Monografia de Graduação

Arquitetura e Implementação de um cliente OPC para aquisição de dados na indústria do petróleo

Gustavo Bezerra Paz Leitão

Natal, julho de 2006

UFRN - CT - NUPEG - Campus Universitário - CEP: 59070-970 - Natal-RN - Brasil
Fone-Fax: (84)3215 3773 - www.nupeg.ufrn.br - prhanp14@nupeg.ufrn.br

Universidade Federal do Rio Grande do Norte
Centro de Tecnologia
Departamento de Engenharia de Computação e Automação

ARQUITETURA E IMPLEMENTAÇÃO DE UM CLIENTE OPC PARA AQUISIÇÃO DE DADOS NA INDÚSTRIA DO PETRÓLEO

Gustavo Bezerra Paz Leitão

Monografia submetida ao Departamento de Engenharia de Computação e Automação do Centro de Tecnologia da Universidade Federal do Rio Grande do Norte, como parte dos requisitos necessários para obtenção do grau de Engenheiro de Computação.

Orientador: Prof. Dr. Luiz Affonso H. Guedes
Co-orientador: Eng. Clauber Bezerra

Natal, Julho de 2006

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO

Aprovada em 25 de Julho de 2006 pela comissão examinadora, formada pelos seguintes membros:

Prof. Dr. Luiz Affonso H. Guedes (Orientador)
Departamento de Engenharia de Computação e Automação - UFRN

Eng. Clauber Bezerra (Co-Orientador)
Departamento de Engenharia de Computação e Automação - UFRN

Prof. Dr. André Laurindo Maitelli (Examinador Interno)
Departamento de Engenharia de Computação e Automação - UFRN

Prof. Alessandro J. de Souza (Examinador Interno)
Departamento de Engenharia de Computação e Automação - UFRN

à Lala

Resumo

A comunicação com os diversos tipos de dispositivos de chão de fábrica se tornou uma necessidade para indústria. Porém essa comunicação era uma tarefa bastante complexa, tendo em vista que cada fabricante desenvolvia seus próprios *drivers* e protocolos de comunicação. Havia a necessidade do desenvolvimento de um padrão. O **OPC** (*OLE for Process Control*), foi criado a fim de solucionar tal paradigma. Utilizando a tecnologia de comunicação entre processos **COM/DCOM** criados pela Microsoft, o **OPC** é um padrão industrial para intercomunicação entre sistemas baseado na arquitetura de cliente/servidor. Este trabalho visa detalhar o desenvolvimento prático de um cliente **OPC**, utilizado como uma forma alternativa de aquisição de dados para um sistema de Gerência de Informação (GERINF) na indústria de petróleo.

Abstract

The communication with the different types of plant floor sensors has become an industrial necessity. However, such communication used to be a very complex task considering each vendor developed their own drivers and communication protocols. There was a necessity for the development of a pattern. The OPC (OLE for Process Control) was created to solve the issue. OPC is a public industrial standard that offers interoperability between systems based on the client/server architecture. It uses the technology of communication between COM/DCOM processes created by Microsoft. In this project, we will be giving details about the practical development of an OPC client used as an alternative way to obtain data for an information management system (GERINF) in the oil industry.

Agradecimentos

Aos professores Luiz Affonso, Adrião Dória e André Laurindo Maitelli por ajudarem na minha formação acadêmica e pessoal.

Aos meus pais e irmãos pelo apoio incondicional.

Aos amigos Leonardo Guanabara e Heitor Penalva por ajudarem no desenvolvimento deste projeto.

Aos companheiros de projeto Alessandro Souza, Clauber Bezerra, Rafael Feijó e Wany Leydiane .

À ANP pelo apoio financeiro e ao LAMP do Departamento de Engenharia de Computação e Automação pelo apoio técnico.

Lista de Símbolos

API	Application Programming Interface
CLP	Controlador Lógico Programável
COM	Component Object Model OPC - OLE for Process Control
DA	Data Access
DCOM	Distributed Component Object Model
DDE	Dynamic Data Exchange
DX	data eXchange
EPS	Enterprise Production Systems
ERP	Enterprise Resource Planning
GUID	Global Unique Identifiers
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IP	Internet Protocol
ISO	International Organization for Standardization
J2EE	Java 2 Enterprise Edition
LAMP	Laboratório de Avaliação de Medições em Petróleo
MVC	Model View Controller
NetDDE	Network Dynamic Data Exchange
OLE	Object Linking Embedding
OSI	Open Systems Interconnection
SCADA	Supervisory Control and Data Acquisition
TCP	Transmission Control Protocol
UFRN	Universidade Federal do Rio Grande do Norte
VB	Visual Basic
XML	eXtensible Markup Language
Web	World Wide Web

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

"Automação (do inglês Automation) é um sistema automático de controle pelo qual os mecanismos verificam seu próprio funcionamento, efetuando medições e introduzindo correções, sem a interferência do homem" (Holanda, 1975). Foi sem dúvida uma das grandes revoluções tecnológicas do século XX, pois possibilitou otimizar processos antes realizados pela força humana e desempenhar tarefas antes impossíveis para o homem. Um sistema de Automação pode ser melhor entendido se o dividirmos em níveis como ilustra a figura 1.1.

Figura 1.1: Modelo em camadas de um Sistema de Automação

Na base da pirâmide encontramos o nível de sensores e atuadores. São eles os responsáveis pela interação direta com o processo, fazendo a leitura das variáveis relevantes através dos sensores e interferindo no processo por intermédio dos atuadores. No nível imediatamente acima encontramos os *controladores lógico programáveis* (CLP) e os *Supervisory Control and Data Acquisition* (SCADA) que realizam o controle regulatório e supervisão, respectivamente. O terceiro nível, *Enterprise Production Systems* (EPS), é o responsável pela gerência de informação. São armazenados dados referentes aos processos para utiliza-los como informação útil. No topo da pirâmide estão os sistemas responsáveis pela transformação desses dados em informação de negócio, *Enterprise Resource Planning* (ERP) (Souza et al., 2005).

A integração das informações de chão-de-fábrica aos sistemas corporativos possibilita a otimização da gerência refletindo, positivamente, nos aspectos financeiros. Dentre os vários benefícios existentes nessa integração, podemos destacar: a disponibilidade para comprometimento, redução do tempo de produção e a implantação e otimização da cadeia de suprimentos.

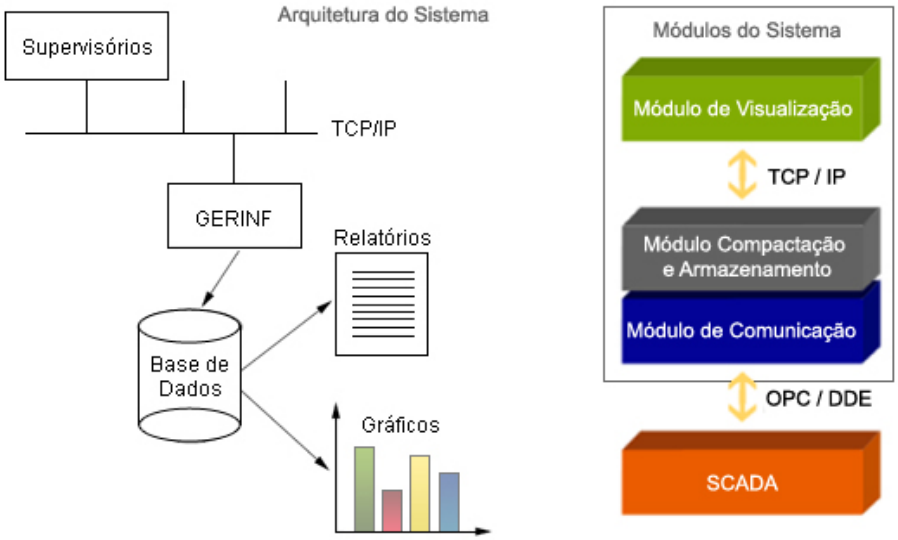
Este trabalho busca detalhar procedimentos de aquisição de dados utilizando o padrão de comunicação OPC (OLE for Process Control) (Iwanitz e Lange, 2001) implementado durante o desenvolvimento de um sistema de gerência de informação de processos industriais, o qual enquadra-se na categoria EPS da pirâmide de automação, figura 1.1.

1.1 O GERINF

O GERINF (Gerência de Informação) é um sistema de gerência de informação que visa disponibilizar dados provenientes de sistemas SCADA, através de interface

web. Permite, desta forma, que tais informações sejam acessadas de qualquer local que possua conexão com a rede local.

O sistema possibilita tanto a visualização on-line dos dados, bem como a realização de consultas históricas. Isto é possível pelo fato de possuir uma base de dados onde são armazenados todos os dados considerados relevantes pelo algoritmo de compactação utilizado. Todo o sistema pode ser configurado e parametrizado remotamente, através de um simples *browser* via protocolo HTTP (HyperText Transfer Protocol). A figura 1.2a ilustra a arquitetura do sistema.



(a) (b)

Figura 1.2: Arquitetura e Módulos do Sistema

O GERINF está organizado em três módulos independentes, como mostra a figura 1.2b. São eles: Módulo de Comunicação, Módulo de Compactação e Módulo de Visualização. O Módulo de Comunicação funciona como interface entre os sistemas supervisórios e o Gerinf sendo o responsável pela aquisição de dados dos supervisórios. Em seguida, O Módulo de Compactação utiliza um algoritmo de compressão nos dados

e os armazenam na base de dados para, por fim, serem manipulados pelo Módulo de Visualização.

1.1.1 Módulo de Comunicação

O supervisório é um software destinado a promover a interface homem/máquina e proporciona uma supervisão plena de um processo através de telas devidamente configuradas, representando o processo, onde estas podem ser animadas em função das informações recebidas ou enviadas pelo CLP, controlador, etc. Atualmente, são disponíveis comercialmente, vários programas supervisórios, tais como: INTOUCH, ELIPSE, AIMAX, FIX-32, VIEW e CIMPLIST.

Para adquirir as informações provenientes do campo é necessária uma interface de comunicação entre o software Gerinf e o supervisório. Devido ao fato que os supervisórios já existem e é inviável manuseá-los para gerar um banco de dados, é necessário conectar-se no supervisório para adquirir essas informações e enviá-las para o banco de dados do Gerinf. Isto é feito, no nosso caso, utilizando-se uma interface de comunicação através do protocolo DDE (*Dynamic Data Exchange*).

O DDE é um protocolo para troca dinâmica de informações entre aplicativos Windows. Através dele é possível a duas ou mais aplicações (um servidor e um ou mais clientes) conversarem através de mensagens padronizadas que permitem ao cliente obter dados a partir do servidor.

É nesse contexto de aquisição de dados dos supervisórios para o sistema que se encaixa o foco do presente trabalho. Este teve como objetivo desenvolver uma maneira alternativa para a captura dessas informações utilizando o padrão de comunicação OPC(*OLE for Process Control*). Esta alternativa não substituirá a já existente(DDE),

mas sim será mais uma opção para a aquisição dos dados. O administrador do sistema é quem decidirá a forma de aquisição a ser utilizada, dentre outras razões, dependendo da disponibilidade protocolais dos sistemas SCADA a serem conectados.

De acordo com essa abordagem, o software de gerência inicia uma conversação com o software supervisório, solicitando-lhe os dados relevantes ao usuário para o gerenciamento do processo em questão. Feito isto, o supervisório começa a transmitir os dados do processo que está sendo monitorado, para o software de gerência a cada intervalo de amostragem do sistema e as envia para o módulo de compactação e armazenamento, para a sua posterior recuperação quando solicitado pelo usuário.

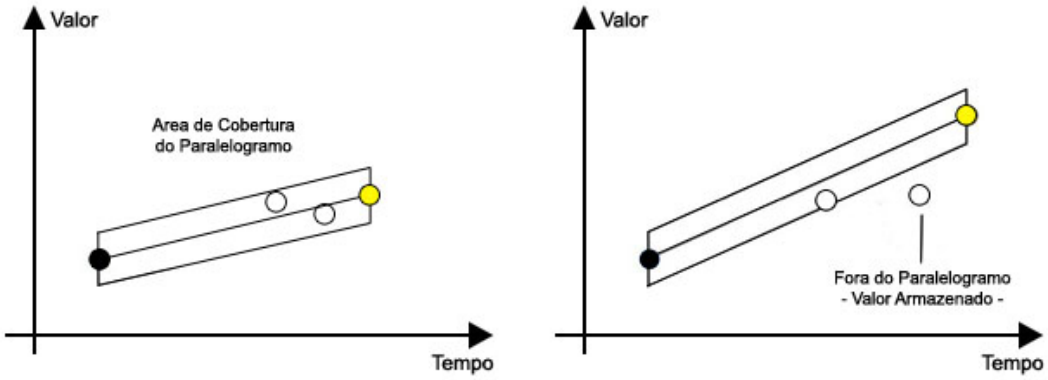
1.1.2 Módulo de Compactação

A compressão dos dados coletados do sistema SCADA é realizada para eliminar informações redundantes e para tornar possível o armazenamento de um grande volume de informações no banco de dados. Espera-se que o algoritmo de compressão de dados não elimine informações relevantes, tenha uma taxa de compressão alta e que os dados reconstituídos sejam o mais próximo possível dos originais.

