenciais para realizar a conexão com outros sistemas de gerenciamento de replicação e com o servidor central. A Figura 25 apresenta a tabela dados da loja que contém o código e o nome da loja.

Figura 25 – Tabela dados da loja

	codloja [PK] smallint	nomeloja character (30)
1	1	Morumbi
2	0	Matriz

Fonte: Desenvolvido pelo autor (2021).

Para completar os dados de acessos são gravados na tabela informações do sistema PDV apresentada na Figura 26. Contém atributos como o código da loja, número do caixa, endereço *host* e IP do servidor.

Figura 26 – Tabela informações do sistema PDV

	codloja [PK] smallint	nropdv [PK] smallint	host character (15)	ip character (15)
1	1	2	localhost	127.0.0.1
2	1	3	localhost	127.0.0.1
3	0	0	DESKTOP-706OJMG	192.168.1.109
4	1	1	DESKTOP-ROILNGQ	192.168.1.101

Fonte: Desenvolvido pelo autor (2021).

5.2 INSTRUÇÃO

A comunicação entre os sistemas de gerenciamento de replicação é baseada em instruções. Os processos que geram as instruções já foram detalhadas na seção da Arquitetura de Software. A instrução é montada seguindo a ordem dos atributos:

1. Número do Processo
2. Origem - Código da Loja
3. Origem - Número do PDV
4. Destino - Código da Loja
5. Destino - Número do PDV
6. Data do Movimento
7. Hora do Movimento
8. Sequência
9. Número da Loja
10. Número do PDV
11. Nome da Tabela
12. Query

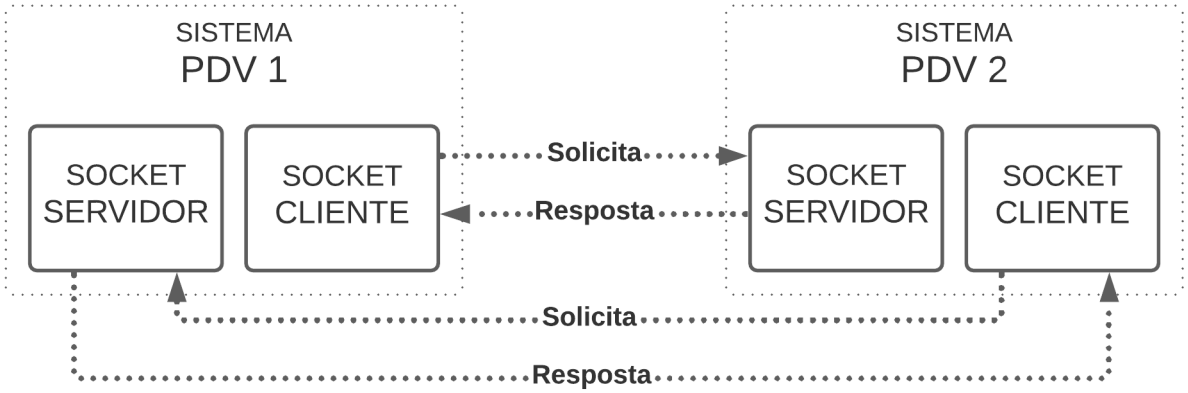
Basicamente a instrução contém informações de qual é o processo deve ser executado, qual é a origem e o destino do sistema de gerenciamento de replicação e os dados da tabela de movimento de replicação que deve ser atualizado. Um exemplo, convertendo esses dados em uma única instrução, seguindo a ordem das colunas, a instrução ficaria assim:

- 3|1|1|0|0|12/10/2021|10:20:12|1|1|1| cadpro | insert into cadpro values(746,'Faca')

5.3 PROGRESSO

Durante o desenvolvimento da aplicação foram encontrados alguns obstáculos. O *Socket* cliente somente se comunica através de um *Socket* servidor, portanto, para fazer a comunicação entre dois sistemas PDV que é o modo bidirecional, foi incluído o *Socket* servidor no sistema PDV também. Dessa forma, cada sistema de gerenciamento de replicação esta executando em paralelo o *Socket* cliente e *Socket* servidor, exceto o servidor central que executa somente o *Socket* servidor. A Figura 27, o sistema PDV realiza a comunicação com outro sistema PDV através dos *Socket* cliente e servidor, dessa maneira, é atualizada as informações do sistema PDV no outro sistema PDV e vice-versa.

