Monografia de Graduação

Desenvolvimento da Capacidade de Comunicação Segundo o Protocolo OPC (OLE for Process Control) para o Sistema Supervisório SISAL (Sistema Supervisório de Automação da Elevação)

João Teixeira de Carvalho Neto

Natal, Julho de 2010

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

João Teixeira de Carvalho Neto

Desenvolvimento da Capacidade de Comunicação Segundo o Protocolo OPC (OLE for Process Control) para o Sistema Supervisório SISAL (Sistema Supervisório de Automação da Elevação)

Natal – RN

Julho / 2010

João Teixeira de Carvalho Neto

Desenvolvimento da Capacidade de Comunicação Segundo o Protocolo OPC (OLE for Process Control) para o Sistema Supervisório SISAL (Sistema Supervisório de Automação da Elevação)

Trabalho de conclusão de curso submetida como parte dos requisitos para obtenção do grau de Engenheiro de Computação e ao Programa de Recursos Humanos nº 14 da Agência Nacional de Petróleo, Gás Natural e Biocombustíveis da Universidade Federal do Rio Grande do Norte visando a conclusão do trabalho realizado na bolsa de iniciação científica.

Orientador:

Prof. Dr. Adelardo Adelino Dantas de Medeiros

Co-orientador:

Prof. Dr. André Laurindo Maitelli

PROGRAMA DE RECURSOS HUMANOS DA ANP – PRH 14

DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO

CENTRO DE TECNOLOGIA

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

Natal – RN

Julho de 2010

Agradecimentos

Primeiramente a Deus, por iluminar minha mente, ter me concedido saúde, paciência e força para que eu conseguisse realizar meus trabalhos com sucesso e me passado tranqüilidade inclusive nos momentos nos quais pensei que estava tudo perdido.

Aos meus pais, Eugênio Teixeira de Carvalho e Lane Vivian Varela Rodrigues, pelo grande apoio, incentivo e acima de tudo por me passar seu amor, força, carinho e dedicação durante toda esta jornada referente ao meu curso.

À minha namorada, Gleyce Any Freire de Lima, pelo companheirismo, dedicação, por ter me dado força, amor e carinho que foram muito importantes nesta fase de minha vida, e principalmente por nunca ter desistido de alcançar meus sonhos juntamente comigo.

Ao meu orientador, professor Adelardo Adelino Dantas de Medeiros, pela amizade, orientação e dedicação no desenvolvimento deste trabalho.

A todos os meus colegas de curso, que sempre me ajudarem trocando idéias comigo e me apoiando em especial ao meu colega Alan Diego Dantas Protásio por ter me ajudado no desenvolvimento de uma parte do projeto.

Ao Engenheiro de Computação e colega, Gustavo Bezerra Paz Leitão por ter me ajudado na implementação do projeto, ter me concedido o material para estudo e por ter me tirado dúvidas referentes ao trabalho sempre que as tinha.

Aos Engenheiros de Computação Lennedy Campos Soares, João Maria Araújo do Nascimento e Heitor Penalva Gomes pela orientação e pelo suporte ao SISAL.

Ao Programa de Recursos Humanos da Agência Nacional de Petróleo, Gás Natural e Biocombustíveis nº 14 (ANP-PRH-14), pelo apoio financeiro.

Resumo

O emprego de sistemas de supervisão e controle baseados em protocolos de comunicação digital tem crescido nas mais variadas plantas industriais. A diversidade desses protocolos e dos equipamentos baseados nos mesmos, bem como a evolução de suas aplicações na indústria acabou por gerar sistemas de automação de grande complexidade, que requerem utilização de *drivers* para comunicar seus dispositivos com seus sistemas de supervisão e controle. Estes *drivers* são desenvolvidos pelos fabricantes dos equipamentos e por desenvolvedores de *software*. Isso pode causar problemas de interoperabilidade. A necessidade de interoperabilidade entre sub-redes heterogêneas de um mesmo sistema de automação industrial motivou o desenvolvimento do protocolo OPC. O OPC é um protocolo de comunicação entre aplicações que estabelece regras para o acesso de dados em tempo real. O OPC utiliza o modelo cliente/servidor para comunicação de dados onde o servidor faz a aquisição dos dados e deixa a informação acessível para os clientes que nele se conectarem.

O SISAL é um sistema supervisório, pertencente à Petrobras, que tem como objetivo criar uma interface simples para a supervisão de poços de petróleo equipados com sistemas de elevação artificial. O SISAL, atualmente, utiliza as funcionalidades do protocolo OPC através de um servidor OPC, que coleta os dados diretamente do banco de dados do servidor SISAL, gerando um sobrecarregamento de processamento no banco de dados do servidor SISAL. A proposta é desenvolver um servidor OPC que acesse as informações diretamente do servidor SISAL através do protocolo PCI (protocolo de comunicação interna). A confecção desta ferramenta requer o desenvolvimento de uma documentação técnica destinada aos desenvolvedores de *software* que possam fazer futuramente alguma atualização no aplicativo e o desenvolvimento de uma documentação de uso destinada aos usuários finais do aplicativo a aos administradores do sistema que se deseja monitorar através do protocolo OPC.

Abstract

The use of supervisory and control systems based on digital communication protocols has increased in various industrial plants. The diversity of these protocols and equipment based on the same as well as the evolution of its applications in industry ultimately generate automation systems of great complexity, requiring drivers to use their devices to communicate with their supervisory systems and controls. These drivers are developed by equipment manufacturers and software developers. This can cause interoperability problems. The need for interoperability between heterogeneous subnets of the same automated system motivated the development of the OPC protocol. The OPC is a communication protocol between applications that establishes rules for access to data in real time. The OPC uses the client/server model for data communication where the server makes the data acquisition and let the information available for customers who connect it.

The SISAL is a supervisory system, owned by Petrobras, which aims to create a simple interface for monitoring of oil wells equipped with artificial lift. The SISAL currently uses the features of the OPC protocol through an OPC server, which collects data directly from the database server SISAL, creating an overload of processing on the server SISAL's database. The proposal is to develop an OPC server to access information directly from the server SISAL through the ICP protocol (internal communication protocol). The making of this tool requires the development of technical documentation for software developers that can make any update on the future application and development of a documentation of use for end users of the application and for system administrators who wants to monitor through the OPC protocol.

Sumário

Lista de Figuras

Monografia de Graduação – Servidor OPC para o SISAL

Anexo A – Principais Classes, Métodos, Enumerações, Diagramas de Classe e os Possíveis Mecanismos de Atualização de Valores do Softing OPC Toolbox

Anexo B – Documentação Técnica do Servidor OPC para o SISAL

Anexo C – Documentação de Uso do Servidor OPC para o SISAL

Lista de Tabelas

Monografia de Graduação – Servidor OPC para o SISAL

Anexo A – Principais Classes, Métodos, Enumerações, Diagramas de Classe e os Possíveis Mecanismos de Atualização de Valores do Softing OPC Toolbox

Anexo B – Documentação Técnica do Servidor OPC para o SISAL

Lista de Símbolos e Abreviaturas

OLE	Object Link and Embedding
DDE	Dynamic-Data Exchange
COM	Component Object Model
DCOM	Discribuited Component Object Model
PCI	Protocolo de Comunicação Interna
PCM	Protocolo de Comunicação com os Mestres
TCP/IP	Transmission Control Protocol/Internet Protocol
DA	Data Access
AE	Alarm & Events
FIFO	First In First Out
DLL	Dynamic-Link Library
UML	Unified Modeling Language
OPC	OLE for Proccess Control

1. Introdução

A primeira tecnologia para computador que permitiu conversações entre múltiplas aplicações foi a tecnologia DDE (*Dynamic Data Exchange*). Lançada em 1986, juntamente com o lançamento do Windows 2.0 DDE, a tecnologia ficou limitada ao tráfego de uma variável por vez e por cinquenta conexões simultâneas. Surgiram melhorias como *Fast* DDE e *Advance* DDE, no entanto ambas não estão sob domínio público. Clientes e servidores DDE trabalham através de uma conexão de rede ou em uma mesma máquina. A tecnologia OLE (*Object Linking and Embedding*) foi desenvolvida pela Microsoft nos anos 90, para suprir a necessidade de se integrar diferentes aplicações dentro da plataforma Windows, de forma a solucionar os problemas de desempenho e confiabilidade do até então utilizado padrão DDE (*Dynamic Data Exchange*). A tecnologia DCOM (*Distribuited Component Object Model*) surgiu juntamente com o sistema operacional Windows NT e foi logo aceita pela indústria. O DCOM é um conjunto de definições para a implementação de aplicações de arquitetura com modelo cliente/servidor e esta tecnologia veio para suprir a necessidade da comunicação de aplicações localizadas em computadores diferentes. Desta forma, um cliente pode acessar diferentes servidores ao mesmo tempo e um servidor pode disponibilizar suas funcionalidades para diferentes clientes ao mesmo tempo e em diferentes localidades.

O OPC, baseado na tecnologia OLE/DCOM, estabeleceu regras de uso dos componentes COM *(Component Object Model*) para acesso a dados em tempo real e não sofre do desempenho e assuntos de escalabilidade presentes na tecnologia DDE. A tecnologia OLE/DCOM também surgiu como forma de estabelecer regras de comunicação entre aplicações, definindo interfaces que estão no nível do usuário, acima dos protocolos de comunicação. O OPC utiliza um modelo cliente-servidor onde o servidor oferece interfaces para os objetos OPC, gerenciando os mesmos. Dessa forma, existem interfaces, métodos e classes especialmente voltadas para a necessidade de controle de processos, reunidas na forma de especificações, cada uma delas implementando um conjunto específico de funcionalidades. O OPC estabelece um padrão que permite aos desenvolvedores criar servidores de alto desempenho para

extrair dados de qualquer fonte, equipamento ou aplicação, fornecendo um produto utilizável por qualquer aplicação cliente que contenha a tecnologia OPC.

Antes do OPC, caso uma aplicação cliente (sistema supervisório, por exemplo) requeresse acesso a uma determinada fonte de dados do sistema, o próprio fabricante deveria desenvolver o *driver* necessário, o que gerava os seguintes problemas: fabricantes de *software* desenvolvendo *drivers* distintos para o mesmo *hardware*, funcionalidade do *hardware* indisponível da mesma forma por *drivers* de fabricantes diferentes, mudança de funcionalidade do *hardware* levando *drivers* antigos à incompatibilidade e dois *drivers* independentes não podem (geralmente) acessar um mesmo dispositivo simultaneamente. Em outras palavras, os fabricantes de *hardware* tentaram resolver esses problemas desenvolvendo *drivers*, mas não conseguem resolver problemas relativos a protocolos, permanecendo a restrição quanto ao uso por todas as aplicações. Atentos aos problemas, em 1995 alguns fabricantes de *softwares* de automação reuniram-se e desenvolveram, com o suporte da Microsoft, o 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.