Para atender a essas necessidades foi utilizado um algoritmo de compressão bastante eficiente chamado Swinging Doors (Filho e Szuter, 1993), usado no produto PI da OSI Software Inc. O algoritmo Swinging Doors contem três parâmetros que são definidos de acordo com a variável cujas informações serão comprimidas. São eles: tempo mínimo de compressão, tempo máximo de compressão e desvio de compressão. Quanto maior for o tempo máximo de compressão e o desvio de compressão, maior será a taxa de compressão.

No algoritmo implementado para o nosso sistema, o parâmetro tempo mínimo de

compressão não foi utilizado, essa adaptação foi feita para que informações relevantes não possam ser perdidas. O algoritmo Swinging Doors cria uma área de cobertura no formato de um paralelogramo e com largura igual ao dobro do desvio de compressão. Este paralelogramo se estende desde o último valor armazenado até o último valor recebido. Se qualquer um dos valores recebidos entre o último armazenado e o último recebido ficar fora da área de cobertura, então o valor anterior ao último recebido será armazenado no banco de dados. O último valor recebido é sempre armazenado se o tempo transcorrido desde o último valor armazenado for maior que o tempo máximo de compressão. O funcionamento deste algoritmo é explicado nas figuras 4a e 4b.



(a) (b)

Figura 1.3: Algoritmo de compactação de dados Swinging Doors.

A bola preta na figura 1.3 representa um dado que foi armazenado, a bola branca representa um dado que não foi armazenado e a bola amarela representa o último dado recebido. A figura 1.3a apresenta um paralelogramo formado pelo último valor armazenado e o último valor recebido. Nenhum dos valores intermediários ficou fora do paralelogramo, portanto nada será armazenado. A figura 1.3b mostra outra situação, onde um dos valores intermediários ficou fora da área do paralelogramo.

Logo, o valor anterior ao último valor recebido (em amarelo), será armazenado na base da dados.

1.1.3 Módulo de Visualização

Sistemas tradicionais de gerência de processo têm sido instalados como sistemas autônomos (Standalone) com limitadas interações com os ambientes de escritório da corporação, o que leva à necessidade dos usuários destes ambientes fazerem uso de outros aplicativos, tais como planilhas, para ter uma posição real do que está acontecendo na linha de produção.

Após analisar as inúmeras aplicações para resolver esse problema de conectividade, entre o chão de fábrica e a gerência do processo, tornou-se claro que a solução estaria no desenvolvimento de um sistema em ambiente *Web*. O navegador Web (*browser*) se tornou, para o usuário final, o ambiente onde ele faz tudo o que precisa na empresa, desde a troca de e-mails até a visualização de relatórios, gráficos e imagens. A utilização de um sistema Web pode trazer as seguintes vantagens: facilidade e baixo custo de manutenção e atualização dos sistemas, pois para efetuar a manutenção do sistema será necessário alterar apenas arquivos HTML, Applets Java e Scripts que se encontram no servidor; possibilidade de execução de aplicações em máquinas sem grande poder de processamento e sem a necessidade de constantes atualizações de hardware e software; o sistema não fica condicionado à instalação em nenhuma máquina cliente, pois todos os computadores têm *browser* previamente instalados.

O sistema foi desenvolvido baseado na tecnologia Java, mais especificamente a plataforma Java 2 Enterprise Edition (J2EE) (Bond et al., 2003). Esta tecnologia consiste em um conjunto de especificações coordenadas e um guia de práticas que

permitem o desenvolvimento, instalação, execução e gerenciamento de aplicações n-camadas no servidor.

O principal objetivo da utilização da Plataforma J2EE foi explorar suas capacidades para suporte a aplicações corporativas tais como: robustez, estabilidade, segurança e desempenho. Tais capacidades foram de extrema importância para nosso projeto, haja vista, a excessiva quantidade de requisições recebidas e volume de informações tratadas em cada requisição.

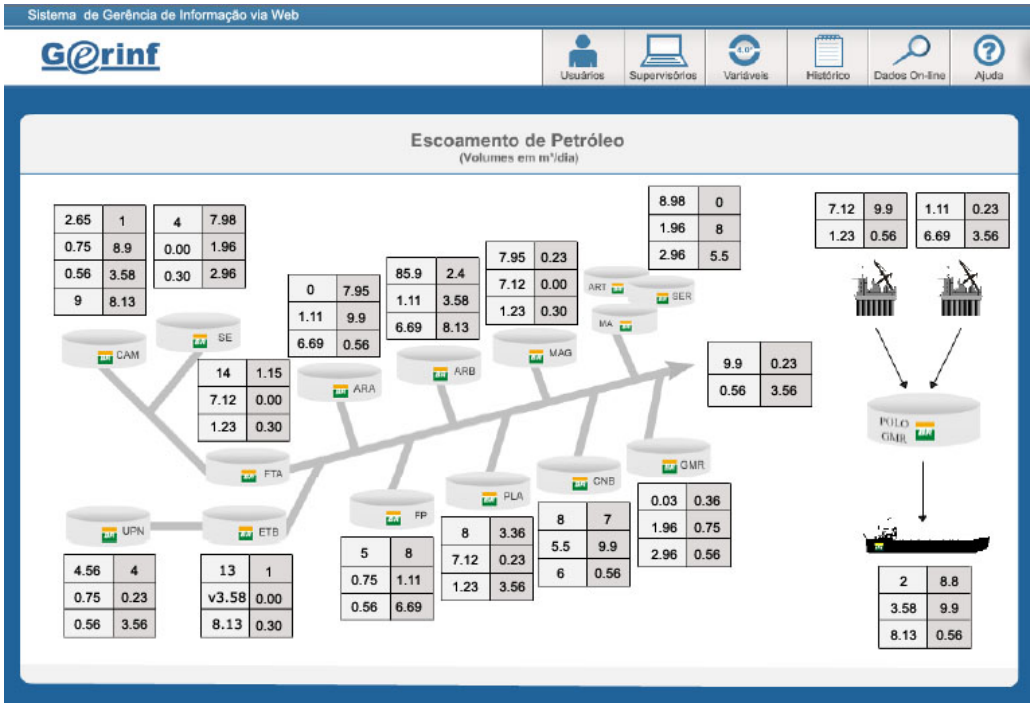


Figura 1.4: Tela de escoamento de petroleo

Para a parte de visualização de dados em tempo real foi escolhido a tecnologia Flash. Tal solução foi utilizada pois o Flash é uma poderosa ferramenta para desenvolvimento em ambiente *web*, possibilitando adquirir informações sem a necessidade

do carregamento pleno da página em foco. Ainda mais, para executa-lo basta a necessidade de um plug-in que já vem instalados por padrão em todas as máquinas que utilizam o Windows como sistema operacional. Outra Solução seria a utilização de Applets Java, porém tal alternativa foi descartada pela necessidade de instalar a máquina virtual Java em todas os computadores que necessitassem visualizar os dados on-line no sistema. Uma demostração de uma das telas de visualização on-line pode ser vista na figura 1.4.

O desenvolvimento do sistema seguiu um padrão de projeto, *Design Patterns*, bastante difundido no desenvolvimento de aplicações *web*, chamado MVC (*Model View Controller*) (Tichy, 1997), possibilitando um maior controle sobre as requisições vindas das máquinas clientes para a aplicação. Esse padrão foi implementado através do *framework* de aplicação Struts da *Apache Software Foundation* (Husted et al., 2004).

Capítulo 2

COM

Este capítulo tem como principal objetivo explorar de forma teórica e prática a utilização do **COM** (*Component Object Model*), uma tecnologia criada pela Microsoft para definições de componentes que possam ser reutilizados por outras aplicações. É por esta implementação que se tornou possível a troca de informações em tempo de execução por duas aplicações diferentes, como por exemplo, inserir tabelas do Microsoft Excel no Word, dentre outras interconexões.

O **OPC**, objeto de estudo deste trabalho, nada mais é do que definições de como utilizar e implementar os objetos COM/DCOM. Isso é possível graças a especificações de utilização mantidas e publicadas pela **OPC Foundation**. Esta é uma fundação com mais de trezentos membros pelo mundo, dentre eles, os principais fornecedores de sistemas de controle e instrumentação.

A Microsoft é um forte membro da OPC Foundation. Ela age como um conselheiro tecnológico e fornece inspeções prévias das especificações a serem lançadas. As outras companhias guiam o trabalho da organização, testando as tecnologias desenvolvidas e expondo suas necessidades.

2.1 Histórico

As etapas do desenvolvimento do Microsoft COM/DCOM podem ser visualizado na figura 2.1, que segue.

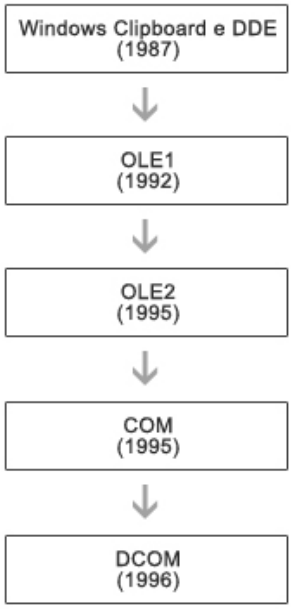


Figura 2.1: Desenvolvimento Histórico do COM/DCOM

O Windows Clipboard foi a primeira solução criada pela Microsoft que permitia a transferência de informações entre aplicações. Tal tecnologia foi baseada no simples mecanismo de "copiar e colar" e possuía uma fácil utilização. O **DDE** (*Dynamic Data Exchange*) foi uma outra solução criada para intercomunicação entre diferentes aplicações, entretanto não era de fácil utilização.

O mecanismo clipboard permitia a criação de documentos compostos. Isto significa que se tornou possível colar tabelas do Microsoft Excel no Word, por exemplo. O problema é que esta conexão não era dinâmica, não era inteligente. As alterações não eram transferidas automaticamente. Para suprir esta necessidade a Microsoft

desenvolveu o OLE1. Com a implementação deste padrão a intercomunicação entre aplicações se tornou dinâmica, ou seja, as alterações nas tabelas do Excel, agora, seriam automaticamente transferidas para o documento do Word, no nosso exemplo. O DDE serviu então como protocolo de comunicação. O desenvolvimento do OLE2 permitiu a interação entre qualquer tipo de aplicação, não somente entre softwares de processamento de texto, como anteriormente.

Em 1995 tal tecnologia de intercomunicação foi chamada de Component Object Model (*COM*) e se tornou a base para o desenvolvimento de aplicações orientadas a objeto.

Lançado em 1996, junto com a plataforma Windows NT, o **DCOM** (*Distributed Component Object Model*) veio para suprir a necessidade da comunicação entre aplicações localizadas em computadores diferentes. O DCOM é, basicamente, um conjunto de definições para permitir a implementação de aplicações distribuídas em uma arquitetura clente-servidor (Fonseca, 2002).

Vale salientar que o desenvolvimento destas tecnologias, pela Microsoft, foi impulsionado pela criação do publish and subscribe, pela Apple, na década de 80. Esta tecnologia permitia "publicar" parte de um documento que depois podia ser "subscrito" (ou utilizado) por outro documento. Se o documento original fosse alterado, a parte subscrita pelo outro documento também seria alterada, tornando-se assim numa forma de interligar documentos, mas como os documentos são manipulados por aplicações, tecnicamente era uma forma de integrar aplicações.

2.2 Introdução ao COM

COM (*Component Object Model*), como já explicitado, é um padrão para criação de componentes que possam ser reutilizados. Um objeto COM é um código binário que pode ser utilizado por qualquer outro programa, escrito em qualquer linguagem. Os objetos COM visam trazer a modularização dos sistemas. Uma vez criado e publicado sua documentação, o objeto COM funciona como uma caixa preta onde o usuário não precisa conhecer detalhes sobre sua implementação e sim, apenas, saber utilizar tal componente.

Um bom exemplo seria: se você aprendeu a usar um recurso qualquer, por exemplo, utilizar processamento de imagem para realizar um redimensionamento, pode encapsular este código em um componente que poderá ser utilizado por outra pessoa que não possui, necessariamente, o conhecimento dos algoritmos de redimensionamento de imagens. Este programador utilizará o componente como uma caixa preta a partir das definições de suas interfaces incluídas na sua documentação (Fonseca e Filho, 2005).

Outro fator bastante importante dos componentes COM é que qualquer linguagem que possua funções, utilizando ponteiros para elas, pode ser utilizada para escreve-los. Linguagem como C++, Delphi e Java são ideais. Porém até linguagens interpretadas como VB(*Visual Basic*), podem ser utilizadas, desde que o interpretador possa propiciar este serviço em nome da aplicação.

Por fim o COM/DCOM constituem uma revolução conceitual muito importante, pois implica numa grande alavancagem na velocidade e capacidade de uma equipe produzir programas.

2.3 Interfaces

A manipulação dos métodos dos componentes COM/DCOM, são possíveis graças ao conceito de interface. Uma interface, é a porta de entrada dos componentes. É através dela que são disponibilizados as funcionalidades (métodos) do objeto desejado. Cada interface possui "n" métodos e o número máximo de interface de um componente não é especificado. Cada interface é identificada através de um identificador único de 128 bits chamado de **IDD**, detalhado na sessão seguinte.