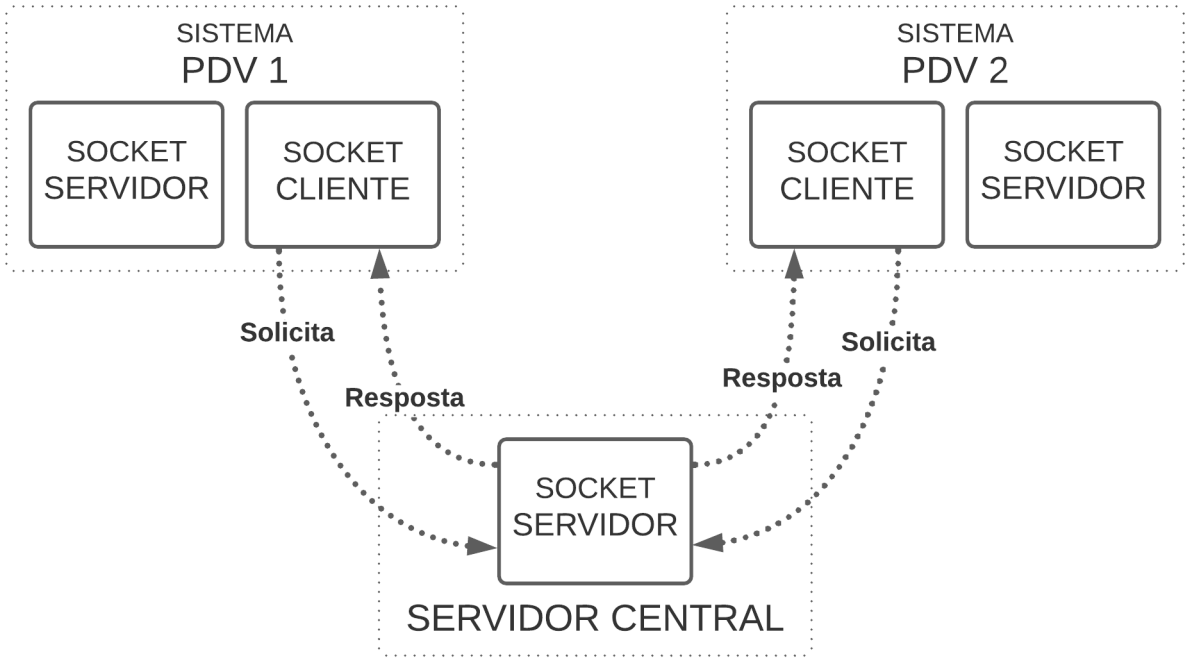
Figura 27 – Processo Bidirecional



Fonte: Desenvolvido pelo autor (2021).

Tanto quanto Figura 27 e a Figura 28, o cliente solicita dados para o servidor enviando instruções e o servidor retorna mensagens através de instruções também. O sistema de gerenciamento de replicação do servidor central realiza comunicação com todos os sistemas PDV das lojas e somente fica habilitado o *Socket* servidor. A Figura 28 apresenta o processo Cliente-Servidor.

Figura 28 – Processo Cliente-Servidor



Fonte: Desenvolvido pelo autor (2021).

Inicialmente o projeto da solução foi estruturada utilizando apenas uma *Thread*. Devido aos processos do *Socket* cliente e servidor que executam em paralelo, foi necessário adicionar mais uma *Thread*. Portanto uma *thread* para o processo cliente e outra *Thread* para o processo servidor. Enquanto o cliente solicita dados, o servidor responde para outros clientes que solicitaram.