A primeira especificação produzida pelo grupo foi publicada em agosto de 1996, e foi denominada "*OPC Specification Version* 1.0" que mais tarde passou a ser chamada "*OPC Data Access Specification Version* 1.0". O principal objetivo da publicação da especificação OPC é atender às necessidades da indústria, através de seu aprimoramento e sua ampliação. Atualmente existem as seguintes especificações: *OPC Overview*, *OPC Common Definitions and Interfaces*, *OPC Data Access Specification*, *OPC Data Access Automation Specification*, *OPC Data Exchange Specification*, *OPC Complex Data Specification*, *OPC Unified Arquitecture Specification*, *OPC Express Interface*, *OPC Alarm and Events Specification*, *OPC Historical Data Access Specification*, *OPC Batch Specification*, *OPC Security Specification* e *OPC XML Data Access Specification*. A especificação que foi utilizada para a confecção do servidor OPC foi a especificação de acesso de dados (OPC DA) na versão 3.00.

A especificação *OPC DA (Data Access)* especifica regras para a troca de dados em tempo real entre clientes e servidores OPC. Por essa especificação é que os servidores OPC 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 clientes e servidores. Desse modo, o funcionamento dos servidores OPC consiste em fazer aquisição de dados e deixar essa informação acessível para os clientes OPC que neles se conectarem.

A publicação das especificações para o padrão OPC possibilitou o desenvolvimento de diversos produtos para automação industrial, os quais se beneficiam das seguintes vantagens: padronização das interfaces de comunicação entre os servidores e clientes de dados de tempo real, facilitando a integração e manutenção dos sistemas; eliminação da necessidade de *drivers* de comunicação específicos (proprietários); melhoria do desempenho e otimização da comunicação entre dispositivos de automação; interoperabilidade entre sistemas de diversos fabricantes; redução dos custos e tempo para desenvolvimento de interfaces e *drivers* de comunicação, com conseqüente redução do custo de integração de sistemas; facilidade de desenvolvimento e manutenção de sistemas e produtos para comunicação em tempo real e facilidade de treinamento.

O SISAL tem como objetivo criar uma interface simples para supervisão de poços de petróleo equipados com sistemas de elevação artificial. Esse sistema foi desenvolvido pela Petrobras em parceria com a Universidade Federal do Rio Grande do Norte (UFRN). O SISAL utiliza o padrão de comunicação cliente/servidor na troca de informações onde o cliente se comunica com o servidor através do protocolo de aplicação PCI.

A arquitetura atual do SISAL já apresenta a capacidade de comunicação segundo o protocolo OPC que acessa informações (referentes à poços), de um banco de dados e as disponibiliza para clientes. Em outras palavras temos, neste contexto, um servidor OPC que acessa informações de poços diretamente do banco de dados do SISAL e disponibiliza as mesmas para supervisão em clientes OPC, gerando um problema de sobrecarregamento de processamento no referente banco de dados. O objetivo do projeto é desenvolver um *software* que se comunique com o servidor SISAL através do protocolo PCI e faça o papel de um servidor de informações referentes à poços via protocolo OPC.

2. Revisão Bibliográfica

Neste tópico são discutidos aspectos do projeto referente à capacidade de comunicação segundo o protocolo OPC para o SISAL através de um servidor OPC. É importante o entendimento de alguns assuntos para a compreensão do projeto. Para a compreensão do protocolo OPC, é necessário entender como os clientes OPC e os servidores OPC funcionam e trocam informações e, para entender isto, são necessários alguns conceitos relativos à especificação de acesso de dados do OPC. Para a compreensão do modo de funcionamento de servidores OPC é de extrema importância o entendimento do conceito de *namespace* ou espaço de nomes do OPC, bem como seus tipos: *namespace* dinâmico e *namespace* estático. Para a confecção do servidor OPC foi utilizado um *toolbox*, dessa forma alguns aspectos referentes ao *toolbox* também precisam ser compreendidos como: vantagens e limitações do mesmo e seus principais métodos e construtores utilizados para a confecção do servidor OPC em questão. No contexto deste projeto o servidor SISAL é que irá fornecer as informações para o servidor OPC, portanto, o sistema supervisório SISAL também é discutido.

2.1. Aplicabilidade do Protocolo OPC

Como já foi discutido anteriormente, o DDE é um protocolo para troca dinâmica de informações entre aplicativos na plataforma Windows. Através deste protocolo é possível comunicação entre duas ou mais aplicações (um servidor e um ou mais clientes), através de mensagens padronizadas que permitem ao cliente obter dados a partir do servidor.

O protocolo OPC surgiu como um novo padrão de comunicação capaz de integrar verticalmente todos os níveis hierárquicos relacionados ao controle da produção como, por exemplo, gerenciamento, supervisão de processos, controle e equipamentos no chão de fábrica, facilitando o acesso à informação de forma a acelerar tomadas de decisão.

O trabalho proposto por Souza et. al. (2005), "Gerência de Informação de Processos Industriais: Um Estudo de Caso na Produção de Petróleo e Gás", se refere ao desenvolvimento de um sistema que propõe uma estratégia para gerência de dados históricos e *on-line* de plantas industriais através da Internet (GERINF – Gerência de Informação). A coleta de dados neste sistema pode ser realizada a partir dos sistemas de supervisão ou diretamente a partir dos CLPs (Controladores Lógicos Programáveis), utilizando a tecnologia de comunicação DDE. O trabalho realizado por Leitão (2007) propõe o desenvolvimento de uma maneira alternativa para a captura destes dados no sistema GERINF utilizando o padrão de comunicação OPC, através da implementação de um cliente OPC. De acordo com Leitão (2007), o baixo rendimento e instabilidade do padrão DDE, além das tendências mercadológicas e a eficiência e bom desempenho do protocolo OPC, foram motivações para o desenvolvimento de seu trabalho. Leitão (2007, p.4) "afirma que esta alternativa não substituirá a já existente (DDE), mas sim será mais uma opção para a aquisição dos dados".

Dessa forma, vantagens como interoperabilidade (devido à padronização da comunicação entre clientes e servidores), menores tempos e custos de implementação de projetos, organização dos dados tanto por parte dos servidores como por parte dos clientes e o bom desempenho e eficiência da comunicação OPC fazem desta tecnologia a melhor alternativa a ser utilizada para aquisição de dados em ambientes industriais.

2.2. O SISAL

O SISAL é um sistema que tem como objetivo criar uma interface simples para supervisão de poços de petróleo equipados com sistemas de elevação artificial. Este sistema pertence à Petrobrás e foi desenvolvido pela UFRN.

O SISAL contém 4 módulos: Cliente, Servidor, Mestre de Campo, e Mestre de Banco, além de um banco de dados (SGBD – Sistema Gerenciador de Banco de Dados). Os clientes são responsáveis por apresentar de uma forma amigável as informações de supervisão aos usuários. É dos clientes que partem as requisições de dados para o servidor e são eles que devem utilizar essa informação no processo de monitoramento. O servidor deve rotear as informações entre os cliente e os mestres recebendo as requisições dos clientes, enviando-as aos seus respectivos mestres e devolvendo as

respostas aos clientes quando recebidas dos mestres. O servidor também é responsável pelo controle de acesso dos usuários, gerenciamento da segurança do sistema e das ações automáticas. O SISAL oferece como recurso adicional o acesso aos dados do sistema utilizando o protocolo OPC, como é discutido ainda nesta seção. Os mestres de campo representam o último ponto do sistema antes da comunicação com os poços. Um Mestre é uma aplicação dedicada ao controlador que é utilizado para traduzir as requisições de alto nível dos clientes para um acesso a uma determinada região da memória do controlador. Em outras palavras, o mestre é aquele que esteja conectado ao poço cuja informação se está solicitando. Os mestres de campo devem ainda realizar o *polling* de dados dos poços ciclicamente. O SGBD é o responsável por armazenar todas as informações de configuração dos poços, do sistema e dos usuários. O mestre de banco, ou mestre BD, centraliza as solicitações dos clientes, e repassadas pelo servidor, destinadas a ler ou escrever na base de dados.

Em uma rede de campo, de acordo com o funcionamento dos protocolos Mestre/Escravo, as solicitações de informação originam-se no Mestre e são enviadas para todos os escravos através do canal único de comunicação, o que caracteriza uma comunicação em *broadcast*. Somente o escravo que recebeu a mensagem deve responder à solicitação.

A maioria das redes de supervisão local utiliza o padrão de comunicação cliente/servidor na troca de informações e este é o padrão utilizado pelo SISAL. Em uma rede de supervisão local, o servidor deve receber às requisições de serviço dos clientes. O servidor, neste caso, tem o objetivo de interconectar esta rede à rede de campo, permitindo que qualquer cliente seja capaz de acessar, de forma indireta, os dados do processo físico adquiridos na rede de campo. Dentro da rede de supervisão local o SISAL utiliza dois protocolos no nível de aplicação, o Protocolo de Comunicação Interna (PCI) e o Protocolo de Comunicação com os Mestres (PCM).

O protocolo PCI é responsável pela troca de informações no SISAL entre os clientes e o servidor SISAL. O PCI é um protocolo da camada de aplicação encapsulado no TCP/IP. A solicitação ou envio de informações utilizado pelo PCI deve ser iniciada sempre através de requisições que são processadas pelo servidor, que interliga as duas redes. Geralmente, as requisições têm origem no cliente. Em um caso particular, uma requisição pode ter sua origem no próprio servidor. O destino de uma requisição será sempre o servidor, que deve colocá-la em uma fila do tipo FIFO (*First In First Out*),

processá-la e respondê-la. No PCI, para cada requisição recebida pelo servidor, existem duas respostas correspondentes, sendo uma parcial e outra definitiva. A resposta parcial é gerada pelo servidor quando o mesmo receber uma requisição. Logo após isso, o servidor deve enviar uma esta resposta para a origem da requisição, para informar se tem condições de processar a requisição recebida. O servidor também deve, após processar a requisição, enviar para o(s) cliente(s) apropriado(s) uma Resposta Definitiva contendo os dados solicitados ou a confirmação dos dados enviados na requisição.

O protocolo PCM é responsável pela troca de informações no SISAL entre o servidor e os Mestres. O PCM é um protocolo da camada de aplicação encapsulado no TCP/IP. A solicitação ou envio de informações utilizando o PCM deve ser iniciada sempre através de requisições que partem do servidor e são processadas pelos mestres.

Se um usuário necessitar alguma informação referente ao poço, fará uma requisição ao servidor (através do cliente utilizando o protocolo PCI) que a transmitirá para o devido mestre (através do protocolo PCM). Essa requisição chegará ao mestre que deverá, através do protocolo de comunicação utilizado pelo controlador do poço, fazer a leitura da região de memória correspondente no controlador.

A situação atual do SISAL apresenta um servidor OPC que acessa as informações dos poços diretamente do SGBD, causando sobrecarregamento de processamento no referido banco de dados do SISAL. As informações do servidor OPC são enviadas para os clientes OPC que se conectarem a ele e façam alguma requisição de acesso de dados. A figura a seguir ilustra a situação atual do SISAL.