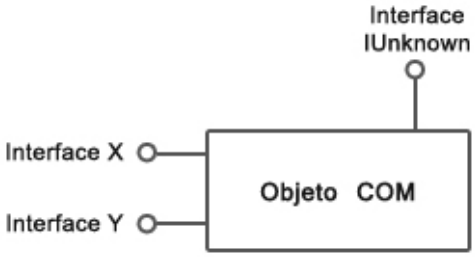


Figura 2.2: Protótipo de Implementação de um Objeto COM

Por definição da Microsoft, todo objeto COM deve implementar a interface padrão **IUnknown**, figura 2.2. Todas as outras interfaces do componente são derivadas dela que por sua vez deve conter três métodos, são eles:

- **QueryInterface** - É utilizado para verificar se uma outra interface específica é suportada pelo objeto COM. É através deste métodos que obteremos o ponteiro para a interface desejada, caso esteja implementada.
- **AddRef** - É utilizado para a administração do ciclo de vida do componente. Seu funcionamento consiste em incrementar um contador de referência toda vez que uma interface é retornada a um cliente.

- **Release** - Assim como o método anterior, também é utilizado para a administração do ciclo de vida do componente, porém seu funcionamento consiste em decrementar o contador de referência. Quando este contador chega a zero o objeto é destruído automaticamente.

Uma interface é imutável, isto é, uma vez publicada sua assinatura nunca muda. A descrição de cada método da interface é feito em sua documentação. A semântica de cada comando não pode mudar. Quando se deseja definir um método adicional, deve-se definir uma nova interface que conterá todos os métodos anteriores acrescido do novo método. Este conceito irá simplificar o controle de versões de um produto de software.

2.4 Identificadores

Esta sessão irá mostrar alguns detalhes dos identificadores utilizados para várias tarefas no COM/DCOM. Todos estes identificadores são GUIDS (*Global Unique Identifiers*). GUID é um código alfanumérico de 128 bits gerados através de um algoritmo que os tornam globalmente únicos. Uma vez gerado não existirá outro igual (Cerqueira, 2000). No nosso contexto, vale salientar:

- CLSID - É o identificar único do componente. Este valor é utilizado como parâmetro para conexão com o componente COM/DCOM no qual desejamos manipular.

- **IDD** - É o identificador único de uma interface. Este valor é utilizado como parâmetro para se requisitar determinada interface através do *QueryInterface*.

- **CATID** - É o identificador de categorias dos componentes. No caso do OPC este valor serve para identificar cada especificação implementada.

2.5 Valores de Retorno dos Métodos

O método de uma interface pode desejar retornar um valor que contém um dado ou o resultado da execução do mesmo: sucesso ou falha. Este valor que indica o status da execução do método, das interfaces, é do tipo **HRESULT** e possui 32 bits de tamanho. A Microsoft já definiu alguns padrões de bits para erros. A tabela 2.1 abaixo mostra constantes pré-definidas que representam alguns dos valores típicos de **HRESULT**. O detalhamento dos valores de retorno de cada método do cliente OPC desenvolvido será melhor indicado no capítulo 4.

Nome	**Significado**
S_OK	Função teve sucesso.
NOERROR	O mesmo que S_OK
S_FALSE	Função não teve sucesso.
E_UNEXPECTED	Falha Inesperada.
E_NOTIMPL	Função membro não implementada.
E_NOINTERFACE	Não suporta a interface solicitada.
E_OUTOFMEMORY	Não pode alocar a quantidade de memória solicitada.
E_FAIL	Falha Inesperada.

Tabela 2.1: Valores Típicos de HRESULT

2.6 Implementação Básica

Para uma implementação de qualquer cliente COM deve-se seguir alguns rigorosos passos, a saber:

1. Inicializar a biblioteca COM;
2. Obter o CLSID do objeto;
3. Criar uma instância do objeto;
4. Usar o objeto COM;
5. Desinstalar o componente.

2.6.1 Inicializando a biblioteca COM

A inicialização da biblioteca COM deve ser o primeiro passo para a utilização de um componente. Tal inicialização se dá através da função *CoInitialize* da API do Windows. O valor de retorno da função é um HRESULT e seu valor deve ser comparado com o valor das constantes pré-definidas afim de saber o resultado da operação.

Abaixo segue um pequeno trecho de código escrito em C++ para a inicialização da biblioteca.

```
HRESULT hr;
hr = CoInitialize(NULL);
if (hr != S_OK) {
    return false;
}
```

2.6.2 Obtendo o CLSID do objeto

Após ter inicializado a biblioteca COM o próximo passo consiste em obter o CLSID do componente a ser conectado. Cada objeto COM possui um CLSID globalmente único e o mesmo fica localizado no *registry* do Windows na seguinte chave:

HKEY_CLASSES_ROOT\CLSID.

Como já explicado, na sessão 2.4, O CLSID de um componente é um código alfanumérico de 128 bits. Afim de facilitar a maneira de identifica-los, foi criado o conceito de ProgID. Este corresponde a um nome legível, que identifica um componente, respeitando o seguinte protótipo:

biblioteca.tipo.versão

Devido a maneira de sua construção, apesar de muito raro, pode ocorrer dois ProgIDs iguais que representam objetos COM distintos.

Os ProgIDs são gravados também no *registry* do Windows, porém na seguinte chave:

HKEY_CLASSES_ROOT

Portanto uma das maneiras de se obter os CLSIDs dos objetos **COM** disponíveis é varrer o registro do windows na chave **HKEY_CLASSES_ROOT\CLSID**. Porém, uma maneira mais interessante é obter o CLSID a partir do ProgID. Isto pode ser feito através da função *CLSIDfromProgID* disponível na API do Windows, caso o ProgID seja conhecido. Caso contrário, pode se varrer o registro do Windows na chave **HKEY_CLASSES_ROOT** e então localizar os ProgIDs disponíveis. Existe

também a função reversa da *CLSIDfromProgID* que obtém o ProgID a partir do CLSID, a saber: *ProgIDfromCLSID*.

Exemplo:

```
HRESULT clsfromProg(LPOLESTR ProgID, CLSID *clsid){
    return CLSIDFromProgID(ProgID,&clsid);
}

HRESULT progfromCls(CLSID clsid, LPOLESTR *ProgID){
    return ProgIDFromCLSID(clsid, &ProgID);
}
```

Abaixo um exemplo de implementação para varrer o registro, a fim de localizar os ProgIDs de objetos COM que implementam servidores OPC. Os valores encontrados são colocados em uma lista para posteriormente serem obtidos os seus CLSIDs através da função *CLSIDfromProgID*.

```
    HKEY hk = HKEY_CLASSES_ROOT;
    char szKey[MAX_KEYLEN];
    int aux =  0;

    for(int nIndex = 0; ::RegEnumKey(hk, nIndex, szKey, MAX_KEYLEN)
    == ERROR_SUCCESS; nIndex++)
    {
        HKEY hProgID;
        char szDummy[MAX_KEYLEN];
        if(::RegOpenKey(hk, szKey, &hProgID)
        == ERROR_SUCCESS)
        {
            LONG lSize = MAX_KEYLEN;
            if(::RegQueryValue(hProgID, "OPC", szDummy, &lSize)
             == ERROR_SUCCESS){
                list[aux++] =  szKey;
            }
```

```
            ::RegCloseKey(hProgID);
        }
    }
```

No caso de busca por CLSIDs de componentes que implementam servidores OPC, a OPC Foundation fez e disponibilizou o OPCServerBrowser. Este componente consiste em um servidor DCOM com o CLSID definido em sua especificação. O cliente portanto não seria obrigado a varrer o registro a fim de localizar servidores disponíveis e sim apenas utilizar funcionalidades do OPCServerBrowser que serão detalhadas no capítulo seguinte.

2.6.3 Criando uma instância do objeto

Para criar uma instância do objeto desejado, devemos chamar a função *COCreateInstance* esta função também está definida na API do Windows e retorna um ponteiro para a interface escolhida pelo usuário.

A assinatura desta função pode ser visualizada abaixo:

```
STDAPI CoCreateInstance(
  REFCLSID rclsid,
  LPUNKNOWN pUnkOuter,
  DWORD dwClsContext,
  REFIID riid,
  LPVOID * ppv
);
```

onde:

- **[in] rclsid** - Identificador(CLSID) da classe do objeto
- **[in] pUnkOuter** - Apontador para IUnknown. usado para criar o objeto como parte de um objeto maior.

- **[in] dwClsContext** - Contexto para rodar o código executável (in server, local server ou remote server).
- **[in] riid** - Referencia para o identificador da interface desejada.
- **[out] ppv** - Endereço da variável que recebe o ponteiro para a interface pedida em riid.

Retorno da função:

Nome	Significado
S_OK	Uma instância do objeto desejado foi criada.
E_NOINTERFACE	A classe não implementa a interface desejada.
REGDB_E_CLASSNOTREG	Classe não registrada no *registry*.

Tabela 2.2: Valores retornados pela função *CoCreateInstance*

Segue abaixo um exemplo de implementação em C++ para instanciar um objeto COM que implementa um servidor OPC.

```
const IID IID_IOPCServer =
{0x39c13a4d,0x011e,0x11d0,{0x96,0x75,0x00,0x20,0xaf,0xd8,0xad,0xb3}};
IOPCServer *m_ptrServer;
HRESULT hr;
hr = CoCreateInstance (clsid,
    NULL,
    CLSCTX_LOCAL_SERVER,
    IID_IOPCServer,
    (void**) &m_ptrServer
    );
```

2.6.4 Usando o objeto COM

Uma vez obtido um ponteiro para o objeto, este pode ser utilizado. é importante lembrar que após a utilização ele deverá ser liberado através do comando **Release**.

```
if (hr == S_OK) {
    m_ptrServer->Read(  );
    m_ptrServer->Release();
}
```

2.6.5 Desinstalando o componente

Quando não se fizer mais necessário o uso do componente, a biblioteca COM deve ser liberada através do comando **CoUninitialize**.

```
    CoUninitialize();
```

Capítulo 3

OPC - OLE for Process Control

Nos últimos anos o uso de softwares para sistemas de controle de processos tem aumentado substancialmente. Softwares como os de supervisão e controle nos quais se necessitam comunicar com os diversos dispositivos de chão-de-fábrica se tornaram indispensáveis em um sistema automático. Entretanto esses dispositivos eram normalmente desenvolvidos por diferentes fabricantes e conseqüentemente possuíam os seus próprios *drivers* e protocolos de comunicação. Logo, a integração e conectividade entre esses diversos dispositivos, se tornava uma tarefa bastante árdua. Havia então a necessidade do desenvolvimento de um padrão de comunicação.

Em 1995, algumas empresas se reuniram com o objetivo de desenvolver este padrão (Fonseca, 2002). Foram envolvidos membros da Microsoft para suporte técnico à solução a ser adotada. Desta forma, foi então criado o padrão OPC (*OLE for Process Control*). Baseado na tecnologia OLE/DCOM o OPC estabeleceu regras de uso dos Componentes COM para acesso à dados em tempo real dentro do sistema operacional Windows. O OPC veio solucionar o paradigma de interconexão dos diversos dispositivos e foi logo aceito pela indústria.

3.1 Comunicação de dados em sistemas tradicionais

Apesar de existirem alguns tipo de redes especificas para ambientes industriais (*Profibus*. *ControlNet*, *Foundation Fieldbus*, etc), a maioria dos fabricantes utilizam a rede *Ethernet* com o protocolo TCP/IP para transmissão de dados. Como o TCP/IP encapsula diferentes tipo de protocolos, cada fabricante desenvolve um *driver* específico para comunicação com seu protocolo proprietário. Portanto, existe a necessidade do uso do *driver* específico para acessar informações dos dispositivos de um determinado fabricante.

Como *Ethernet* é uma rede multi-protocolar e a maioria dos fabricantes utilizam TCP/IP, é possível interligar vários dispositivos de fabricantes diferentes na mesma rede, porém apenas os dispositivos que possuírem o mesmo *driver* de comunicação irão se comunicar entre si.

Para solucionar o problema de comunicação de dados os sistemas tradicionais como os SCADA (*Supervisory Control And Data Acquisition*), utilizados normalmente para comunicar com dispositivos de controle, possuíam um conjunto de diversos drivers para então permitir a comunicação entre estes dispositivos de fabricantes distintos, conforme ilustra a figura 3.1.