A comunicação com todos os sistemas de gerenciamento de replicação é realizada de forma individual, o cliente se conecta apenas com um servidor e executa todos os processos disponíveis até que ambos os bancos de dados estejam sincronizados. No final, o cliente se desconecta do servidor e busca o próximo servidor da lista para se conectar, consequentemente, repetindo todos os processos. O próximo servidor pode ser um sistema PDV ou um servidor central.

As configurações do *Socket* servidor exige que seja configurada a porta para a comunicação dos clientes, portanto, não foi possível manter o número da porta padrão devido aos *Sockets* cliente e servidor que executam na mesma máquina. As portas foram padronizadas seguindo o número da loja e o número do caixa para não haver duplicidades. A Figura 29 mostra como foi configurada o número da porta do *Socket*.

Figura 29 – Configuração da porta do *Socket*

```
Self.FNroPDV := ANroPDV;
Self.FHost   := AHost;
Self.FIP     := AIP;

// Configura o número da porta do servidor conforme o código da loja e o número do PDV
Self.FPort := StrToInt('11' + Format('%2.2d',[ACodLoja]) + Format('%2.2d',[ANroPDV]));
```

Fonte: Desenvolvido pelo autor (2021).

Em consequência de utilizar duas *Threads*, a conexão com o banco de dados *PostgreSQL* também foi separada para evitar conflitos de transações para a mesma conexão dentro do sistema de gerenciamento de replicação. Portanto, existe uma conexão ao banco de dados do processo cliente e outra do processo servidor.

Os tratamentos de erros foram essenciais para descobrir as falhas do sistema de gerenciamento de replicação. Para o ambiente de programação em *Delphi* foi adicionado o *Exception*, logo, os erros são exibidos na tela para o programador e foi relevante para o progresso do desenvolvimento do sistema.

Figura 30 – Tratamento de erros

```
try
  // Verifica se existe registros que não estão atualizados no banco local do PDV e atualiza
  FDadosReplicacao.AtualizarMovimetoReplicacaoBD;

except
  on E:Exception do
    ShowMessage('Ocorreu um erro ao atualizar movimento replicação: ' + E.Message);
end;
```

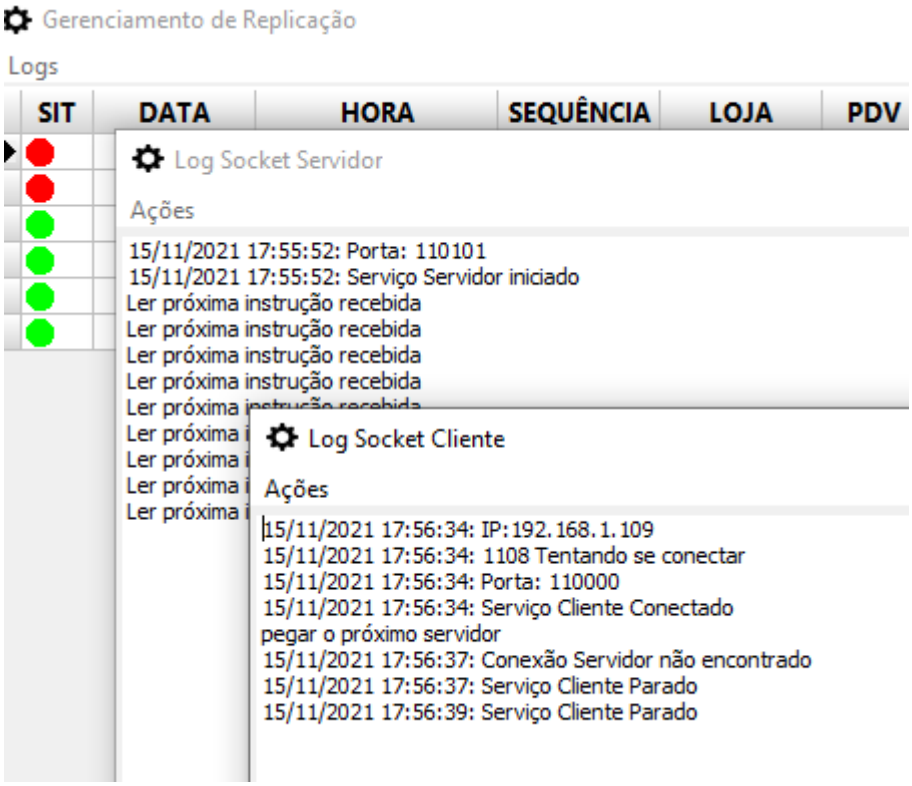
Fonte: Desenvolvido pelo autor (2021).