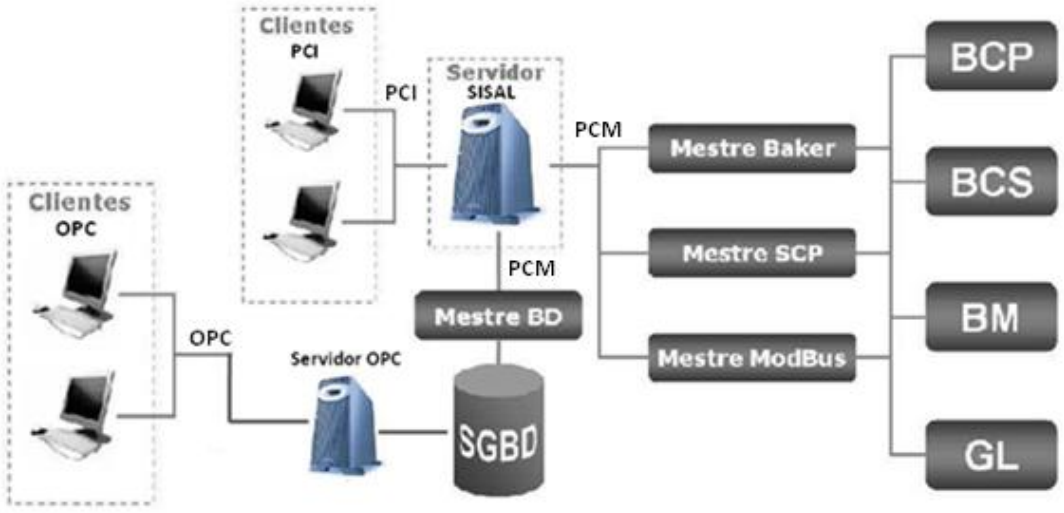


Figura 2.1 – Situação atual do SISAL

Neste trabalho, deseja-se desenvolver um novo servidor OPC que, através do protocolo PCI, acesse as informações diretamente do servidor SISAL e disponibilize-as para os clientes OPC que nele se conectarem e requisitarem pelo acesso de dados através do protocolo OPC. A figura a seguir ilustra a situação desejada para o SISAL.

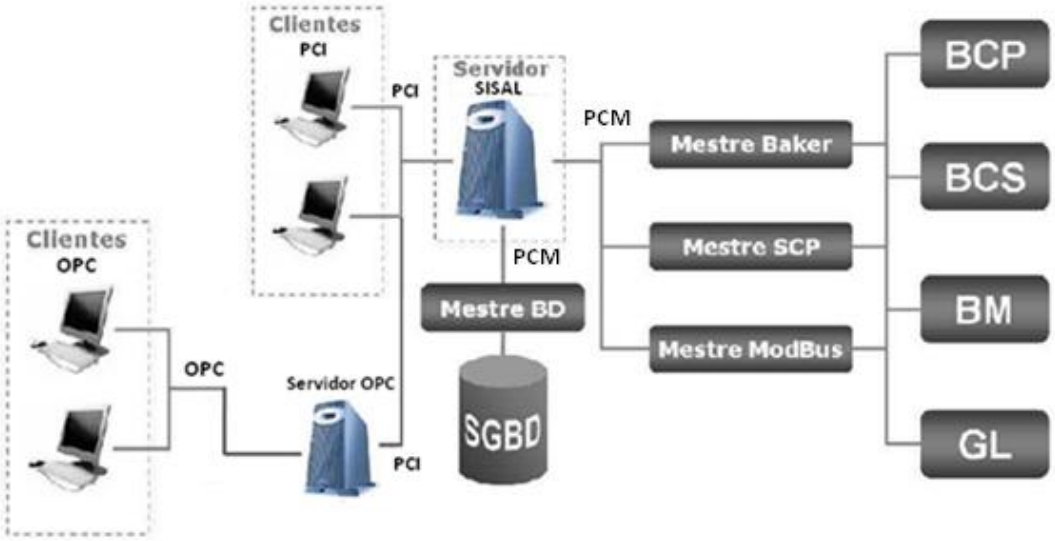


Figura 2.2 - Situação desejada para o SISAL

2.3. Clientes OPC e Servidores OPC

O funcionamento dos servidores OPC consiste em fazer a aquisição dos dados e deixar uma informação acessível para os clientes OPC, que nele se conectarem, através do espaço de nomes OPC ou *namespace*. Esta disponibilização se dá através do agrupamento de nós e itens que depende da aplicação a qual será utilizada o servidor OPC. A característica mais evidente dos nós é que eles servem para organizar o *namespace* em árvores – formando um *namespace* hierárquico (*hierarquical namespace*) – e funcionam como uma espécie de diretório para organização dos itens. Um item é um dado que pode ser uma temperatura, pressão, estado de uma válvula, ou qualquer outra variável, disponível para ser lido ou escrito por um cliente OPC. Também é esperado que o servidor consolide e otimize as requisições de acesso a dados de vários clientes, promovendo comunicações eficientes com os dispositivos de campo. Para leitura, os dados retornados pelos dispositivos são armazenados em um buffer para distribuição assíncrona ou coleta síncrona por vários clientes OPC. Para escritas, o servidor OPC atualiza os dados nos dispositivos físicos, independente dos clientes 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 estruturá-los. Isso pode ser feito por aspectos lógicos (variáveis de processo com comportamentos 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 (LEITÃO, 2006, p. 32).

De acordo com Fonseca (2002), o desempenho da comunicação OPC se aproxima do desempenho apresentado por sistemas que utilizam *drivers* de comunicação específicos e otimizados. Como um servidor OPC é uma camada de *software* a mais para implementar a organização da disponibilização de dados (através do *namespace*) e os mecanismos de comunicação com o cliente, é de se esperar que o desempenho do mesmo só seja afetado em relação a comunicação com o cliente e não com o dispositivo de campo. Em relação ao dispositivo de campo, pode-se implementar o *driver* e o protocolo que melhor se ajustem às necessidades do dispositivo e da rede de comunicação. Como estes *drivers* específicos possuem normalmente um bom desempenho, o desempenho do servidor OPC está mais relacionado à capacidade dos recursos de *hardware* da máquina que executa a aplicação do servidor do que propriamente do *driver* específico. Como os recursos de *hardware* estão cada vez mais poderosos em relação à capacidade de processamento, isto não tem se mostrado como um problema real.

2.4. Especificação de Acesso de Dados OPC

A especificação de acesso de dados fornece funcionalidades básicas para o acesso (leitura e escrita) de dados a partir de vários dispositivos através de um conjunto padrão de interfaces. As interfaces facilitam a interoperabilidade entre clientes e servidores e a comunicação entre o conjunto de recursos do cliente e do servidor, além de disponibilizar os vários mecanismos de ler e escrever os itens de acordo com as necessidades do aplicativo cliente. Dessa forma, um cliente OPC pode se conectar a servidores OPC de um ou mais fabricantes diferentes, como é ilustrado na figura abaixo.

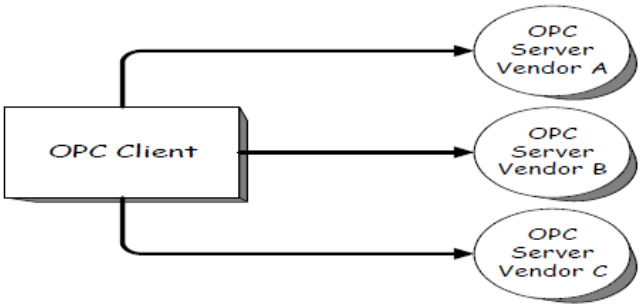


Figura 2.3 - Cliente OPC conectado a servidores OPC de fabricantes diferentes

"Um servidor OPC determina, através de seu código, os dispositivos e dados a que ele terá acesso, os detalhes sobre como o servidor disponibiliza os dados fisicamente, dentre outras coisas" (OPC FOUNDATION, 2003, p.2). A figura 4 ilustra a interoperabilidade entre servidores OPC e cliente OPC – servidores OPC operando juntamente com clientes de vários fornecedores diferentes.

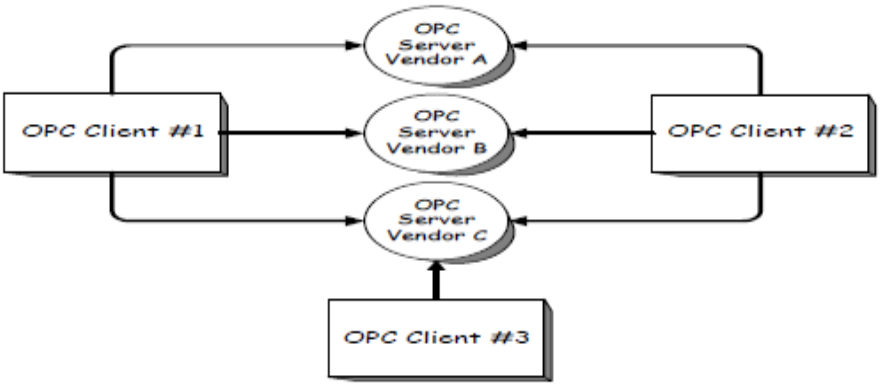


Figura 2.4 - Interoperabilidade entre clientes OPC e servidores OPC

Como já foi discutido anteriormente, a especificação OPC DA especifica regras para a troca de dados em tempo real entre clientes OPC e servidores OPC. Por essa especificação, são que os servidores OPC se comunicam com os dispositivos e disponibilizam essas informações para os clientes. Essa especificação trata apenas de como devem ser desenvolvidos alguns componentes COM, tanto do lado do cliente como do lado do servidor, para que a troca de dados possa ser feita independentemente de quem implementou os aplicativos. Esses componentes COM possuem interfaces que contêm os seguintes objetos: *OPCServer*, *OPClient*, *OPCGroup* e *OPCItem*. Neste trabalho será utilizado um *toolbox* para a implementação do servidor OPC. Este *toolbox* contém estas interfaces já implementadas. Mais detalhes sobre estes objetos serão discutidos na sessão a seguir.

2.4.1. Objetos da Especificação de Acesso de Dados OPC

O objeto *OPCServer*, *OPC Group* e *OPCItem* são implementados pelo servidor OPC e o objeto *OPClient* é implementado pelo cliente OPC.

- *OPCServer* – 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. Realiza todo o gerenciamento de conexão com o cliente e retorno dos dados, fornece navegação pelas *tags* disponíveis no servidor, métodos para gerenciamento pelo cliente de objetos *OPCGroup* como, por exemplo, criação e destruição, entre outros;

- *OPCGroup* – realiza o agrupamento lógico e gerenciamento de estados dos objetos *OPCItem*, é responsável pela criação, alteração e remoção dos grupos no servidor OPC – o objeto *OPCGroup* fornece meios para os clientes OPC organizarem os dados. É responsável também por disponibilizar os métodos de escrita/leitura nos itens.

- *OPCItem* – representa o dado de campo propriamente dito, também chamado de item, e é totalmente gerenciado pelo objeto *OPCGroup*. Os itens representam conexões a fontes de dados no servidor. Um Item não é acessível como um objeto por um cliente OPC. Portanto, não há interface para implementação definida por um "*OPCItem*". Todo o acesso aos itens é através de um objeto *OPCGroup* que "contém" o *OPCItem*. Um valor, uma qualidade e um rótulo do tempo é associado a cada item. O valor é sob a forma de uma *VARIANT*, e a qualidade é semelhante à especificada pela tecnologia de comunicação *Fieldbus*. Nota-se que os itens não são as fontes de dados - são apenas conexões com eles. O item deve ser encarado apenas como uma maneira para especificar o endereço dos dados, não como uma fonte de física real dos dados.