Um bom sistema SCADA dever conter um "arsenal" de *drivers* para tornar possível a comunicação com os mais diferentes dispositivos. Existem sistemas SCADA que disponibilizam mais de duzentos drivers de comunicação, o que os tornam preferidos, não necessariamente pelas suas funcionalidades e sim pelas possibilidades de comunicação. Apesar do uso de drivers ter resolvido parcialmente os problemas de comunicação, estes apresentam algumas desvantagens:

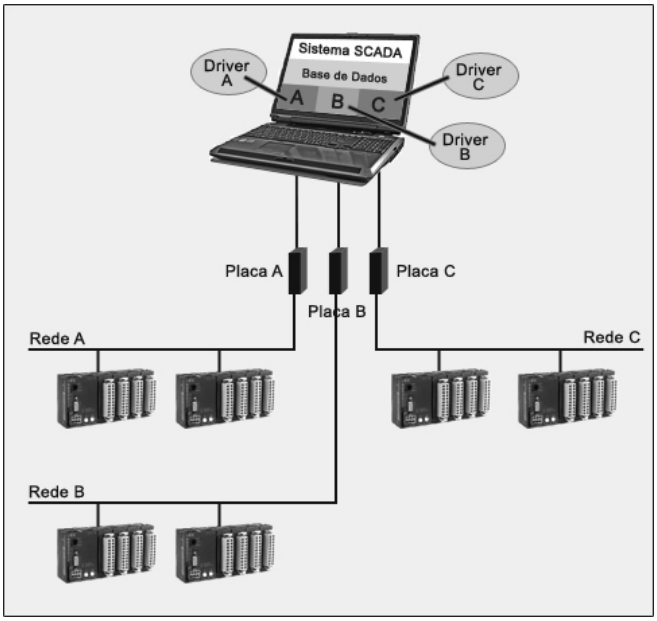


Figura 3.1: Comunicação do Sistema SCADA utilizando *drivers* específicos

- Cada *driver* implica em custo adicional;
- Cada *driver* dispõe de mecanismos específicos para comunicação dos dados, com diferentes desempenhos.
- Cada *driver* tem sua própria interface com o usuário dificultando o treinamento e a manutenção do sistema.

Apesar de tais desvantagens, o uso de drivers como única forma de comunicação entre sistemas perdurou até o surgimento do padrão OPC.

O padrão OPC surgiu como uma forma de estabelecer regras de comunicação entre aplicações. Portanto, o uso do OPC não elimina a necessidade dos *drivers* de comunicação. De forma geral o padrão OPC define interfaces que estão no nível do usuário, acima dos protocolos de comunicação, vide figura 3.2.

Portanto o padrão OPC utiliza a arquitetura cliente/servidor e está localizado na camada de aplicação no modelo OSI-ISO. OPC não define como a aplicação servidor

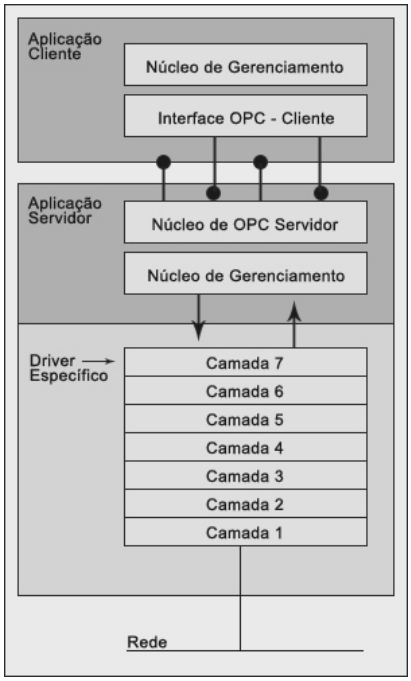


Figura 3.2: Interfaces entre aplicações do Padrão OPC

irá se comportar com relação aos drivers de comunicação e nem como os clientes irão se comportar para outros fins. por isso há grande variedade de diferentes clientes e servidores OPC disponíveis no mercado. Porém qualquer aplicação OPC irá se comunicar entre si, desde que estejam na mesma versão da especificação do padrão.

3.2 Especificações

Existem hoje disponíveis diversas especificações OPC publicadas(ou em processo de avaliação) e mantidas pela OPC Foundation, são elas:

- **OPC Overview** (versão 1.00) - Descrição geral dos campos de aplicações das especificações OPC;

- **OPC Common Definitions and Interfaces** (Versão 1.00) - Definição das funcionalidades básicas para as demais especificações;
- **OPC Data Access Specification** (Versão 3.00) - Definição da interface para leitura e escrita em tempo real;
- **OPC Alarms and Events Specification** (Versão 1.10) - Definição da interface para monitoração de eventos;
- **OPC Historical Data Access Specification** (Versão 1.10) - Definição da interface para utilização de dados históricos;
- **OPC Batch Specification** (Versão 2.00) - Definição da interface para acesso aos dados de processos por batelada (*batch*). Esta especificação é uma extensão da **OPC Data Access Specification**;
- **OPC Security Specification** (Versão 1.00) - Definição da interface para utilização de políticas de segurança;
- **OPC data eXchange (DX) Specification** (Versão 1.00) - Definição da interface para troca direta de dados entre Servidores OPC;
- **OPC and XML Specification** (Versão 1.00) - Integração entre OPC e XML para aplicação via Internet (*web*).

A figura 3.3 mostra uma visão das especificações do padrão.

As especificações apresentadas estão em constante desenvolvimento e atualização, suas últimas versões podem ser conseguidas através do site da **OPC Foundation** (*www.opcfoundation.org*). Tais especificações tem o intuito de estabelecer regras para o desenvolvimento das aplicações clientes e servidor. Os usuários das aplicações desenvolvidas não necessitam, necessariamente, ter conhecimento profundo das especificações e sim apenas o conhecimento do aspecto prático para a utilização do padrão (Iwanitz e Lange, 2001).

No presente trabalho será detalhada e explicada apenas detalhes da especificação de aquisição de dados em tempo real: **OPC Data Access Specification**, pois foi a especificação utilizada para o desenvolvimento do cliente em questão.

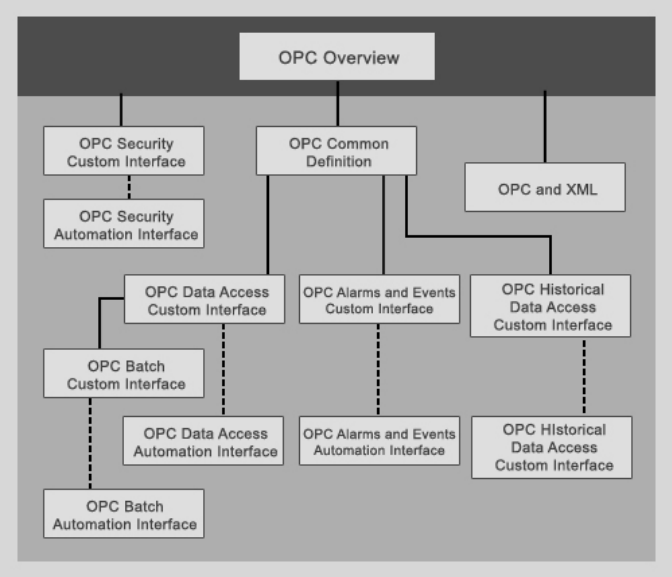


Figura 3.3: Especificações do padrão OPC

3.3 DDE vesus OPC

O DDE é um dos mais utilizados protocolos de comunicação em sistemas de aquisição de dados como os SCADA(*Supervisory Control And Data Acquisition*). É através do **DDE** que o sistema **GERINF**, atualmente, faz a sua aquisição de dados dos supervisórios. O OPC, objeto de estudo deste trabalho, é uma outra alternativa sugerida para a aquisição de dados. Nessa sessão iremos fazer uma breve comparação entre essas duas tecnologias, mostrando vantagens e desvantagens de cada uma delas de diversos pontos de vista. Serão levados em conta os seguintes itens:

- **Padronização** - Existe alguma definição internacional aprovada para o uso desta tecnologia em embiente industrial?
- **Interoperabilidade** - Podemos comunicar dispositivos de diferentes fabricantes um com os outros?

- **Performace** - Eficiência com relação ao tempo necessário para os dados transferidos serem acessados.
- **Acesso remoto** - É possível criar sistemas distribuídos?
- **Interação** - As funcionalidades vão além do puro intercâmbio de dados?
- **Complexidade** - Que tipo de conhecimento é necessário para a criação do produto correspondente?
- **Parcela de Mercado** - Como anda hoje a parcela de mercado dessa tecnologia? Quais as perspectiva para seu futuro?
- **Disponibilidade em outras plataformas** - é possível utilizar tal tecnologia em plataformas não Microsoft?

A tabela 3.1 explicita tal comparação.

	DDE	OPC
Padronização	Não	Sim
Interoperabilidade	Limitado aos padrões DDE (FastDDe,AdvancedDDE).	Sim.
Performace	DDE é baseado em troca de mensagens. Isso é lento. A eficiência melhora se colocarmos vários dados em uma mesma mensagem.	Muito alta.
Acesso remoto	Sim (NetDDE).	Sim (DCOM).

Interação	Não.	SIM. OPC define vários métodos que podem ser usados para a troca de informações adicionais entre cliente e servidor.
Complexidade	Vários *toolkits* são disponibilizados pelos fabricantes. Programar DDE sem a ajuda dos *toolkits* é bastante difícil.	Basta saber programar objetos DCOM. Existem *toolkits* para diminuir o tempo dispensado.
Parcela de Mercado	Servidores DDE irão desaparecer no futuro.	As soluções OPC provavelmente dominarão o mercado.
Outras plataformas	Apenas para o sistema operacional Windows.	OPC é limitado apenas onde o DCOM está disponível. Primariamente foi desenvolvido apenas para o Windows porém existem especificações(**OPC and XML**) para a utilização do OPC em outros sistemas operacionais.

Tabela 3.1: Comparação entre as tecnologias

Fica fácil verificar que o OPC é hoje a forma mais adequada para aquisição de dados, seja de sistemas supervisórios ou diretamente dos dispositivos. Seu desempenho é superior ao do **DDE**, além de ter se tornado um padrão altamente aceito pela indústria (Iwanitz e Lange, 2001). O desempenho, a padronização, o constante desenvolvimento, a integração com a internet e com outros sistemas operacionais fazem hoje do OPC a melhor alternativa para aquisição e controle de dados nos sistemas supervisórios e de gerência de informação.

3.4 OPC DA 2.0

A especificação OPA DA foi a primeira a ser lançada pela **OPC Foundation**. É nela que estão definidas regras para a troca de dados em tempo real entre clientes e servidores OPC. Por essa especificação, são os servidores OPC que se comunicam com os dispositivos e disponibilizam essas informações para os clientes. A forma como os servidores irão adquirir as informações dos dispositivos não é definida pela especificação e sim apenas o padrão de comunicação entre os clientes e servidores.

Como já mostrado no capitulo 2 o OPC utiliza a tecnologia COM/DCOM para comunicação entre aplicações. Essa especificação trata apenas de como deve ser desenvolvidos os componentes COM, tanto do lado do cliente como do lado do servidor, para que a troca de dados possa ser feito independente de quem implementou os aplicativos. Esses componentes COM possuem interfaces e estas possuem algumas pequenas particularidades que vale a pena salienta-las:

- O nome de uma interface sempre começa com a letra **I** maiúscula.
- Se as interfaces não foram definidas pela Microsoft, a primeira letra é seguida

por uma abreviação que descreve o campo de aplicação. Para o caso das especificações OPC esta abreviação é OPC.

De forma geral o funcionamento dos servidores OPC consiste em fazer a aquisição dos dados e deixar essa informação acessível para os clientes OPC que nele se conectarem. Esta disponibilização se dá através do que a especificação OPC chama de Itens. Um item é um dado que pode ser uma temperatura, pressão, estado de uma válvula, etc, disponível para ser lido ou escrito por um cliente OPC.

Um cliente OPC é uma aplicação a qual irá se conectar com um servidor para interagir com os itens disponibilizados. Para uma melhor organização e melhoria na transmissão de informações entre servidores/clientes, os clientes OPC devem criar grupos de itens com características semelhantes. Os grupos além de definir as principais características de leitura dos itens (taxa de atualização,estado ativo ou inativo, banda morta, leitura síncrona ou assíncrona) são usados para estruturar-los. Isso pode ser feito por aspectos lógicos(variáveis de processo com comportamento semelhantes) ou de acordo com a vontade do usuário. Vale salientar que o grupo é criado no lado do servidor, porém por requisição de um cliente. Os grupos de um determinado cliente não ficam visíveis a outro, a não ser que no ato da criação o cliente especifique ser um grupo público.

Nesse contexto, podemos organizar os objetos OPC através de uma hierarquia. A figura 3.4 nos ilustra como estão organizados esses objetos.

3.4.1 Namespace

O *namespace* é o conjunto de dados (itens) disponibilizados pelo servidor, vide figura 3.4.O *namespace* pode ter duas formas de organização: *flat* e *hierarchical*.