5.4 REGISTRO

Algumas funcionalidades foram adicionadas durante o tempo de desenvolvimento da aplicação com o propósito de auxiliar na verificação de falhas. Foram adicionadas na aplicação as telas de registros que chamamos de *Logs* dos *Sockets* cliente e servidor. Elas foram proveitosas para analisar o comportamento da comunicação dos *Sockets* entre os sistemas de gerenciamento de replicação.

Para os testes realizados foi possível identificar se o processo reinicia a comunicação automaticamente em caso de perda ou falha de conexão, verificar como o sistema de gerenciamento de replicação se comporta quando é alterado o tempo de repetição do processo das *Threads* e como as instruções estão sendo enviadas e recebidas. A Figura 31 apresenta as telas de *Logs* do *Socket* cliente e servidor.

Figura 31 – Telas de *Logs* do *Socket* cliente e servidor



Fonte: Desenvolvido pelo autor (2021).

Mais algumas funcionalidades para auxiliar a manutenção da aplicação foram adicionadas no menu na tela de *Log* três opções, iniciar e parar a comunicação do *Socket* e limpar a tela de mensagens de *Logs*. Essas funções também auxiliaram para forçar uma falha de conexão e analisar o comportamento do sistema de gerenciamento de replicação. A Figura 32 apresenta essas opções disponíveis.

Figura 32 – Botão *Start*, *Stop* e *Clear*

Fonte: Desenvolvido pelo autor (2021).

5.5 AMBIENTE DE TESTE

Os primeiros testes foram realizados utilizando apenas uma máquina com um banco de dados ativo. Todavia, para simular em um ambiente mais próximo da realidade, foi instalado uma máquina virtual com todos os requisitos necessários para rodar a aplicação do gerenciamento de replicação.

Com dois sistemas de gerenciamento de replicação ativos foram inseridos dados manualmente na tabela de movimento de replicação para o sistema de gerenciamento de replicação automaticamente replicar os dados para outro sistema. Dessa forma foram realizados os testes, verificando as replicações de dados do cliente no servidor e dados do servidor no cliente.

Como é um ambiente de testes controlado, é possível configurar o tempo de execução das *Threads*, quanto menor o tempo de iteração dos processos de um cliente e de um servidor, mais processamento da máquina irá consumir e menor será o tempo de sincronização dos dados. Para um ambiente controlado, o melhor tempo foi de cinco segundos para processar cada processo dentro das *Threads*, sem consumir muito processamento da máquina. É possível alterar esse tempo no arquivo de inicialização já apresentada na seção pré-configuração.

O tempo que leva para se conectar com um novo servidor depende da quantidade de dados que um sistema de gerenciamento de replicação tem para replicar. Quanto mais dados existir para sincronizar, mais tempo irá levar para se conectar com um novo servidor. Durante 30 segundos, se o cliente não receber nenhuma resposta do servidor, automaticamente irá se desconectar e conectar com um novo servidor. O tempo de 30 segundos não é configurável, pois foi estabelecido dentro do programa de gerenciamento de replicação.