- *OPCClient* – 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. O servidor ao acusar a variação significativa (maior que a banda morta) de um ou mais itens ativos irá passar os itens alterados como parâmetro. Cabe ao cliente saber o que fazer com este dado.

2.4.2. Tipos de Comunicação

"Existem 4 tipos de comunicação para que o cliente obtenha um determinado dado: escrita/leitura síncrona, assíncrona, *refresh* e *subscription"* (LEITÃO, 2006, p. 34). Os valores podem ser obtidos de um *CACHE*, localizado no servidor OPC, e podem também ser obtidos diretamente do dispositivo de origem (*DEVICE*).

- Escrita/Leitura síncrona – as escritas/leituras síncronas devem somente ser usadas se os dados de processo forem obtidos rapidamente. Na escrita/leitura síncrona, o cliente realiza uma requisição dos dados e os recursos de sistema só são liberados quando os valores são retornados pelo servidor, dessa forma, o cliente chama o método e espera pelo seu retorno.

- Escrita/Leitura assíncrona – o cliente chama o método no servidor e fica liberado imediatamente para continuar sua execução normal. Após certo intervalo, que depende de como o dado é requisitado, o cliente obtém o valor. Em outras palavras, o cliente se "cadastra" no servidor para receber determinada quantidade de dados através de uma requisição e libera os recursos logo após a chamada. A requisição é colocada em uma fila no servidor OPC e depois de processada a requisição, os dados solicitados são enviados ao cliente. Leituras assíncronas devem ser utilizadas se a obtenção dos dados no servidor demorar muito.

- *Refresh* – é o tipo de aquisição que faz com que o cliente leia regularmente todos os itens ativos de um determinado grupo, que também deverá se encontrar ativo independentemente se os dados sofreram alteração de valor ou não.

- *Subscription* – os três tipos de aquisição de dados descritos – síncrono, assíncrono e *refresh* – são feitos por iniciativa do cliente. Porém nem sempre isso é uma alternativa eficiente, já que o cliente obterá valores independentes deles terem sofrido qualquer alteração. 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).

2.4.3. Formato dos Dados

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

- Valor do dado (*Value*) - espaço reservado para a disponibilização do valor do item correspondente. Todo valor de item é do tipo VARIANT, definidos pelo objeto COM.

- Rótulo do tempo (*Time Stamp*) – informação de tempo correspondente ao instante em que o dado foi lido ou escrito. Esta informação representa o tempo em que o servidor recebeu um dado do dispositivo de campo ou por geração interna.

- Informação de Estado ou Qualidade (*Quality*) - são reservados 2 bytes para a qualidade do estado do dado fornecido pelo servidor. As qualidades podem ser:

  - *Good* (bom) - dado válido.

  - *Bad* (ruim) - no caso de perda do link, de comunicação com o dispositivo de campo, por exemplo.

  - *Uncertain* (indefinido) - no caso de existir o link de comunicação, mas o dispositivo de campo estiver fora de operação.

2.4.4. Hierarquia de Objetos e Espaço de Nomes OPC

A especificação de acesso de dados OPC, define dois diferentes conceitos que um servidor OPC deve implementar e que um cliente OPC deve utilizar: o *namespace* e a hierarquia de objetos. A hierarquia de objetos é uma forma de organização dos objetos da especificação de acesso de dados que permite flexibilidade aos clientes para criar seu conjunto de itens e grupos, definindo sua própria visão do processo. Os detalhes de como o *namespace* são definidos e configurados são de responsabilidade dos servidores OPC. Diferentemente da hierarquia de objetos, o *namespace* é único para cada servidor, e pode se associar com várias hierarquias de objeto ao mesmo tempo.

> O *namespace* nada mais é do que outra hierarquia criada e configurada no servidor para representar a topologia de todos os dispositivos monitorados pelo servidor. Ela é composta por itens com identificadores chamados *ItemIDs*, que identificam unicamente um dispositivo de campo (SILVA et. al., 2007, p. 35).

O objeto *OPCServer* encontra-se no topo da hierarquia de objetos. Logo após vem o *OPClient*, depois o *OPCGroup* e logo após o *OPCItem* que acessa diretamente o *namespace* do servidor, podendo acessar tanto um nó como também um item (dado). O *namespace* é o conjunto de dados (itens) e nós disponibilizados pelo servidor OPC e pode ser representado de duas maneiras:

- *Namespace* Hierárquico (*Hierarchical Namespace*) – estrutura de árvore com qualquer profundidade. Neste caso, os nós podem ser utilizados formando estruturas. Eles podem, por exemplo, representar dispositivos em uma instalação, enquanto os itens apareceriam nos nós representando uma determinada fonte de dados. A figura abaixo, ilustra a estrutura de um *namespace* hierárquico.

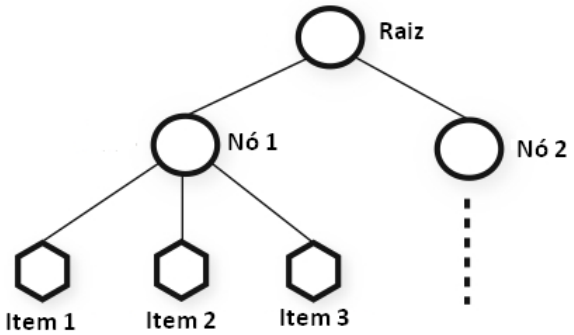


Figura 2.5 - Estrutura de um namespace hierárquico

- *Namespace* plano (*flat namespace*) – todos os itens estão em um único nível, ou seja, não existem nós.

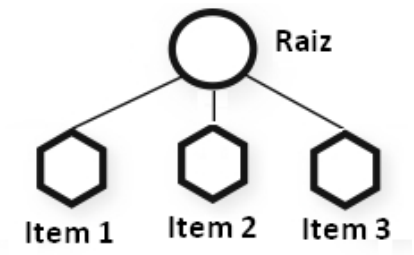


Figura 2.6 - Estrutura de um namespace plano

Existe também, a figura do nó raiz (*root*) que não é visível ao usuário no *namespace.* Entretanto a raiz representa a base de qualquer espaço de nomes, ou seja, a raiz é presente em ambos os tipos de *namespace* (*flat namespace* ou *hierarquical namescape*). É a partir da raiz que os nós e itens são construídos. A figura a seguir ilustra a estrutura da hierarquia de dados e do *namespace* OPC.

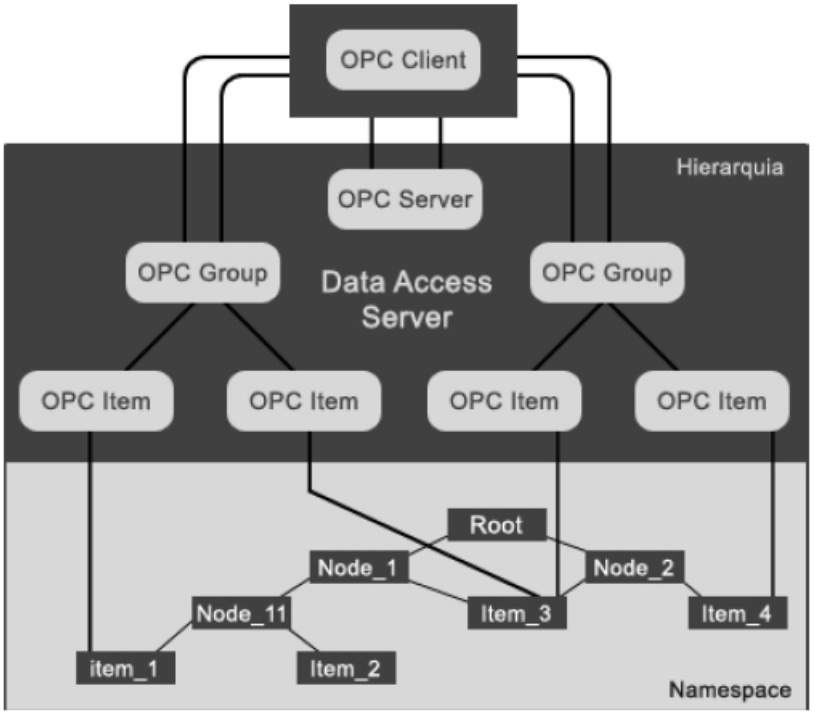


Figura 2.7 - Estrutura da hierarquia de objetos e do namespace OPC

A estrutura do nome para identificação de um item pela especificação de acesso de dados é: *Raiz.Nó1.Nó2.....Non.Item*. Na figura acima o *Item_2* é representado pelo nome *Root.Node_1.Node_11.Item_2*.

2.4.5. Namespace Estático e Namespace Dinâmico

O *browsing* do *namespace* pelo uso de clientes é bastante conveniente e é um dos fatores que garantem a interoperabilidade e fácil uso de servidores OPC. O *namespace* da maioria dos servidores OPC é criado durante a adaptação do servidor para aplicações concretas e se encontra logo disponível no ínicio de sua execução. Eles são estáticos. Porém, existem casos que requerem outra solução.

- A informação do *namespace* já se encontra disponível (base de dados de configuração) e não precisa ser acessada durante a configuração.

- O *namespace* é muito grande (mais de 20.000 itens), podendo influenciar no desempenho do servidor durante a partida e browsing.

- O *namespace* não é conhecido no momento de partida. (Aplicações dinâmicas).

Nestes três casos acima, a implementação de um *namespace* dinâmico é uma vantagem. No primeiro caso, a solução para o servidor seria redirecionar, durante o browsing do *namespace*, as requisições do cliente, por exemplo, para a base de dados de forma a obter a informação correspondente. Com essa solução, o cliente pode dar um *browse* no *namespace*, o qual não é estaticamente ligado ao servidor. Este é um caso especial para um *namespace* estático. Usualmente, um *namespace* criado durante a configuração é completamente lido durante a partida do servidor. No caso mencionado acima, isso é feito dependendo das requisições do cliente. Nos últimos dois casos, a lista de identificadores seria definida pela sintaxe e pela semântica que o cliente deve sempre usar. Baseado nessa sintaxe, o servidor cria os itens correspondentes à requisição do cliente.

2.4.6. Browsing do Namespace

Após iniciar o servidor OPC, a primeira tarefa do cliente é interfacear com o objeto *OPCServer*. O cliente deve percorrer o *namespace* para obter informações sobre sua estrutura e obter os identificadores *ItemIds* utilizados na criação de objetos *OPCItem*. O cliente faz, primeiramente, uma consulta à estrutura do *namespace*. Partindo do nó raiz, o cliente também obtém informações dos nós e itens que estão abaixo. O cliente pode ter acesso aos nós abaixo do nó corrente e aos Itens abaixo do nó corrente. Em um hierarquical namespace, o cliente tem a habilidade de mover-se pelo namespace, de forma a obter informações referentes a todos os nós e itens.