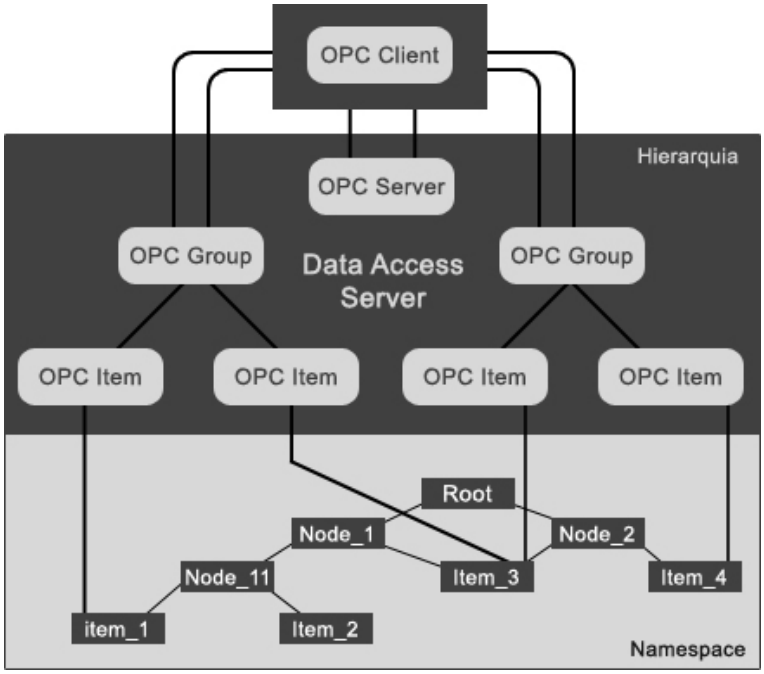


Figura 3.4: O *namespace* e a hierarquia de objetos

Em um *flat namespace* todos os itens estão localizados em um mesmo nível, ou seja não existem nós. Esta é a forma mais simples de disponibilizar os itens, porém não existirá nenhuma organização hierárquica entre eles. Para melhor entendimento imaginemos que determinado servidor OPC disponibilizará informações de processo de diversos poços de petróleo. Todos os itens (temperatura,pressão,vazão) de todos os poços serão disponibilizados em um mesmo nível cabendo ao cliente saber quais itens são de um mesmo poço.

Já no *hierarchical namespace* os itens estão disponibilizados em estrutura de árvore com qualquer profundidade, figura 3.4. Neste caso, retomando nosso exemplo, cada nó da árvore poderá representar um poço que conterá, por sua vez, os itens correlacionados. Esta sem dúvidas é uma maneira bem mais interessante de disponibilizar os itens disponíveis, cabendo ao desenvolvedor do servidor optar por implementar um

dos dois métodos especificados.

3.4.2 Formato do Dado OPC

O formato no qual os dados são trocados entre cliente e servidores OPC também é estabelecido pela especificação. Todo troca de dados deve seguir o seguinte formato:

- **Valor do dado** - Espaço reservado para a disponibilização do valor do item correspondente. Todo valor de item é do tipo *VARIANT*, definidos pelo objeto **COM**.
- **Time Stamp** - Esta informação corresponde basicamente ao instante em que o dado foi lido ou escrito. Esta informação é fornecida pelo servidor através da leitura do *time stamp* dos dispositivos de campo ou por geração interna.
- **Informação de Estado** - São reservados 2 *bytes* para codificação do estado do dado fornecido pelo servidor. Por enquanto apenas o uso do *byte* menos significativo foi especificado. Dois bits definem a qualidade do dado que pode ser:
  - **Good** - Dado válido.
  - **Bad** - No caso de perda do *link*, de comunicação com o dispositivo de campo, por exemplo.
  - **Uncertain** - No caso de existir o *link* de comunicação, mas o dispositivo de campo estiver fora de operação.

Quatro *bits* fornecem um detalhamento do estado apresentado, tais como **Not Connected** e **Last Usable Value**. Os últimos dois *bits* podem conter dados de diagnóstico no caso de falha de um sensor, por exemplo.

3.4.3 Tipo de Comunicação

Existem 4 tipos de comunicação para que o cliente obtenha um determinado dado: leitura síncrona, assíncrona, *refresh* e *subscription*, mostrados na tabela 3.2. Note

que os valores podem ser obtidos tanto de um *cache*(**CACHE**), que são dados de processo localizados no interior do servidor OPC, quanto diretamente da fonte de dados de processo (**DEVICE**).

	CACHE	**DEVICE**
Leitura Síncrona	**X**	**X**
Leitura Assíncrona	**X**	**X**
Refresh	**X**	**X**
Subscription		

Tabela 3.2: Tipo de troca de dados permitida

Com a leitura síncrona o cliente chama o método e espera pelo seu retorno. Chamadas síncronas devem somente ser usadas se os dados de processo forem obtidos rapidamente (valores disponíveis no próprio PC, por exemplo), caso contrário o cliente fica travado por um longo tempo.

Nas leituras assíncronas, o cliente chama o método no servidor e fica liberado imediatamente para continuar sua execução normal. Após um certo intervalo, que depende de como o dado é aquisitado, o cliente obtém o valor. Leituras assíncronas devem ser utilizados se a obtenção dos dados no servidor demora muito, como por exemplo, os valores obtidos diretamente nos dispositivos de campo.

O *Refresh*, é o tipo de aquisição que faz com que o cliente leia todos os itens ativos de um determinado grupo, que por sua vez também deverá se encontrar ativo.

Os três tipos de aquisição de dados descritos até o momento são feitos por iniciativa do cliente. Porém nem sempre isso é uma alternativa eficiente, já que o cliente obterá valores independentes deles terem sofridos qualquer alteração. Para suprir tal

necessidade é que existe o tipo de troca de informação chamado *subscription*. Nesta forma de aquisição, o servidor faz leituras em ciclos determinadas pela taxa de atualização e os envia para o cliente caso a mudança de valor ou estado ocorrer. O cliente por sua vez, tem a opção de definir a taxa de atualização com que os dados serão varridos pelo servidor, bem como, a sensibilidade de mudança desses valores para que os mesmos sejam enviados (banda morta).

3.4.4 OPCServer

Esta e as próximas sessões pretendem dar uma visão geral do que pela especificação **OPC DA 2.0** fica determinado. Após a leitura destas sessões o leitor deve ter um conhecimento geral das funcionalidades disponibilizadas do padrão OPC. As assinaturas dos métodos não estão disponibilizadas por que não é a intenção deste documento ser um manual geral de implementação. Porém o próximo capitulo se dedica a detalhar uma implementação realizada em laboratório.

Os objetos OPC são organizados hierarquicamente, figura 3.5. No topo desta hierarquia está o objeto **OPCServer**. Este objeto é criado quando o servidor entra em execução e um ponteiro para uma de suas interfaces é retornado para o cliente quando o mesmo se conecta ao servidor, através da função *CoCreateInstance*, detalhada no capítulo 2. Após o conhecimento deste ponteiro o cliente poderá utilizar as diversas funcionalidades da interface solicitada ou fazer um requisição para uma nova interface pelo *QueryInterface*. A figura 3.5 mostra como esta organizado o objeto **OPCServer**.

Portanto o objeto **OPCServer**, segundo a especificação, deve disponibilizar obrigatoriamente todas as interfaces da figura 3.5, com exceção das entre colchetes que

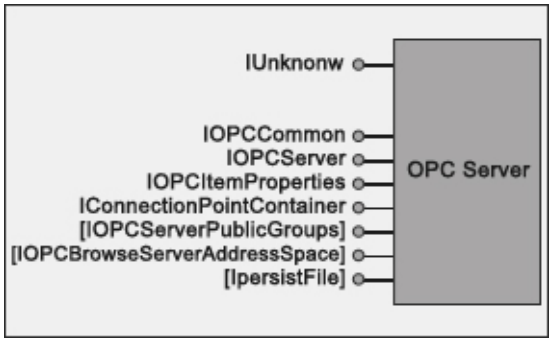


Figura 3.5: Interfaces do Objeto **OPCServer**

são de implementação opcional. A tabela 3.5 mostra informações sobre funcionalidades destas interfaces.

Interface	**Descrição Resumida**
IUnknown	Interface default. Serve para indagar sobre as demais interfaces do componente.
IOPCCommon	Define e ler o **LocaleID**. Este é importante para que os códigos de erro sejam disponibilizados no idioma do usuário.
IOPCServer	Principal Interface. Adiciona e remove grupos ao servidor.
IOPCItemProperties	Retorna as propriedades disponíveis de um item.
IConnectionPointContainer	Permite ao cliente criar um *sink* para receber eventos de um ponto de conexão de um servidor OPC.
[IOPCServerPublicGroups]	Propicia acesso de um cliente a um grupo público.

[IOPCBrowseServerAddressSpace]	Permite realizar um *browsing* dos itens disponíveis no servidor.
[IpersistFile]	Interface Microsoft para salvar as configuração do espaço de memória do servidor.

Tabela 3.3: Descrição das Interfaces do **OPCServer**

O conhecimento das interfaces e métodos disponibilizados pela especificação é um primeiro passo para saber do que o padrão OPC é capaz. O conhecimento de tais funcionalidades é imprescindível na elaboração de qualquer projeto que utilize a comunicação via OPC. Abaixo segue a listagem e descrição dos métodos do objeto **OPCServer**.

- **IUnknown** - Esta interface é obrigatória para todo componente COM. As características dessa interface foram mostradas no capítulo 2.
- **IOPCCommon** - Outros servidores OPC, como por exemplo o de alarmes e eventos, compartilham esta interface. Ela permite definir e localizar um **LocaleID** que será empregado numa sessão particular entre o cliente e o servidor. Isto é, assim como na definição de grupos, as ações realizadas por um cliente não afetam outros clientes. Veja a lista de métodos publicados por esta interface:
  - **SetLocaleID** - Indica o *default* **LocaleID** para esta sessão cleinte/servidor. Este **LocaleID** será usado pelo método **GetErrorString** desta interface.
  - **GetLocaleID** - Retorna o *default* **LocaleID** oara esta seção cliente/servidor.
  - **QueryAvaiableLocaleIDs** - Retorna os **LocaleIDs** disponíveis.
  - **GetErrorString** - Retorna uma *string* de erro para um código de erro específico do servidor.
  - **SetClientName** - Permite que o cliente opcionalmente registre um nome junto ao servidor. Esta propriedade é incluída para propósito de depuração.

- **IOPCServer** - Esta é a principal interface do objeto **OPCServer**. Abaixo os métodos publicados:
  - **AddGroup** - Adiciona um grupo no servidor.
  - **GetErrorString** - Retorna uma *string* de erro para um código de erro específico do servidor.
  - **GetGroupByName** - Dado o nome do grupo provado (criado mais cedo pelo próprio cliente, retorna um ponteiro adicional para a sua interface.
  - **GetStatus** - Retorna a informação corrente de estado do servidor.
  - **RemoveGroup** -Remove um Grupo.
  - **CreateGroupEnumerator** - Cria vários enumeradores para os grupos providos pelo servidor.
- **IOPCItemProperties** -Esta interface pode ser utilizada por clientes para retornar as propriedades disponíveis de um item e ler o valor corrente desta propriedade. Veja a lista de métodos:
  - **QueryAvaiableProperties** - Retorna uma lista de identificadores e descrições para as propriedades disponíveis do item passado.
  - **GetItemProperties** - Retorna o valor corrente das propriedades passadas por parâmetro.
  - **LookupItemIDs** - Retorna o *fully qualifield ItemId* das propriedades passadas por parâmetro.
- **IConnectionPointContainer** - Esta interface provê acesso ao **ConnectionPoint** para o **IOPCShutdown**. Isto permitirá ao cliente ser notificado em casa de um evento no servidor. Segue os métodos publicados:
  - **EnumConnectionPoints** - Cria um enumerados para os Connection Points entre cliente e servidor OPC Group. Este enumerador deve incluir **IOPCShutdown**.
  - **FindConnectionPoint** - Acha um Connection Point particular entre o cliente e o OPC Server.
- **[IOPCServerPublicGroups]** - Esta interface provê um mecanismo conveniente para que os clientes e servidores compartilhem grupos públicos. Lista de métodos publicados por esta iterface:
  - **GetPublicGroupByName** - Conecta um cliente em um grupo através do retorno de um ponteiro para a sua interface.

  - **RemovePublicGroup** - Remove um grupo público.

- **[IOPCBrowseServerAddressSpace]** - Esta interface provê uma forma dos clientes realizarem um browsing dos itens disponíveis no servidor. Segue a lista de métodos publicados:

  - **QueryOrganization** - Indica se o *namespace* é do tipo *flat* ou *hierarchial*.
  - **ChangeBrowsePosition** - Move para cima, para baixo ou para um determinado ponto do *namespace*.
  - **BrowseOPCItemIDs** - Retorna um enumerator que contém uma lista de **ItemIds** que vai de acordo com os parâmetros informados.
  - **GetItemID** -Monta o *fully qualifield Itemid* a partir dos **ItemIds** retornados no método **BrowseOPCItemIDs**.
  - **BrowseAccessPaths** - retorna os **AccessPaths** disponíveis para um dado item.

- **[IpersistFile]** - Esta interface é propiciada pela Microsoft. É uma interface opcional e permite que um cliente salve ou carregue uma configuração de um servidor. A operação é feita em um arquivo qualquer especificado. Esta interface salva os parâmetros do cliente como definição de grupo e item.

  - **IsDirty** - Retorna se houve alguma mudança de configuração por parte de algum cliente desde a ultima operação de salvamento.
  - **Load** - Carrega as configuração a partir do arquivo especificado.
  - **Save** - Salva a configuração corrente.
  - **SaveCompleted** - Notifica o objeto que ele pode voltar para o modo de operação normal.
  - **GetCurFileName** - O servidor retorna o nome do arquivo de onde foi carregada a configuração corrente.

3.4.5 OPCGroup

Se descermos na hierarquia OPC, o próximo objeto trata-se do **OPCGroup**. Com este objeto, dentre outras funcionalidades, podemos criar, alterar e remover grupos no servidor OPC.