6 CONCLUSÃO

O principal objetivo para a empresa *STORE* é centralizar as informações de todas as lojas para garantir a integridade dos dados e também o acesso às informações. É uma mudança forte para a organização da empresa. Para esse propósito, o sistema de gerenciamento de replicação desenvolvido neste trabalho foi eficiente nas etapas iniciais do processo do desenvolvimento da aplicação. Mesmo que os testes tenham sido realizados em um ambiente controlado utilizando uma máquina virtual, o sistema apresentou consistência de dados e garantiu a confiabilidade na execução dos processos de replicação.

O uso da interface do sistema de gerenciamento da aplicação permite mais clareza sobre os dados que estão sendo replicados, facilitando dessa forma para o operador ou o administrador do sistema a manutenção dos registros de atualizações do banco de dados. Além disso, é permite identificar os problemas de comunicação através da tela de registro dos *logs*, onde é exibida toda a intercomunicação do *Socket*.

Verificou-se que é possível integrar o sistema de gerenciamento de replicação com diferentes bancos de dados. Para a empresa *STORE* é utilizada no servidor central o banco de dados *Informix*.

Para implantação nas lojas, o sistema de gerenciamento de replicação é acoplado ao sistema PDV sendo necessário realizar poucos ajustes internamente. Dessa forma, a empresa *STORE* poderá implantar o sistema em pouco tempo.

No gerenciamento de replicação somente o cliente solicita dados para o servidor e o servidor responde com os dados solicitados, desta maneira, houve um maior controle da execução dos processos, pois o servidor que o cliente irá acessar pode estar disponível ou indisponível. Também o cliente pode requisitar os dados quando desejar, desse modo, o sistema PDV não precisa ficar sempre disponível.

Com os benefícios deste trabalho, a empresa *STORE* está buscando adaptar-se às sugestões do método proposto, visando aplicar nas demais lojas da empresa, identificando as melhorias necessárias e se adaptando ao ambiente do mundo real.

REFERÊNCIAS

AKHTER, S. *et al.* Corba based distributed framework for gpgpu processing. Journal of Engineering Research, v. 3, jan. 2021.

BALTZAN, P.; PHILLIPS, A. **Sistema de Informação**: A importância e as responsabilidades do pessoal de ti nas tomadas de decisões. Porto Alegre, RS: AMGH Editora Ltda, 2012.

BATISTA, E. d. O. **Sistemas de informação**. [S.l.]: Saraiva Educação SA, 2017.

CAMÕES, R. S. J. As abstrações relevantes nas camadas de arquitetura cliente-servidor em um sistema distribuído. **TECNOLOGIAS EM PROJEÇÃO**, v. 7, n. 1, p. 49–61, 2016.

CARVALHO, V. **PostgreSQL: Banco de dados para aplicações web modernas**. [S.l.]: Editora Casa do Código, 2017.

CAWENDE, O. J. J. **Sincronização e Reconciliação de Dados em Base de Dados**. Tese (Doutorado), 2018.

COSTA, J. M. da; ROSA, S. de O. Gestão empresarial aplicada a empresas de tecnologia. **Humanidades & Inovação**, v. 7, n. 17, p. 36–43, 2020.

COULOURIS, G. **Caracterização de Sistemas Distribuídos**. Porto Alegre, RS: Bookman Editora, 2014.

CUNHA, N. C. Utilização de tecnologia da informação no desempenho organizacional. GETEC, 2020.

ELMASRI, R. A.; NAVATHE, S. B. **Sistemas de Banco de Dados**. 7. ed. São Paulo, SP: Pearson Education do Brasil, 2018.

ETZKON, L. H. **Introducion to Middleware**: Web services, object components and cloud computing. Huntsville, EUA: CRC Press, 2017.

FALSARELLA, O. M.; JANNUZZI, C. A. S. C. Planejamento estratégico empresarial e planejamento de tecnologia de informação e comunicação: uma abordagem utilizando projetos. **Gestão & Produção**, SciELO Brasil, v. 24, n. 3, p. 610–621, 2017.