Depois do *browse* no *namespace* o cliente OPC cria objetos *OPCGroup* e *OPCItem*. Quando requisita a criação de um objeto *OPCGroup*, o cliente OPC transmite valores para nome simbólico, taxa de atualização de requisição (*request update rate*), percentual de banda morta (*percent dead band*) e estado ativo (*active state*).

Para criar o objeto OPCItem, o cliente OPC passa valores para os seguintes parâmetros: *fully qualified item id*, *active state*, *request data type*, *access path* e *client handle*.

2.5. Softing OPC Toolbox

Para o desenvolvimento do servidor OPC para o SISAL foi utilizado um *toolbox* (ou toolkit) que, entre outras funcionalidades, contém uma biblioteca de classes em C++ para o desenvolvimento de servidores OPC. O *Softing OPC Toolbox C++* é uma ferramenta de alta qualidade para a rápida e fácil criação de clientes e servidores OPC – já otimizados para aplicações e necessidades individuais. A *Softing* é membro da *OPC Foundation*. Todos os toolkits da *Softing* foram aprovados e certificados com os adequados testes de qualidade. Nesta sessão serão discutidas as funcionalidades, vantagens e limitações deste *toolbox*, as principais classes, construtores, métodos e enumerações que podem ser utilizados para a implementação de um servidor OPC e os mecanismos utilizados pelo *toolbox* para atualização dos valores do espaço de nomes OPC. A versão utilizada pelo *toolbox* para o desenvolvimento do produto é a versão 4.22.

2.5.1. Funcionalidades, Vantagens e Limitações

Como já foi discutido anteriormente, o *toolbox* OPC fornece um conjunto de ferramentas de ponta (ou uma biblioteca com classes, métodos, construtores e propriedades) que são utilizadas para o desenvolvimento rápido e fácil de clientes e servidores OPC, deixando bem mais fácil a implementação de qualquer produto com a tecnologia OPC. Utilizando uma abordagem similar a alguns outros fornecedores, este kit de desenvolvimento simplifica todas as funcionalidades OPC encapsulando-as em algumas DLL. Até mesmo para um desenvolvimento mais rápido, a ferramenta permite a geração de um projeto usando uma interface do tipo *wizard*, dessa forma o desenvolvimento do servidor pode ser preenchido através da aplicação de apenas oito passos. A partir deste ponto o código pode ser modificado pelo programador para atender a necessidade do projeto que será executado.

A versão do *toolbox* utilizada para o desenvolvimento do servidor OPC para o SISAL tem as seguintes funcionalidades e vantagens.

- Pode ser integrado a aplicações já existentes, podendo ser modificado o código fonte tanto do servidor OPC como do cliente OPC.
- Suporta as especificações: DA 1.0, DA 2.05, DA 3.0, AE 1.10, XML-DA 1.01 para Windows, XML-DA 1.0 para Linux, extensões DA/AE e XML-DA para Windows CE e XML-DA para Linux.
- Pode ser desenvolvido nos sistemas operacionais: Windows 2000, Windows XP, Windows 2003 Server, Windows Vista, Windows Server 2008, Windows 7, Windows CE e Linux nas linguagens C#, Visual Basic .NET e C++.
- Pode ser desenvolvido nas plataformas: . NET Framework 1.1, 2.0 e 3.5, Microsoft Visual Studio 2003, 2005 e 2008 e Microsoft Visual Studio 2005 express edition 2005 e 2008.
- Contém o código fonte completo e amostras de programas com o código fonte tanto de clientes OPC como de servidores OPC.
- Wizards para criação de clientes e servidores OPC. Versão demonstrativa.

Dentre todas essas vantagens e funcionalidades a única limitação é que esta ferramenta gera através do código fonte e da plataforma de implementação um aplicativo que roda em um tempo limitado de 90 minutos. Para a utilização do servidor OPC para o SISAL em campo será necessário adquirir o *toolbox* completo. A figura e as tabelas a seguir ilustram algumas das funcionalidades do respectivo *toolbox*.

Tabela 2.1 - Sistemas operacionais suportados pelas especificações OPC do toolbox

Toolkit Types			Operating System												
			Win NT 4	Win 2000	Win XP	Win XP embedded	Win 2003 Server	Win Vista	Win 2008 Server	Win CE .NET 4.x *	Win CE 5.0 *	Win CE 6.0 *	Linux Kernel 2.4	Linux Kernel 2.6	VxWorks
OPC Toolbox (C++, .NET) Version 4.22	Server	DA		✓	✓	✓	✓	✓	✓	✓	✓				
		AE		✓	✓	✓	✓	✓	✓	✓	✓				
* Win CE is supported by Version 4.11		XML-DA		✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	
		UA		🖉	🖉	🖉	🖉	🖉	🖉	🖉	🖉	🖉	🖉	🖉	🖉
	Client	DA		✓	✓	✓	✓	✓	✓	✓	✓				
		AE		✓	✓	✓	✓	✓	✓	✓	✓				
		XML-DA		✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	
		UA		🖉	🖉	🖉	🖉	🖉	🖉	🖉	🖉	🖉	🖉	🖉	🖉
OPC Toolbox ActiveX Version 3.20	Client	DA	✓	✓	✓	🖉	🖉	🖉	🖉						
		AE	✓	✓	✓	🖉	🖉	🖉	🖉						

✓ supported 🖉 under development

Tabela 2.2 - Plataformas de desenvolvimento suportadas pelas especificações OPC do toolbox

Toolkit Types			Environment				
			VS .NET 2003	VS 2005 (all versions C#, VB, C++ incl. Express)	VS 2008 (all versions C#, VB, C++ incl. Express)	GCC V3	GCC V4
OPC Toolbox (C++, C#, VB, .NET 1.1, 2.0, 3.5) Version 4.22 including OPC .NET DataControl (.NET 1.1, 2.0, 3.5)	Server	DA	✓	✓	✓		
		AE	✓	✓	✓		
		XML-DA	✓	✓	✓	✓	✓
		UA	🖉	🖉	🖉	🖉	🖉
	Client	DA	✓	✓	✓		
		AE	✓	✓	✓		
		XML-DA	✓	✓	✓	✓	✓
		UA	🖉	🖉	🖉	🖉	🖉
OPC Toolbox ActiveX Version 3.20	Client	DA	✓	✓			
		AE	✓	✓			

✓ supported 🖉 under development

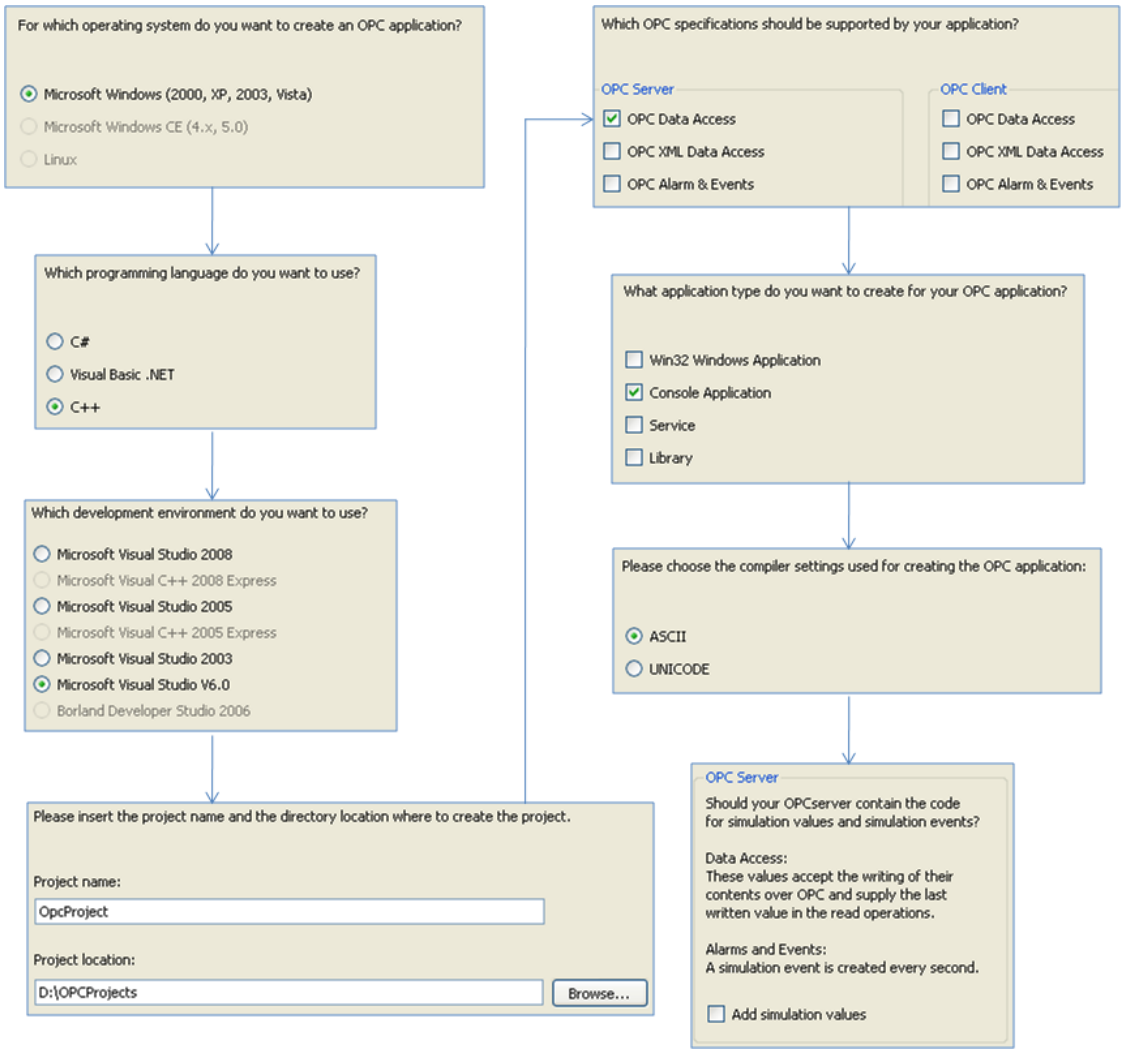


Figura 2.8 – Passos do wizard do Softing OPC Toolbox

2.5.2. Criação de um Namespace a Partir das Funcionalidades do Toolbox

A criação do espaço de nomes é um dos ponto mais importante deste trabalho. É a partir do namespace que os dados são organizados e disponibilizados para os clientes OPC. O namespace desejado para este trabalho é do tipo dinâmico – para se ter um embasamento de como construir este tipo de namespace, é necessário ter a compreensão do funcionamento das funcionalidades das classes do *toolbox* para a criação do citado namespace. Dentre estas classes estão as classes: *Application*, *Creator*, *ValueQT*, *DaAddressSpaceElement*, *DaAddressSpaceRoot*, *DaRequest*, *DaSession* e *DaTransaction*. O *toolbox* contém uma vasta biblioteca para implementação de servidores OPC, porém são somente estas classes que precisam ser compreendidas para o desenvolvimento do espaço de nomes do servidor OPC ou entendimento do funcionamento do respectivo servidor OPC. As principais classes, métodos, enumerações e diagramas de classe e os possíveis mecanismos de atualização dos valores do *toolbox* estão descritos no anexo A deste documento. Os mecanismos de atualização de valores disponibilizados pelo *toolbox* determinam como os servidores OPC verificam se os valores dos itens ativos em grupos ativos mudaram de acordo com a taxa de atualização especificada pelo cliente OPC. Existem 2 formas disponibilizadas pelo *toolbox* nas quais os valores são atualizados: *polling* e *report*.