Os objetos OPCGroups são criados automaticamente quando utilizamos o método **AddGroup** da interface IOPCServer. Este método além de criar o objeto, retorna um ponteiro apontando para tal. Abaixo está ilustrada uma figura 3.6 que relaciona as interfaces disponibilizadas por tal objeto.

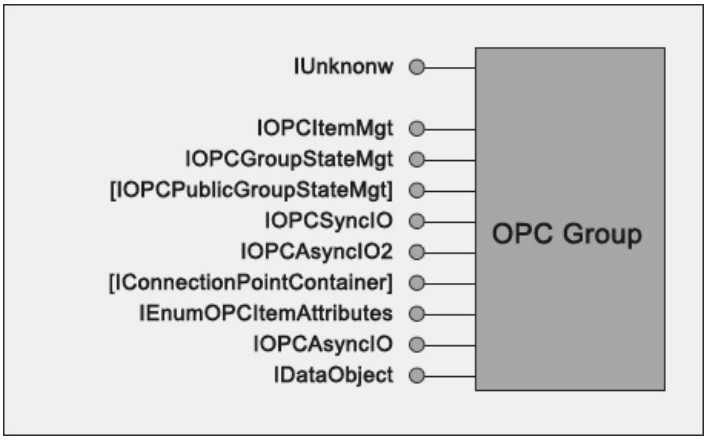


Figura 3.6: Interfaces do Objeto **OPCGroup**

Portanto o objeto OPCGroup, segundo a especificação, deve disponibilizar obrigatoriamente todas as interfaces da figura 3.6, com exceção das entre colchetes que são de implementação opcional. A tabela 3.4 mostra informações sobre funcionalidades destas interfaces.

Interface	**Descrição Resumida**
IUnknown	Interface default. Serve para indagar sobre as demais interfaces do componente.
IOPCItemMgt	Adição, remoção e alteração de itens no grupo.
IOPCGroupStateMgt	Administração do grupo e de suas propriedades pelo cliente.

[IOPCPublicGroupStateMgt]	Converte um grupo privado em público para grupos que suportam esta funcionalidade.
IOPCSyncIO	Função de leitura e escrita síncronas no servidor.
IOPCAsyncIO2	Funções de leitura e escrita assíncronas no servidor.
[IConnectionPointContainer]	Define coletores para receber notificações de eventos de grupo (leituras assíncronas) do servidor.
IOPCAsyncIO	Obsoleta. Era usada para função de leitura e escrita assíncronas. Foi substituida pela **IOPCAsyncIO2**.
IDataObject	Obsoleta. Interface callback utilizada pelo servidor para responder às requisições assíncronas do cliente. Foi substituída pela interface **IConnectionPointContainer**.
IEnumOPCItemAttributes	Permite ao cliente achar o conteúdo (itens) de um grupo e seus atributos. Esta interface só é retornada pela chamada de IOPCItemMgt::CreateEnumerator. Ela não está disponível através de **QueryInterface**.

Tabela 3.4: Descrição das Interfaces do **OPCGroup**

A organização em grupos dos diversos itens disponibilizados pelo servidor é papel do cliente. E para tal o servidor disponibiliza o objeto **OPCGroup** com diversas interfaces e métodos, cuja descrições podem ser visualizadas abaixo.

- **IOPCItemMgt** - Esta interface permite que um cliente adicione, remova e controle o compportamento de itens em um grupo. Segue a lista de métodos:
  - **AddItems** - Adiciona um ou mais itens em um grupo.
  - **ValidateItems** - Determina se um ítem é válido para que seja adicionado sem erros.
  - **RemoveItems** - Remove itens de um grupo.
  - **SetActiveState** - Muda um ou mais itens de um grupo para ativo ou inativo.
  - **SetClientHandles** - Modifica o **ClientHandle** de um ou mais itens de um grupo.
  - **SetDatatypes** - Modifica o **RequestDatatype** para um ou mais itens de um grupo.
  - **CreateEnumerator** - Cria uma lista (enumerador) para os itens de um grupo.
- **IOPCGroupStateMgt** - Esta interface permite que um cliente gerencie todo o estado do grupo.
  - **GetState** - Obtém o estado corrente de um grupo.
  - **SetState** - Realiza modificações no estado do grupo, como por exemplo **UpdateRate** e **ActiveState**.
  - **SetName** - Modifica o nome do grupo. O nome deve ser único.
  - **CloneGroup** - Cria uma cópia de um grupo com um nome único.
- **[IOPCPublicGroupStateMgt]** - Esta é uma interface opcional que é usada para converter um grupo privado em um grupo público. Métodos publicados:
  - **GetState** - Indica se um grupo é público ou não.
  - **MoveToPublic** - Converte um grupo privado em público.

- **IOPCSyncIO** - Esta interface permite que um cliente realize operações de leitura e escrita síncrona no servidor. As operações serão concluídas antes do retorno do método.Métodos publicados:
  - **Read** - Realiza a leitura de um ou mais itens OPC em um grupo.
  - **Write** - Escreve valores em um ou mais itens de um grupo.
- **IOPCAsyncIO2** - Esta interface permite fazer leituras, escritas ou *refresh* assíncronos, isto é, após requisitado pelo cliente o método retorna imediatamente liberando-o para sua execução. Quando um determinado item sofre uma variação no seu valor acima da da sensibilidade especificada para aquele grupo(banda morta) ou a qualidade do estado muda, o servidor chama um método no cliente (onDataChange) através da interface **IOPCDataCallback** para avisar do novo valor relevante recebido. Métodos Publicados:
  - **Read** - Lê um ou mais itens em um grupo.
  - **Write** - Escreve um ou mais itens em um grupo.
  - **Refresh2** - Força uma *callback* para **IOPCDataCallback::OnDataChange** para todos osítens ativos de um grupo (tendo eles mudado ou não). Itens inativos não são inclu idos na *callback*.
  - **Cancel2** - Requisita ao servidor o cancelamento de uma transação enviada.
  - **SetEnable** - Controla a operação do **OnDataChange**. Basicamente, definindo **Enable**para **Falso**, desabilita qualquer *callback* **OnDataChange** com o identificador de transação igual a zero (aqueles que não são o resultado de um *refresh*)
  - **GetEnable** - Retorna o estado definido pelo **SetEnable**.
- **IConnectionPointContainer** - Esta interface substitui a IDataObject lançada na versão anterior da especificação. Sua função é conectar o método de *callback* do cliente com o servidor, através de Connection Points. Métodos Publicados:
  - **EnumConnectionPointContainer** - Cria uma enumeração para os Connection Points suportados entre o grupo OPC e o cliente.
  - **FindConnectionPoint** - Encontra um Connection Point entre grupo OPC e o cliente.
- **IOPCAsyncIO** - Atualmente Obsoleta. Foi substituída pela interface IOPCAsyncIO2, portanto sua descrição foi considerada desnecessária.

- **IDataObject** - Atualmente Obsoleta. Foi substituída pela interface IConnectionPointContainer, portanto sua descrição foi considerada desnecessária.
- **IEnumOPCItemAttributes** - Permite ao cliente achar conteúdo (itens) de um grupo e seus atributos. Estas informações foram definidas pelo cliente quando o mesmo chamou **AddItem**. Esta interface só é retornada pela chamada de **OPCItemMgt::CreateEnumerator**. Ela não está disponível através de **QueryInterface**.Métodos Publicados:
  - **Next** - Busca os próximos *celt* itens do grupo.
  - **Skip** - Salta sobre o próximo *celt* atributos.
  - **Reset** - Reseta o enumerador de volta ao primeiro item.
  - **Clone** - Cria uma segunda cópia do enumerador. O novo enumerador estará no mesmo estado do enumerador corrente.

3.4.6 OPClient

Este é o único objeto que deve ser implementado pelo cliente OPC. É através das interfaces deste objeto que o servidor se comunica com o cliente, seja para enviar um dado (no caso de leitura assíncrona), ou seja, para avisar ao cliente quando o servidor estiver em processo de desligamento.

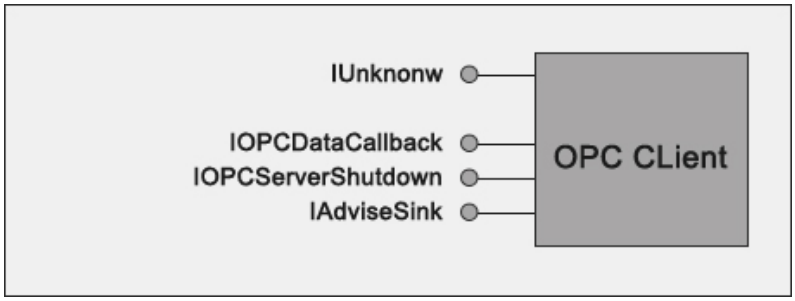


Figura 3.7: Interfaces do Objeto **OPCClient**

Portanto o objeto **OPCClient**, segundo a especificação, deve disponibilizar obrigatoriamente todas as interfaces da figura 3.7. A interface **IAdviseSink** está obsoleta

para a comunicação OPC. Na DA 2.0 ela foi substituída pela **IOPCDataCallback**, porém sua implementação é obrigatória para manter a compatibilidade entre clientes e servidores.

Interface	**Descrição Resumida**
IUnknown	Interface default. Serve para indagar sobre as demais interfaces do componente.
IOPCDataCallBack	Seus métodos são chamados para tratar eventos assíncronos no servidor: leitura, escrita e *refresh*.
IOPCServerShutdown	Utilizado para o cliente liberar suas interfaces, quando o servidor está em processo de desligamento.
IdviseSink	Equivalente a IOPCDataCallback.

Tabela 3.5: Descrição das Interfaces do **OPCClient**

Todos os métodos desta interface são de implementação obrigatória pelo cliente. O servidor ao acusar a variação significativa (maior que a banda morta) de um ou mais itens ativos, irá chamar o método *OnDataChange*, passando os itens alterados como parâmetro. Cabe ao cliente saber o que fazer com este dado. Além dessa funcionalidades existem métodos para quando a operação de escrita for completada, para avisar ao cliente quando o servidor entrar em processo de desligamento, e outras funcionalidades como detalhado a seguir.

- **IOPCDataCallCack** - Implementada pelo cliente e utilizada pelo servidor,

suas funcionalidades consistem em dar um feed-back ao cliente, do termino de uma operação de leitura ou escrita.Métodos Publicados:

  - **OnDataChange** - Gerencia notificações do objeto **OPCGroup** com respeito a modificações do estado dos itens.
  - **OnReadComplete** - Gerencia notificações do objeto **OPCGroup** quando uma leitura assíncrona é completada.
  - **OnWriteComplete** - Gerencia notificações do objeto **OPCGroup** quando uma escrita assíncrona é completada.
  - **OnCancelComplete** - Gerencia notificações do objeto **OPCGroup** quando um pedido de cancelamento é completado.

- **IOPCServerShutdown** - O método **ShutdownRequest** desta interface será chamado quando o servidor entrar em processo de desligamento. O cliente deve liberar todas suas conexões e interfaces deste servidor. Método publicado:

  - **ShutdownRequest** - Nesse método, o servidor requisita ao cliente que ele se desconecte. O cliente deve dar um **UnAdvise** em todas suas conexões, remover todos os grupos e liberar todas interfaces.

- **IAdviseSink** - Está atualmente obsoleta. É equivalente a interface **IOPCDataCallback**. Somente o método **OnDataChange** está disponível nessa in terface.

  - **OnDataChange** - Método fornecido pelo cliente para tratar de notificações do grupo OPC para mudança de dados baseados em exceções: leit uras assíncronas, escrita assíncrona completada e *refresh*.

Capítulo 4

Implementação de um Cliente OPC

4.1 Considerações Gerais

Este capítulo será totalmente dedicado a mostrar aspectos da implementação de um cliente OPC genérico com funcionalidades básicas de leitura de dados e gerenciamento de grupos. O objetivo do desenvolvimento deste cliente é integra-lo como uma forma alternativa de aquisição de dados para o software de gerência de informação na área petroquímica chamado de **Gerinf**.

O padrão OPC é totalmente baseado na tecnologia **COM/DCOM** da Microsoft. Esta tecnologia pode ser implementada em qualquer linguagem de programação que suporte funções e possuam ponteiros para ela, como vimos no capítulo 2, conseqüentemente, OPC também pode ser implementado em qualquer uma dessas linguagens. Por questões de desempenho, documentação e facilidade de integração com o atual sistema, a linguagem **C++** foi a escolhida para o desenvolvimento do cliente em questão.

O software foi totalmente desenvolvido através do ambiente de programação **Borland C++ Builder versão 6.0**. Tal escolha se deveu principalmente ao fato de ser um ambiente com vários recursos para a construção de uma interface amigável, além da documentação e experiência profissional nesse ambiente do desenvolvedor.

4.2 Interface e funcionamento

A interface é o ambiente em que o usuário do sistema interagirá diretamente com o software. Uma boa interface deve facilitar ao máximo esta interação, fazendo com que o usuário, naturalmente, siga um caminho para a utilização correta do sistema. A figura 4.3 mostra o ambiente que o usuário encontrará no ato da execução do aplicativo.