FERREIRA, W. O.; KNOP, I. de O. Estruturação de aplicações distribuídas com a arquitetura rest. **Caderno de Estudos em Sistemas de Informação**, v. 3, n. 1, 2017.

HUSEINI, B.; MEMETI, A. Implementation issue analysis of java rmi and corba. **Journal of Natural Sciences and Mathematics of UT**, Faculty of Natural Sciences and Mathematics-University of Tetova, v. 4, n. 7-8, p. 85–94, 2019.

JUNIOR, R. d. S. C. Uma abordagem para evoluir sistemas web legado para web services. 2017.

LAGE, W. G.; ALVES, I. N. A utilização de sistemas distribuídos no desenvolvimento de material instrucional para a inclusão digital do projeto conhecendo o linux. **Revista Ciências Exatas Tecnologia**, v. 10, n. 10, p. 41–45, 2015.

LECHETA, R. R. **Web Services RESTful: Aprenda a criar web services RESTful em Java na nuvem do Google**. [S.l.]: Novatec Editora, 2015.

LIZAMA, O. *et al.* Redes de computadores: Arquitectura cliente-servidor. **Universidad Tecnica Federico Santa Maria**, p. 1–8, 2016.

MACÁK, B. M. **Tools for big data analysis**. [S.l.]: Brno, 2018.

MARQUES, A. I. A. **Desenvolvimento de API para aplicação cloud**. Tese (Doutorado), 2018.

MOCHALOV, V.; BRATCHENKO, N. Y.; YAKOVLEV, S. Analytical model of object request broker based on corba standard. In: IOP PUBLISHING. **Journal of Physics: Conference Series**. [S.l.], 2018. v. 1015, n. 2, p. 022012.

MONTEIRO, E. R. *et al.* **Sistemas Distribuídos**. Porto Alegre, RS: Mirela Favaretto, 2020.

MORAES, J. P. *et al.* Tecnologia da informação, sistemas de informações gerenciais e gestão do conhecimento com vistas à criação de vantagens competitivas: revisão de literatura. **Revista visão: gestão organizacional**, v. 7, n. 1, p. 39–51, 2018.

MOURA, W. de V. *et al.* Benefícios da tecnologia da informação para as estratégias empresariais: Uma revisão integrativa. **Revista Ciência & Saberes-UniFacema**, v. 3, n. 4, p. 732–739, 2018.

NETO, M. de S. **Virtualização - 2ª Edição: Tecnologia Central do Datacenter**. BRASPORT, 2015. ISBN 9788574527611. Disponível em: <https://books.google.com.br/books?id=4xlSCwAAQBAJ>.

PADOLE, M. C. Big data analysis on heterogeneous distributed systems using remote method invocation. **IOSR Journal of Computer Engineering (IOSR-JCE)**, v. 19, n. 2, p. 70–75, 2017.

PICHETTI, R. F. **Banco de Dados**. Porto Alegre, RS: SAGAH, 2020.

RIBEIRO, S. *et al.* Um sistema de comunicação via socket em uma rede wi-fi para controle de um robô de inspeção. **HOLOS**, Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, v. 2, p. 424–435, 2017.

ROCKENBACH, D. A. *et al.* Estudo comparativo de bancos de dados nosql. **Revista Eletrônica Argentina-Brasil de Tecnologias da Informação e da Comunicação**, v. 1, n. 8, 2018.

SCHENFELD, M. C. *et al.* Arquitetura para fog computing em sistemas de middleware para internet das coisas. In: SBC. **Anais do XLIII Seminário Integrado de Software e Hardware**. [S.l.], 2016. p. 199–209.

STEEN, M. V.; TANENBAUM, A. S. **Distributed systems**. [S.l.]: Maarten van Steen Leiden, The Netherlands, 2017.

TEIXEIRA, M. A.; CATANDUVA, S. Camada de aplicação: Programação de sockets. 2018.

VERAS, M. **Computação em Nuvem**: Nova arquitetura de ti. Rio de Janeiro, RJ: BRASPORT, 2015.