O código a seguir representa um exemplo de criação do *namespace* utilizando as funcionalidades do *toolbox*. A seguir estão descritos os passos utilizados para a criação do *namespace* do exemplo.

- Passo 1 – Para a criação de qualquer objeto que compõe o servidor OPC é necessário acessar a instância da classe *MyCreator*, representada no exemplo pela variável *creator*.
- Passo 2 – Criação do nó raiz.
- Passo 3 – Criação de um nó.
- Passo 4 – Conexão da raiz com o nó, sendo o nó filho da raiz.
- Passo 5 – Criação do item, definição de seus direitos de acesso, definição do tipo de dado e definição do mecanismo de atualização de seu valor.
- Passo 6 – Conexão do item com o nó criado anteriormente.

```
long OpcServer::buildAddressSpace(void)
{
        // Criador de novas instâncias dos objetos que compõem o servidor OPC
         MyCreator* creator = (MyCreator*)getApp()->getCreator();
         tstring aName;

         // Criação da raiz
         DaAddressSpaceRoot* daRoot = getApp()->getDaAddressSpaceRoot();

         // Criação do nó
         m_pDaSimulationElement = (MyDaAddressSpaceElement*)creator->createMyDaAddressSpaceElement();
         aName = tstring(_T("Nó"));
         m_pDaSimulationElement1->setName(aName);
         m_pDaSimulationElement1->hasChildren(TRUE);
         m_pDaSimulationElement1->isBrowsable(TRUE);

         // Nó se conecta à raiz
         daRoot->addChild(m_pDaSimulationElement1);

         // Criação do item
         m_pDaSimulationElement2 = (MyDaAddressSpaceElement*)creator->createMyDaAddressSpaceElement();
         aName = tstring(_T("Item"));
         m_pDaSimulationElement2->setName(aName);
         m_pDaSimulationElement2->setAccessRights(EnumAccessRights_READABLE);
         m_pDaSimulationElement2->setDatatype(VT_BSTR);
         m_pDaSimulationElement2->setIoMode(EnumIoMode_POLL);

         // Item se conecta ao nó
         m_pDaSimulationElement1 ->addChild(m_pDaSimulationElement2);

         return S_OK;
}
```

A figura a seguir ilustra para o usuário o modo com o qual o cliente OPC mostra o *namespace* do construído pelo servidor OPC referente ao exemplo acima.

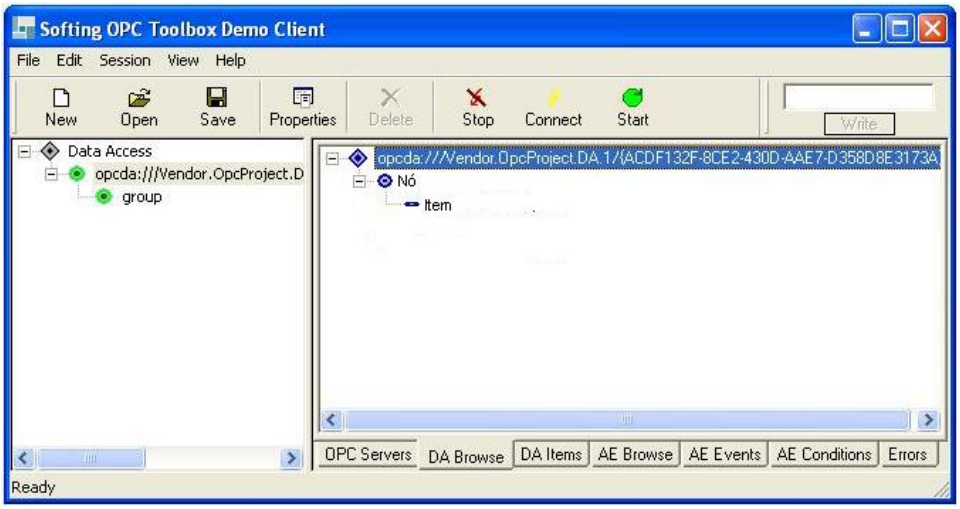


Figura 2.9 - Exemplo de criação de um namespace pelo toolbox

3. *Metodologia*

A metodologia utilizada para a elaboração do seguinte trabalho está composta pela seqüencia abaixo:

1. Estudo do protocolo OPC.
    - ❖ Estudo introdutório das tecnologias DDE, OLE, COM e DCOM.
    - ❖ Histórico da tecnologia OPC.
    - ❖ Estudo da especificação de acesso de dados.
    - ❖ Estudo sobre servidores OPC.
2. Pesquisa e escolha de um *toolbox* que implemente as funcionalidades de um servidor OPC.
3. Estudo detalhado do *toolbox* escolhido.
    - ❖ Estudo das funcionalidades, vantagens e limitações do *toolbox*.
    - ❖ Implementação de servidores OPC para compreensão dos membros das classes do *toolbox*.
4. Implementação de um servidor OPC genérico com *namespace* dinâmico.
5. Estudo do funcionamento da comunicação PCI utilizada no SISAL.
6. Anexação do servidor OPC implementado com o servidor SISAL.
7. Documentação do código do aplicativo.
8. Criação de um instalador para o programa.

9. Desenvolvimento da documentação técnica do servidor OPC para o SISAL (Anexo B).

    - ❖ Arquitetura do sistema.
    - ❖ Modo de funcionamento do servidor OPC.
    - ❖ Descrição dos arquivos utilizados no projeto.
    - ❖ Descrição das classes e diagramas de classe do programa.

10. Desenvolvimento de documentação de uso do servidor OPC para o SISAL (Anexo C).

    - ❖ Requisitos do sistema para instalação do programa.
    - ❖ Passo a passo para instalação e configuração do programa.
    - ❖ Modo de funcionamento do programa.

As tabelas a seguir referem-se ao cronograma das atividades realizadas para o desenvolvimento do trabalho.

Tabela 3.1 - Cronograma de atividades referente ao 1º semestre do projeto

Atividade	Julho 2008	Agosto 2008	Setembro 2008	Outubro 2008	Novembro 2008	Dezembro 2008
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						

Tabela 3.2 - Cronograma de atividades referente ao 2º semestre do projeto

Atividade	Janeiro 2009	Fevereiro 2009	Março 2009	Abril 2009	Maio 2009	Junho 2009
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						

Tabela 3.3 - Cronograma de atividades referente ao 3º semestre do projeto

Atividade	Julho 2009	Agosto 2009	Setembro 2009	Outubro 2009	Novembro 2009	Dezembro 2009
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						

Tabela 3.4 - Cronograma de atividades referente ao 4º semestre do projeto

Atividade	Janeiro 2010	Fevereiro 2010	Março 2010	Abril 2010	Maio 2010	Junho 2010
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						

4. Resultados e Discussão

Como parte inicial do projeto, foi feito um estudo do protocolo OPC, que foi de fundamental importância para a compreensão do código fonte do servidor OPC do *toolbox*. O estudo da especificação de acesso de dados da tecnologia OPC, do modo de funcionamento dos servidores OPC e a compreensão fundamental do conceito do espaço de nomes do servidor OPC foram de extrema importância para o entendimento das funcionalidades do *toolbox* e conseqüentemente da implementação do servidor OPC, através da modificação do código fonte do servidor OPC, pela modelagem do *namespace* do servidor, para que o *namespace* fosse criado de acordo com a quantidade de poços definidas pelo servidor SISAL. Em outras palavras, o estudo dos pontos já citados na metodologia, foi fundamental para a criação de um *namespace* dinâmico que é criado de acordo com a quantidade de poços que estão cadastrados no servidor SISAL.

4.1. Desenvolvimento de Uma Versão de Testes do Servidor OPC

Na parte do projeto referente a implementação do servidor OPC, a primeira versão do programa foi implementada sem que o servidor OPC coletasse as informações do SISAL. As informações e os valores de processo eram passados por um *socket* de um *software* que servia como um "servidor de dados". O *software* contém uma interface que permite que seja carregado um arquivo de texto com uma lista de strings como, por exemplo, uma lista poços e *tags.* Foi utilizada uma lista com as principais *tags* e poços cadastrados no servidor SISAL para demonstrar as seguintes funcionalidades e vantagens do servidor OPC:

- Criação de um *namespace* dinâmico que cresce de acordo com a quantidade de dados disponíveis no servidor ou dispositivo conectado no servidor OPC.
- Atualização dos valores dos dados em tempo real.

- Servidor OPC não é sobrecarregado devido ao grande número de nós e itens no seu *namespace*.

Abaixo é ilustrada a interface gráfica do "servidor de dados". O programa contém um botão para iniciar a conexão com o servidor OPC, um botão para carregar a lista de poços através de um arquivo de texto e outro botão para desconectar o *socket* cliente com o *socket* do servidor OPC.

Figura 4.1 - Interface gráfica do "servidor de dados"

O "servidor de dados" envia via *socket* uma pseudo-lista de poços para o servidor OPC que criará um *namespace* contendo as informações da lista de poços. O "servidor de dados" opera transferindo linhas de um arquivo de texto onde essas linhas são compostas por uma pseudo-lista de poços e *tags*. Cada linha tem a seguinte estrutura: "nome_do_poço&nome_do_dado&valor_do_dado", onde o nome_do_poço é um nó, o nome_do_dado é uma *tag* e o valor_do_dado é o valor da *tag*. Essas informações são passadas linha por linha de um *socket* servidor para um *socket* cliente. O *socket* cliente opera em uma *thread* de recebimento de dados, separando as informações (nome_do_poço, nome_da_tag e valor da *tag*) e as distribuindo, criando um *namespace* com *n* nós e *m tags*. A figura abaixo ilustra um esquema da arquitetura do sistema, mostrando a forma como o "servidor de dados" se comunica com o servidor OPC.

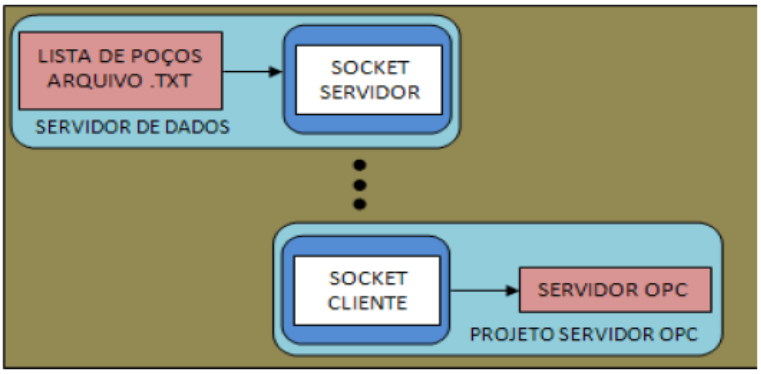


Figura 4.2 - Arquitetura do sistema da versão de testes do servidor OPC

A figura a seguir demonstra um exemplo da criação de um *namespace* hierárquico que é construído a partir de uma pseudo-lista de poços e *tags* contida no arquivo de texto.