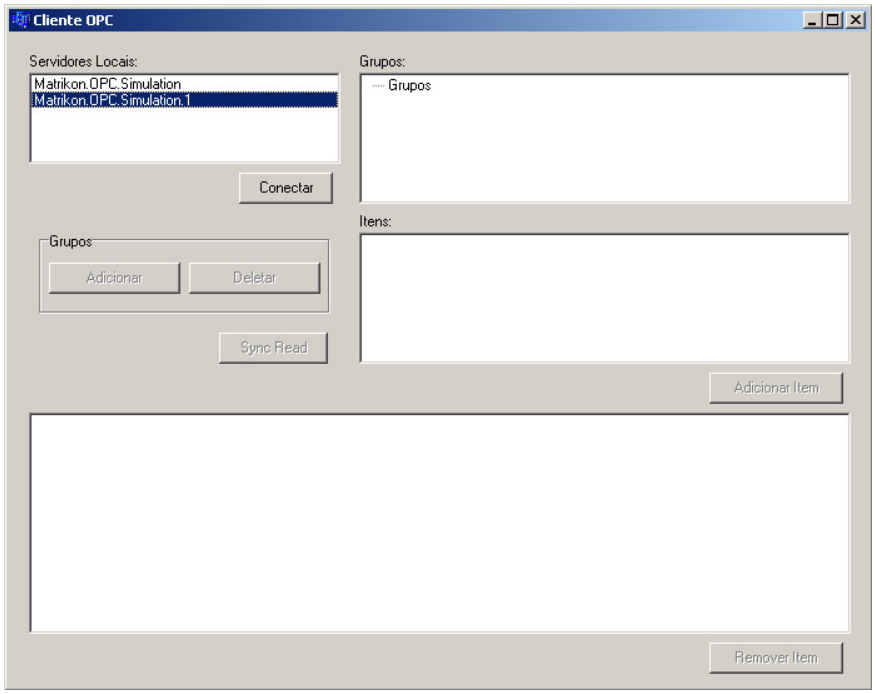


Figura 4.1: Interface inicial do sistema

Após visualizar esta figura fica fácil perceber que o cliente tem apenas um caminho a seguir: o de se conectar com um dos servidores disponíveis, listados no canto superior esquerdo. Após feita a conexão uma gama de novas possibilidades são habilitadas e os itens disponíveis no servidor são lidos e listados no espaço designado por "**Itens**".

De acordo com a especificação OPC os itens de características semelhantes devem ser organizados através de grupos, cabendo ao usuário criar e organizar tais grupos de itens. Para tal basta pressionar o botão "**Adicionar**" localizado na região designada de "**Grupos**". Feito isso, uma nova janela deverá aparecer, como a mostrada na figura 4.2.

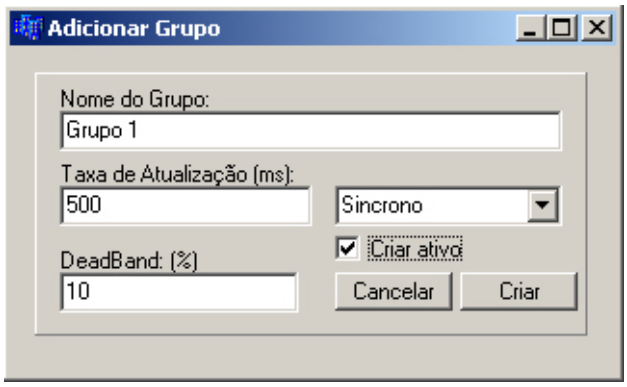


Figura 4.2: Interface de adição de Grupos

Após preencher as informações de nome, taxa de atualização e *Dead Band* o usuário deverá escolher entre o modo síncrono ou assíncrono para a aquisição de dados e ainda selecionar se o grupo será criado ativo ou não. Passado essa etapa basta clicar no botão "**Criar**". A janela deverá sumir automaticamente e o grupo criado poderá ser visualizado no canto superior direito na área destinada aos grupos.

A próxima etapa consiste em adicionar os itens listados em um dos grupos cadastrados previamente. Para isso basta selecionar o grupo que deseja adicionar itens e selecionar simultaneamente o ítem a ser adicionado. Após feitas as seleções basta

clicar no botão "**Adicionar Item**", que só estará habilitado se um item e um grupo estiverem selecionados. Logo após pressionado o botão para adicionar um item, uma nova linha na tabela de aquisição de dados deverá ser incluída, e além disso, o ítem poderá ser visualizado no canto superior direito organizado hierarquicamente com os grupos.

Concluídas as etapas descritas acima terá inicio o processo de aquisição de dados. Os resultados das aquisições serão mostradas na tabela de aquisições, localizada na parte inferior da interface. Cada linha da tabela contém cinco colunas onde serão informadas, além do nome do item e do último valor recebido, o instante que se deu a aquisição, a qualidade do dado e o grupo a que pertence o item em questão.

A figura 4.3 abaixo ilustra a interface após a inclusão de dois grupos(Grupo 1 e Grupo 2), onde primeiro contém três itens adicionados.

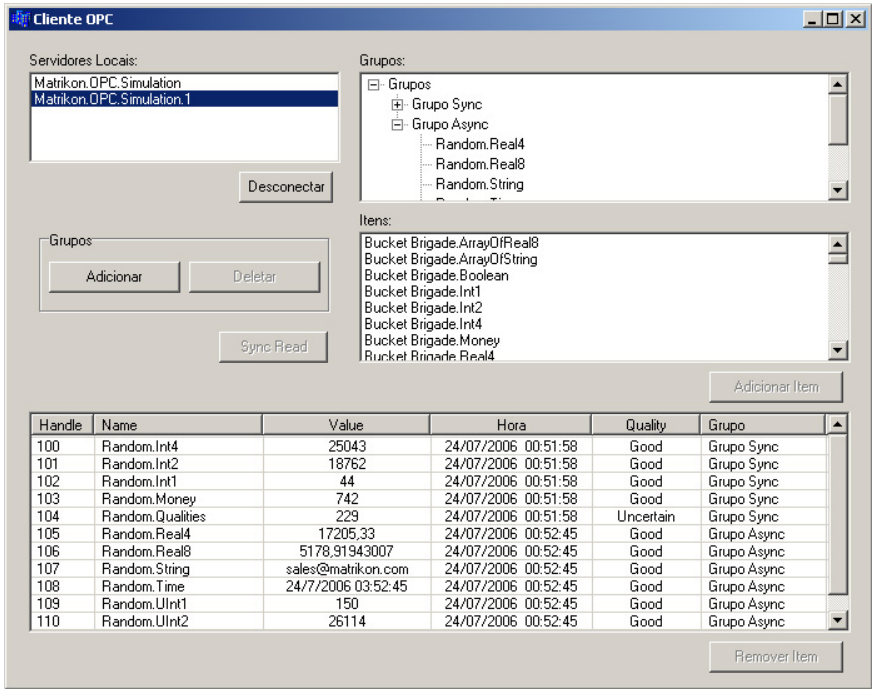


Figura 4.3: Interface após funcionamento

4.3 Arquitetura

A arquitetura do sistema diz respeito a organização interna utilizada para a implementação do mesmo. Para facilitar a reusabilidade o programa foi desenvolvido utilizando o conceito de orientação a objeto.

A orientação a objeto é um paradigma de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas objetos. Na programação orientada a objetos, implementa-se um conjunto de classes que definem os objetos presentes no sistema de software. Cada classe determina o comportamento (definidos nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos. (Wikipédia, 2006).

O sistema em questão consiste de duas classes. Uma delas é a classe destinada a interface do sistema, definida pelo próprio ambiente de desenvolvimento **C++ builder**, nomeada de **TmainForm**. A outra consiste na classe que implementa o padrão OPC, chamada de **Com**.

Através desta arquitetura as duas classes funcionam interligadamente, porém independentes. Ao ser identificado um evento(por exemplo de *click* em algum botão) o objeto da classe **TmainForm** faz uma requisição ao objeto da classe **Com**, que por sua vez retorna um resultado à interface.

Esta forma de implementação permite com que a classe **Com** seja facilmente reutilizada por qualquer aplicativo que deseje implementar funcionalidades básicas da especificação **OPC DA 2.0**. Porém para isso o desenvolvedor deverá ter em mãos o detalhamento da forma de utilização das funcionalidades (métodos) da classe, o qual será apresentado na seção seguinte.

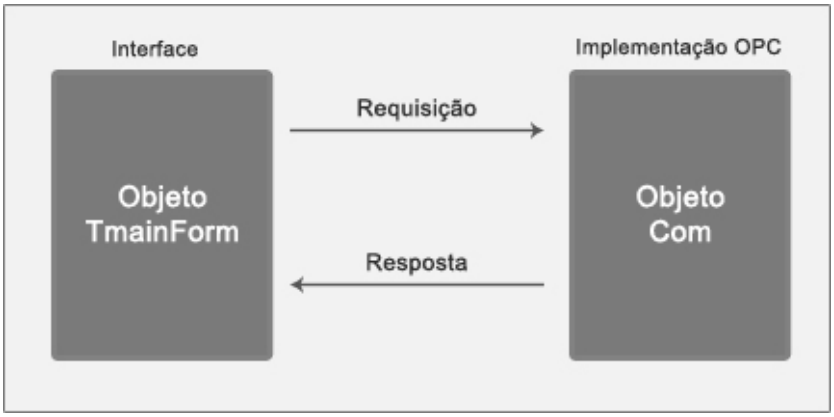


Figura 4.4: Esquema da arquitetura utilizada

4.4 A classe Com

A classe **Com** é a responsável pela implementação básica de funcionalidades da especificação **OPC DA 2.0**. Ela é composta por 18 métodos públicos que vão desde a inicialização da biblioteca COM até leituras síncronas e assíncronas de grupos de itens em servidores OPC.

O primeiro passo para a utilização desta classe consiste em definir a interface **CCreatableDataCallbackSink** a qual interligará o servidor ao método **OnDataChange** (que também deverá ser implementado) para permitir que o cliente seja avisado quando um novo dado obtido a partir de uma leitura assíncrona esteja disponível.

Para definir uma interface do tipo CCreatableDataCallbackSink, basta declara-la:

```
CCreatableDataCallbackSink m_DataCallbackSink;
```

Feito essa declaração o proximo passo consiste em implementar o método **OnDataChange** que possui a seguinte assinatura:

```
void OnDataChange(
   /* [in] */ DWORD dwTransid,
   /* [in] */ OPCHANDLE hGroup,
   /* [in] */ HRESULT hrMasterquality,
   /* [in] */ HRESULT hrMastererror,
   /* [in] */ DWORD dwCount,
   /* [size_is][in] */ OPCHANDLE __RPC_FAR *phClientItems,
   /* [size_is][in] */ VARIANT __RPC_FAR *pvValues,
   /* [size_is][in] */ WORD __RPC_FAR *pwQualities,
   /* [size_is][in] */ FILETIME __RPC_FAR *pftTimeStamps,
   /* [size_is][in] */ HRESULT __RPC_FAR *pErrors);
```

O próximo e último passo consiste em passar o endereço do método **OnDataChange** para a interface **CCreatableDataCallbackSink**, através do seguinte comando:

```
m_DataCallbackSink.EvDataChange = OnDataChange;
```

Feito isso, o cliente ficará interligado ao servidor para leituras assíncronas através do método **OnDataChange**. Sempre que um determinado item mudar de estado (qualidade do dado) ou variar seu valor (acima da banda morta), esse método implementado será chamado.

Abaixo serão mostrados a lista dos métodos com uma descrição simplificada, assinatura, argumentos e valores de retorno.

- **Com::startCom**

  Este método é o responsável pela inicialização da biblioteca **COM** e deve ser o primeiro método a ser chamado antes de qualquer outra funcionalidade disponível. Caso o usuário utilize outra funcionalidade sem ter anteriormente invocado o **startCom**, o método utilizado retornará um erro.

  Sua assinatura é:

```
HRESULT startCom();
```

Valores de retorno:

O valor de retorno é do tipo **HRESULT**, descrito no capítulo 2, que por sua vez é definido pela API do Windows. Os valores possíveis são:

Nome	Significado
S_OK	Biblioteca inicializada com sucesso.
S_FALSE	Falha ao tentar inicializar a biblioteca.
RPC_E_CHANGED_MODE	A biblioteca COM já foi inicializada.

- **Com::stopCom**

  Este método deverá ser chamado quando não se deseja utilizar mais nenhuma funcionalidade do objeto. É ele o responsável por desinstalar a biblioteca **COM** e liberar a memória alocada.

  Sua assinatura é:

  ```
  void stopCom();
  ```

- **Com::getServer**

  Este método faz uma varredura no registro do Windows procurando por servidores OPC disponíveis.

  Sua assinatura é:

  ```
  void getServer(string*);
  ```

Será retornado por referência a lista de **ProgIDs** de servidores disponíveis. Para isso, no entanto, deverá ser passado como parâmetro um ponteiro para um *Array* de *strings*.

- **Com::setProgId**

  Uma vez conhecido o **ProgID** do servidor a ser conectado, o método **setProgId** deverá ser chamado. Este método grava o **ProgID** em uma variável interna ao objeto.

  Sua assinatura é:

  ```
  void setProgId(string ProgID);
  ```

- **Com::setClsid**

  Uma vez conhecido o **ClSID** do servidor a ser conectado, o método **setClsid** deverá ser chamado. Este método grava o **ClSID** em uma variável interna ao objeto.

  Sua assinatura é:

  ```
  void setClsid(CLSID);
  ```

- **Com::getProgId**

  Retorna uma *string* que contém o **ProgID** do servidor OPC, caso esta informação esteja disponível. Caso contrário retorna uma *string* vazia.

  Sua assinatura é:

  ```
  string getProgId();
  ```

- **Com::getCLSID**

  Retorna o CLSID do servidor OPC, caso esta informação esteja disponível.

  Sua assinatura é:

  ```
  CLSID getCLSID();
  ```

- **Com::clsfromProg**

  Obtém o **CLSID** do servidor a partir do **ProgID**. O **ProgID** deverá estar na variável interna do objeto (acessada através do método setProgId e getProgId). Os **ProgIDs** disponíveis poderão ser obtidos através da lista retornada pelo método **getServer**, descrito anteriormente.

  Sua assinatura é:

  ```
  HRESULT clsfromProg();
  ```

- **Com::conectServer**

  Este método é o responsável por fazer a conexão com o servidor OPC.

  Sua assinatura é:

  ```
  HRESULT conectServer();
  ```

  Para seu funcionamento o **CLSID** deverá estar gravado na variável interna (acessada através dos métodos **setCLSID**, **getCLSID** ou do **clsfromProg**), caso contrário será retornado um erro.

  Valores de retorno:

Nome	Significado
S_OK	Uma instância do objeto foi criada com sucesso.
REGDB_E_CLASSNOTREG	A classe especificada pelo CLSID não foi encontrada.
CLASS_E_NOAGGREGATION	A classe não pode ser criada como parte de uma agregação.
E_NOINTERFACE	A interface requisitada não está implementada.

- **Com::getItens**

  Este método retorna por referência a lista de todos os itens cadastrados no servidor conectado. Para o seu funcionamento é preciso anteriormente estabelecer uma conexão com o servidor através do método **conectServer**.

  Sua assinatura é:

  ```
  bool Com::getItens(string *listItens);
  ```

  A lista de itens é colocado em um vetor de *strings*, cujo o ponteiro deverá ser passado como parâmetro. A função retorna um valor booleano que se torna **true** se a operação foi realizada com sucesso e **false** caso contrário.

- **Com::addGrupo**

  Adiciona um grupo no servidor OPC.

  Sua assinatura é:

```
HRESULT Com::addGrupo(string name, string taxa, string dead, bool
active, int tipo, CCreatableDataCallbackSink *interface);
```

Devem ser passados a *string* do nome do grupo a ser adicionado, a taxa de leitura em milissegundos, a banda morta(do inglês *deadband*) em porcentagem, o estado do grupo como ativo (**true**) ou inativo (**false**), inteiro representando o tipo de leitura a ser utilizado: assíncrono (0) e síncrono (1) e por fim um ponteiro para a interface CCreatableDataCallbackSink;

Valores de retorno:

Nome	**Significado**
S_OK	Grupo criado com sucesso.
E_FAIL	Falha na operação.
E_OUTOFMEMORY	Memória Insuficiente.
E_INVALIDARG	Argumento inválido.
OPC_E_DUPLICATENAME	Nome do grupo já existente.
OPC_SUNSUPPORTEDRATE	Servidor não suporta a taxa de atualização requisitada.
E_NOINTERFACE	A interface requisitada não está implementada.

- **Com::delGrupo**

  Exclui um grupo, criado anteriormente, do servidor OPC.

  Sua assinatura é:

```
HRESULT Com::delGrupo(string grupo);
```

Deve ser passado como parâmetro uma string com o nome do grupo a ser excluído.

Valores de retorno:

Nome	Significado
S_OK	Grupo criado com sucesso.
E_FAIL	Falha na operação.
E_OUTOFMEMORY	Memória Insuficiente.
E_INVALIDARG	Argumento inválido.
OPC_S_INUSE	Existem referências ao grupo. O servidor irá excluir o grupo automaticamente quando todas as referências ao grupo forem liberadas.

- **Com::addItens**

  Adiciona um determinado ítem em um grupo previamente criado.

  Sua assinatura é:

```
HRESULT Com::addItens(string grupo, int count, string *itens);
```

  Devem ser passados como parâmetros uma *string* contendo o nome do grupo onde os itens serão adicionados, a quantidade de itens e por fim um ponteiro para um array de string com os identificadores de cada item.

  Valores de retorno:

Nome	Significado
S_OK	Grupo criado com sucesso.

E_FAIL	Falha na operação.
E_OUTOFMEMORY	Memória Insuficiente.
E_INVALIDARG	Argumento inválido.
OPC_EPUBLIC	O item não pôde ser adicionado no grupo público.
E_NOINTERFACE	A interface requisitada não está implementada.

- **Com::removeItens**

  Remove um determinado ítem de um grupo.

  Sua assinatura é:

```
HRESULT Com::removeItens(string item, string grupo);
```

  Devem ser passados como parâmetros duas *strings*: a primeira com o identificador do item a ser excluído e a subseqüente com o nome do grupo no qual pertence o item.

  Valores de retorno:

Nome	**Significado**
S_OK	Item removido com sucesso.
E_FAIL	Falha na operação.
E_OUTOFMEMORY	Memória Insuficiente.
E_INVALIDARG	Argumento inválido.
OPC_EPUBLIC	O item não pôde ser removido de um grupo público.
E_NOINTERFACE	A interface requisitada não está implementada.

- **Com::getItensofGroup**

  Este método retorna a lista de identificadores dos itens de um determinado grupo.

  Sua assinatura é:

  ```
  DWORD Com::getItensofGroup(string grupo, DWORD *hserver, DWORD
  *hclient);
  ```

  Como parâmetros, devem ser passados uma string com o nome do grupo e dois ponteiros para arrays de DWORDs, onde serão colocados os identificadores dos itens fornecidos pelo servidor(hserver) e os identificadores dado pelo cliente(hclient). A saída desse método (identificadores do servidor) serve como entrada para os métodos de leitura síncrona.

  Valor de retorno:

  O método retorna um DWORD que contém o número de itens cadastrados no grupo requisitado.

- **Com::OPCSyncIO**

  Este método realiza leitura síncronas de um conjunto de itens.

  Sua assinatura é:

  ```
  HRESULT Com::OPCSyncIO(string grupo, DWORD *m_hItem, DWORD
  tam,string *qld, string *data, VARIANT *dataValues);
  ```

  Como parâmetros para este método devem ser passados uma *string* com o nome do grupo, um array de identificadores de itens a serem lidos (obtido através

do método **getItensofGroup**), um DWORD com o número de itens a serem lidos, um ponteiro para um vetor de strings onde serão colocado a qualidade dos dados recebidos, um ponteiro para um array de strings onde serão colocados o *timestamp* de cada item e por fim um ponteiro para um array de VARIANTs onde serão colocados os valores lidos.

Valores de retorno:

Nome	**Significado**
S_OK	Grupo criado com sucesso.
E_FAIL	Falha na operação.
E_OUTOFMEMORY	Memória Insuficiente.
E_INVALIDARG	Argumento inválido.
S_FALSE	Operação foi concluída porém existem um ou mais erros.

- **Com::connected**

  Retorna o status atual da conexão com o servidor.

```
bool Com::connected();
```

  Valores de retorno: Retorna um booleano que será **true** quando existir uma conexão com o servidor e **false** caso contrário.

- **Com::desconectServer**

  Desconecta o cliente ao servidor, liberando todas as interfaces utilizadas.

```
void Com::desconectServer();
```

Com o conhecimento da assinatura dos métodos e com a posse da classe descrita acima, outros programadores poderão implementar um cliente básico OPC para aquisição de dados. A classe funciona como uma caixa preta onde o conhecimento das estruturas internas da implementação do protocolo OPC não se faz necessário para sua utilização.

4.5 Resultados Obtidos

Após a conclusão da implementação, o cliente desenvolvido, foi submetido a dois tipos de testes: o primeiro consistiu na aquisição de dados a partir do *MatrikonOPC Server for Simulation* e o seguinte a aquisição foi realizada a partir de uma planta real de controle de nível.

O *MatrikonOPC Server for Simulation* é um servidor OPC disponível pela empresa de tecnologia **Matrikon** no qual simula o comportamento de servidores no campo. Para isso os valores de seu itens são gerados aleatoriamente, não existindo portanto, a figura do dispositivo. Por conseguinte, é um servidor desenvolvido exclusivamente para realização de testes.

O teste, para esse servidor, consistiu em criar dez grupos com vinte itens em cada, totalizando 200 itens sendo monitorados. Em apenas um dos grupo foi utilizado o modo de comunicação síncrono. As mais diferentes taxas de atualização e valores de banda morta foram utilizadas na criação destes grupos. O teste se realizou por um período de 12 horas ininterruptas e o cliente não apresentou qualquer instabilidade tanto na comunicação assíncrona como na síncrona durante este tempo. A menor taxa de aquisição da informação conseguida foi de 100 milissegundos, para uma rede local.

A segunda etapa do teste consistiu na aquisição de dados a partir de uma planta de controle de nível na qual está instalada no **LAMP** (Laboratório de Avaliação de Medições em Petróleo) localizado na **UFRN** (Universidade Federal do Rio Grande do Norte). O Objetivo principal deste teste foi avaliar as funcionalidades implementadas numa situação real. Os itens disponíveis (nível, vazão, etc) foram adquiridos sem nenhum problema, provando um dos principais benefícios da utilização do OPC: A padronização da comunicação.

A comunicação assíncrona se mostrou muito mais eficaz para uma situação real, pois evita o desperdício de comunicação no envio de itens que não sofreram alterações significativas, além de liberar o cliente logo após a requisição do dado, evitando travamentos em caso de demora para a resposta do servidor ao pedido.

Capítulo 5

Conclusões

Com a expansão da automação industrial, um grande volume de informações de processos tornam-se disponíveis, porém, se não forem inteligentemente organizadas e processadas, tornam-se inúteis. Os sistemas de gerência de informação visam tornar esse grande volume de dados em informação útil. Um bom gerente além de disponibilizar informações em tempo real do processo deve gravar informações relevantes para serem utilizadas desde o controle de estoque até as estimativas de produções e rendimentos.

O software do **Gerinf** tem essa função. Com ele qualquer usuário (desde que autorizado) pode acessar o sistema e acompanhar em tempo real os processos de produção,gerar relatórios, gráficos, etc. O **Gerinf** não necessita da instalação de softwares específicos, pois todas informações são adquiridas de um servidor e acessadas via *browser*. Esta forma de acesso está sendo muito bem aceita pela indústria e pela Petrobras UN-RN/CE, onde o sistema encontra-se em fase de testes, realizando a gerência do escoamento de petróleo na região de Guamaré-RN. Os dados estão sendo obtidos a partir de dois supervisórios do tipo *inTouch* da própria companhia.

Para a captura dos dados atualmente está sendo utilizado o protocolo DDE. O baixo rendimento e instabilidade, além das tendências mercadológicas foram motivações sine qua non para o desenvolvimento deste trabalho. Apesar do cliente desenvolvido ainda não ter sido incorporado ao sistema do Gerinf, experiências e testes de eficiência e desempenho publicados mostram que o OPC, hoje em dia, é a melhor alternativa a ser utilizada para a aquisição de dados em ambientes industriais.

Como perspectiva, a incorporação do cliente desenvolvido ao sistema de gerência de informação do **GERINF** é sem dúvidas a principal meta ainda a ser cumprida.

Referências Bibliográficas

Bond, M., Haywood, D. e Law, D. (2003). *Aprenda J2EE em 21 dias com EJB, JSP, Servlets, JNDI, JDBC e XML*, Makron Books.

Cerqueira, R. F. G. (2000). *Um Modelo de Composicão Dinâmica entre Sistemas de Componentes de Software*, Tese de Doutorado, PUC - Rio.

Filho, C. S. e Szuter, M. (1993). *Programação Concorrente em ambiente Windows - Uma visão de automação*, Editora da UFMG.

Fonseca, M. O. (2002). Comunicação opc - uma abordagem prática, Universidade Federal de Minas Gerais.

Fonseca, M. O. e Filho, C. S. (2005). *Padrão OPC:Aplicação e Implementação*, ISA - The Instrumentation, Systems, and Automation Society.

Holanda, A. B. (1975). *Novo dicionário da língua portuguesa*, Nova Fronteira.

Husted, T., Dumoulin, C., Franciscus, G. e Winterfeldt, D. (2004). *Struts em Ação*, Ciência Moderna Ltda.

Iwanitz, F. e Lange, J. (2001). *OLE for Process Control*, Hüthig.

Souza, A. J., Bezerra, C. G., de Andrade, W. L. S., Feijó, R. H., Guedes, L. A. H., Leitão, G. B. P., Maitelli, A. L. e de Medeiros, A. A. D. (2005). Gerência de informação de processos industriais: um estudo de caso na produção de petróleo e gás, *VII Simpósio Brasileiro de Automação Inteligente - SBAI*.

Tichy, W. F. (1997). A catalogue of general-purpose software design patterns. IEEE Computer Society.

Wikipédia (2006). Orientação a objeto — wikipédia a enciclopédia livre. Disponível em `http://pt.wikipedia.org/w/index.php?title=Orienta%C3%A7%C3%A3o_a_objeto%&oldid=2387188`.