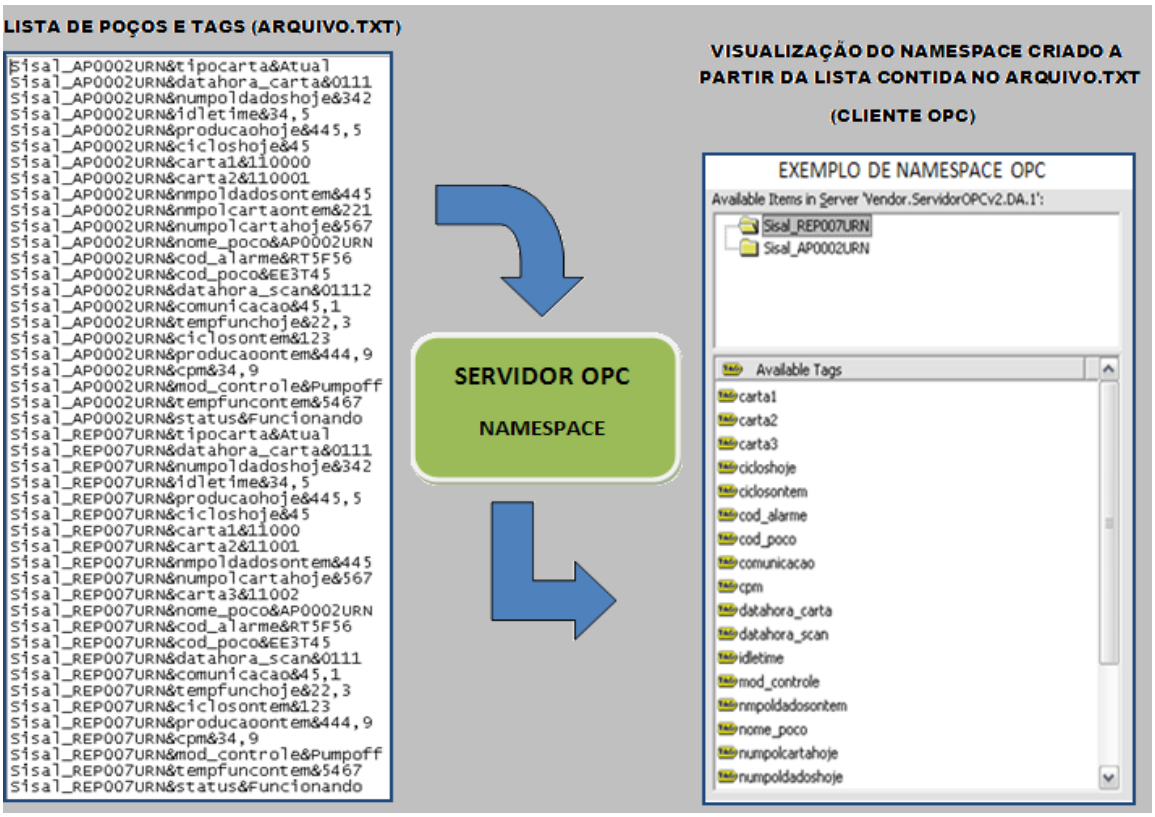


Figura 4.3 - Resultados obtidos através da implementação de uma versão de servidor OPC de testes

O “Servidor de dados” foi implementado para demonstrar que o servidor OPC desenvolvido pelo *toolbox* escolhido é um *software* bem generalizado, isto é, pode se comunicar com qualquer outro *software* servidor via *socket*. Dessa forma, o servidor OPC pode ser modelado para coletar as variáveis de processo de qualquer sistema ou

dispositivo, como por exemplo, as variáveis de sistemas supervisórios, porém é necessário ser modificado o protocolo de comunicação deste sistema com o servidor OPC e, se necessário, modificar a maneira de como o *namespace* é montado. Essa característica só é válida devido à flexibilidade que se tem para modificar o código fonte do servidor OPC através das ferramentas que são disponibilizadas pelo toolkit. É importante destacar que o Servidor de dados, assim como o protocolo de comunicação deste aplicativo com o servidor OPC não foram utilizados para a confecção do aplicativo Servidor OPC para o SISAL – estas implementações serviram somente para demonstrar as funcionalidades, vantagens e limitações do *toolbox* do servidor OPC e para implementar um *namespace* dinâmico eficiente que atualize seus valores em tempo real. Dessa forma, o servidor OPC de testes foi modificado para funcionar juntamente com o servidor SISAL, sendo necessário utilizar o protocolo de comunicação PCI para coletar os dados do servidor SISAL e disponibilizá-los no *namespace* do servidor OPC. A sessão a seguir tratará da versão do servidor OPC anexado ao servidor SISAL.

4.2. Desenvolvimento do Servidor OPC para o SISAL

O projeto é dividido em duas partes: uma parte trata-se do desenvolvimento de um servidor OPC com características dinâmicas e a outra a adaptação deste servidor OPC para coletar informações do SISAL. A primeira parte referente ao desenvolvimento do servidor OPC genérico já foi discutido na sessão anterior. A segunda parte refere-se à anexação do servidor OPC com o servidor SISAL. Dessa forma é necessário utilizar o protocolo de comunicação PCI para que o servidor OPC colete os dados de poços do servidor SISAL. Dessa forma, o servidor OPC pode ser considerado mais um cliente PCI do servidor SISAL. Como já foi discutido anteriormente, o cliente SISAL é responsável por disponibilizar os dados do servidor SISAL para o usuário. É dos clientes que partem as requisições de dados para o servidor e são eles que devem utilizar essa informação no processo de monitoramento – para isso o cliente SISAL utiliza o protocolo de comunicação PCI. A idéia então é a utilização da parte do código, contidos no cliente SISAL, referente à coleta da lista de poços e suas variáveis. Dessa forma, a idéia é incorporar estes arquivos do cliente SISAL ao servidor OPC, adicionando a comunicação PCI ao referido servidor.

4.2.1. Problemas e Soluções

Alguns problemas foram encontrados para a anexação do servidor OPC com o servidor SISAL, uma vez que o cliente SISAL foi desenvolvido na plataforma de desenvolvimento Borland C++ Builder versão 5.0, enquanto que o servidor OPC foi desenvolvido na plataforma Visual Studio C++ 2005 – o que acarretou em incompatibilidade de código das bibliotecas do C++ Builder e das bibliotecas do Visual Studio C++. O trabalho que passou a ser feito foi o de retirar componentes da biblioteca *vcl* (*Visual Component Library*) do C++ Builder e substituí-los por uma solução do Visual C++. As incompatibilidades de códigos encontradas foram referentes ao tratamento de: semáforos, *mutexes*, strings, e principalmente, *threads* e *sockets*. Uma nova solução foi encontrada para anexação do código do cliente SISAL, referente à coleta de dados dos poços do servidor SISAL, com o código do servidor OPC sem a necessidade da modificação devido à incompatibilidade das bibliotecas utilizadas.

O aluno de graduação em Engenharia de Computação, Alan Diego Dantas Protássio, desenvolveu uma DLL (*Dynamic-Link Library*), que tem a função de criar uma interface de comunicação PCI entre o servidor SISAL e o Servidor OPC. Uma DLL é uma biblioteca dinâmica de funções que podem ser chamadas a partir de outra aplicação. Uma DLL permite que programas possam acessar as suas funções, independentemente da linguagem de desenvolvimento destes programas e da linguagem de desenvolvimento da DLL. Dessa forma, é possível utilizar funções do código desenvolvido utilizando as bibliotecas do C++ Builder através do Visual C++. Os arquivos de código do cliente SISAL de extensão "*cpp*" e "*h*" utilizados para a implementação da DLL foram: *md5, Unt_Status, Unt_ClassePoco, Unt_ClasseSistema, Unt_PCI, Unt_FuncoesPCI, Unt_Principal, Unt_StringList, Unt_ProcessoPrincipal, Unt_ProcessoReceptor* e *Unt_ProcessoVerificador*. Além desses arquivos foi incluído o arquivo *Unt_Buffercircular.h*. Foi criado um arquivo (*Unit.cpp)* que contém 3 funções: uma função para verificar a versão do servidor SISAL e logar o usuário no servidor SISAL, uma função para capturar a lista de poços e uma para armazenar a lista em um vetor. Após fazer as devidas modificações nestes arquivos a fim da obtenção das funções que retornam a listas de poços e suas variáveis, um arquivo de extensão "dll" foi gerado (pci.dll). Os arquivos *UntPciInterface.cpp* e *UntPciInterface.h* foram criados, obtendo-se, desta forma, uma interface entre o aplicativo servidor OPC e o servidor SISAL. A descrição dos arquivos da DLL está localizado no anexo B deste documento.

4.2.2. Implementação do Código

Como já foi esclarecido antes, o assistente de instalação do *toolbox* gera um servidor OPC básico. Para criação do servidor OPC para o SISAL foi necessário a modificação de algumas partes do código original e a criação de outros arquivos, como é o caso dos arquivos *UntPciInterface.cpp* e *UntPciInterface.h* que são responsáveis por criar a interface de comunicação entre o servidor OPC e o servidor SISAL. Os três pontos mais importantes referentes à implementação do programa são:

- Interface de comunicação entre o servidor OPC e o servidor SISAL – Trata do mecanismo utilizado pelo servidor OPC para se logar no servidor SISAL, acessar a lista de poços e variáveis, atualizá-las e armazená-las em algum container para serem posteriormente tratadas na formação do *namespace* e atualização dos valores dos itens do *namespace* do servidor OPC.

- Definição e configuração dos elementos do espaço de nomes – Trata da criação dos de novas instâncias dos nós e dos itens do espaço de nomes, bem como suas configurações quanto a(o): definição de ser um nó ou um item, direito de acesso, mecanismo de atualização de seus dados, tipo de dado e nome.

- Criação do espaço de nomes e atualização dos valores do espaço de nomes – Trata do armazenamento dos nós e itens que já foram criados e configurados em um mapa de itens – o mapa de itens tem a função de armazenar todo o espaço de nomes para que seus valores não sejam perdidos. Trata também: da ligação dos itens, que representam as variáveis, com seus respectivos nós, que representam os poços, da definição dos valores, qualidade e *time stamp* dos itens e da atualização dos itens através da atualização do vetor de poços.

Outro ponto bastante importante é que o código gerado para implementação deste servidor OPC foi projetado de modo a permitir a incorporação da especificação de alarme e eventos do protocolo OPC (*OPC Alarm & Events Specification*). As descrições completas dos arquivos utilizados no projeto e o modo de funcionamento do aplicativo estão relatados na documentação técnica do programa que está localizada no anexo B deste documento.

4.2.3. A Interface do Programa

O aplicativo foi desenvolvido na plataforma Visual Studio C++ 2005 e foi utilizado o *Softing OPC Toolbox v4.22* para implementação das sua funcionalidades. O aplicativo, depois de inicializado, é executado em segundo plano, dessa forma sua interface não é visível ao usuário do programa. Como já foi discutido, o cliente é quem é responsável por mostrar como os dados estão distribuídos no servidor.

A figura a seguir, mostra a forma com que o *namespace* do servidor OPC é mostrado para o usuário através de um cliente OPC. Na figura percebe-se claramente que os nós correspondem ao nome dos poços (ARG0216U, ARG0249U, etc) e os itens correspondem às variáveis do poço (*cod_poço, Estação, End_Controlador e Grau API*). Cada nó no *namespace* pode ainda fornecer a informação do tipo de poço que está sendo monitorado. Para selecionar a *tag* ou o nó para serem monitorados basta clicar duas vezes no respectivo nó ou *tag*.

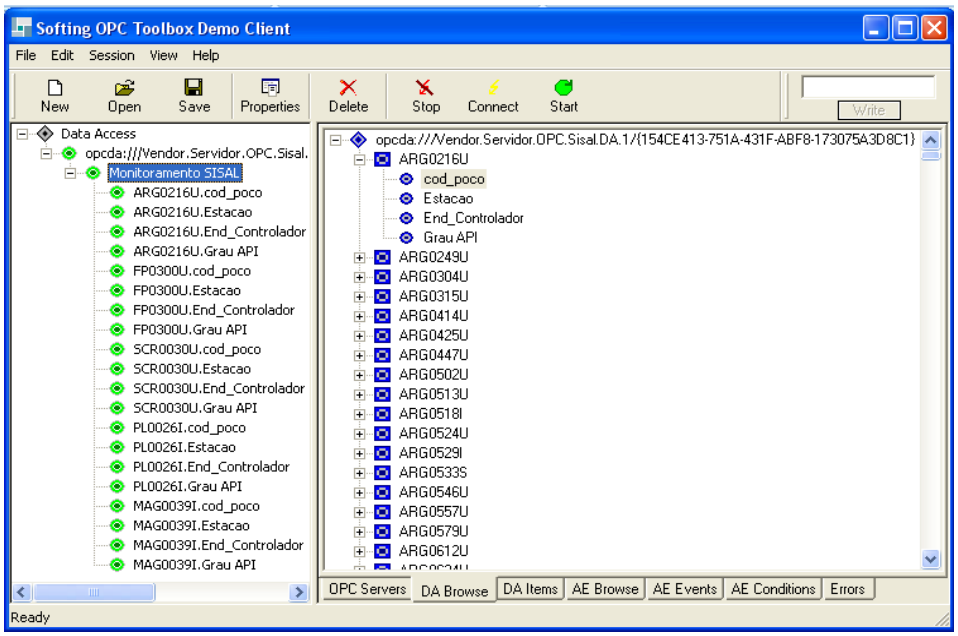


Figura 4.4 - Namespace do servidor OPC para o SISAL

A figura a seguir ilustra as propriedades da *tag Grau API* do poço ARG0216U. Na figura percebe-se claramente os três dados sobre os itens já discutidos na sessão referente à especificação de acesso de dados: o valor, a qualidade do item e o *time stamp*. Além destes dados, o cliente OPC da figura em questão também permite a visualização do tipo do item, do direito de acesso ao item e da taxa de atualização do item pelo servidor OPC.

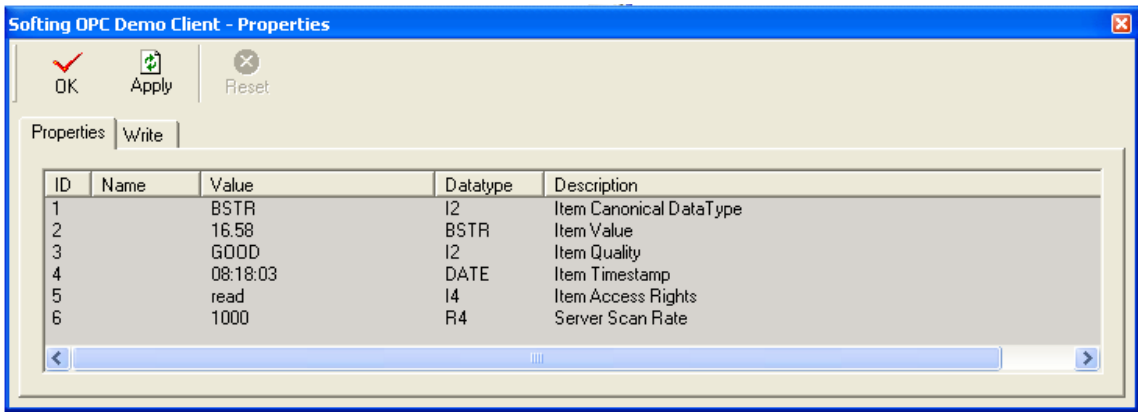


Figura 4.5 - Propriedades de um item do namespace

Como foi discutido anteriormente, o servidor OPC pode criar um *namespace* no qual seu tamanho varia de acordo com a quantidade de poços disponíveis no servidor SISAL, porém as *tags* estão limitadas às quatro variáveis já discutidas. Para adicionar alguma variável no *namespace* do servidor OPC, é necessário modificar o código do servidor OPC e o código da interface de comunicação com o servidor SISAL. Para ilustração de um exemplo, algumas *tags* do servidor OPC foram escolhidas. A figura a seguir, mostra os itens ativos pelo cliente OPC. Na figura a seguir é mostrada cada variável de poço (*tag*) referente a cada poço (nó). O nome da *tag* é composto pela seguinte estrutura: nome_do_poço.nome_da_tag. Nesta tela do cliente OPC, é possível observar os valores, a qualidade e o *time stamp* referente a cada *tag* selecionada para o monitoramento.

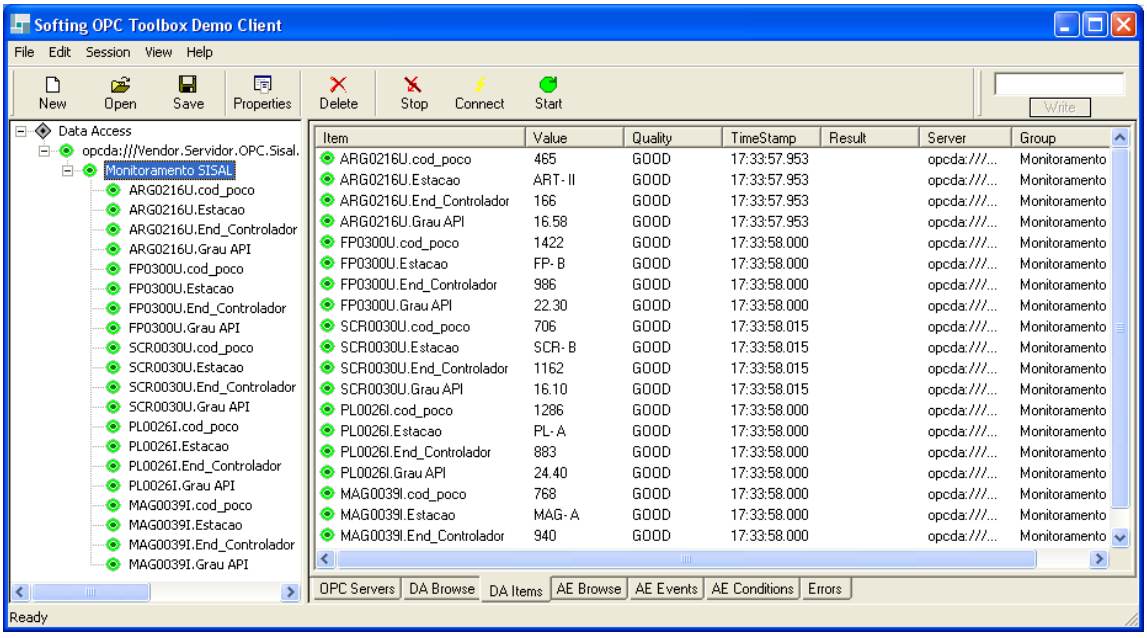


Figura 4.6 - Itens do namespace do servidor OPC ativos pelo cliente OPC

4.2.4. O Desempenho do Programa

O cenário de testes envolve o Cliente OPC que solicita as informações do servidor OPC que, por sua vez, faz requisições ao servidor SISAL. O servidor SISAL deve retornar as respostas às devidas requisições assim que as tiver. Neste mesmo cenário, o servidor SISAL utiliza o SGBD (banco de dados do SISAL) como fonte das informações dos poços. As primeiras versões do programa não faziam uso adequado dos recursos de *hardware*, causando o consumo excessivo de processamento, no momento da construção do *namespace*. A versão final do programa, já otimizado, realiza a construção do namespace de maneira mais eficiente, porém é verificado um grande consumo de processamento ao listar todas as *tags* (no cliente OPC) presentes no *namespace*. Dessa forma é mais viável acessar somente as variáveis que necessitem ser monitoradas no momento. Percebe-se também a ótima eficiência da comunicação OPC – a eficiência da comunicação OPC analisada neste trabalho refere-se à confiabilidade e rapidez com que os dados são disponibilizados do dispositivo de campo para o servidor OPC. Dessa forma percebeu-se que a comunicação foi realizada com rapidez e sem falhas ou com perda de dados.

4.2.5. Instalador do Servidor OPC SISAL V1.00

O desenvolvimento do instalador do aplicativo foi dividido em duas partes: uma referente à execução do aplicativo, compilado pelo *Visual Studio 2005 C++*, em outros computadores com o mínimo de necessidade de se instalar outros programas que serviriam de pré-requisitos para a execução do Servidor OPC SISAL e outra parte referente à confecção do instalador propriamente dito. Para executar o aplicativo em outras máquinas sem que seja necessária a instalação de outros programas, foi preciso encontrar as bibliotecas DLL que são utilizadas pela aplicação desenvolvida. Foram encontrados diversos problemas para executar o aplicativo gerado pelo *Microsoft Visual Studio C++* sem que os *drivers* da plataforma não estivessem instalados na devida máquina. A importação das prováveis DLL para executar o arquivo não solucionou o problema. Dessa forma, para instalar o servidor OPC para o SISAL, é necessária a instalação da versão livre da referida plataforma de desenvolvimento, a versão *Express Edition* do *Microsoft Visual Studio 2005*. Este problema é caracterizado como uma limitação do aplicativo devido ao fato de sua instalação ser dependente de outra

instalação. O instalador do Servidor OPC SISAL v1.00 foi criado a partir do programa *Inno Setup*.

Após a instalação do aplicativo, foi percebido que o CLSID (identificador exclusivo que identifica um objeto da classe COM) do servidor OPC não é cadastrado automaticamente no Cliente OPC. Dessa forma, outra limitação do aplicativo é a necessidade de cadastrar o CLSID do servidor OPC no cliente OPC manualmente para que o mesmo possa ser executado. O passo a passo da instalação do aplicativo está descrito na documentação de uso do programa localizado neste documento no anexo C.

4.2.6. Documentação Técnica do Servidor OPC para o SISAL

A documentação técnica do programa servidor OPC para o SISAL foi desenvolvida para o profissional de tecnologia da informação e ao desenvolvedor e descreve tecnicamente as linhas do projeto que foi implementado. A documentação técnica desenvolvida, além de ter toda a explicação de funcionamento do programa, arquitetura do sistema no qual o aplicativo vai funcionar e das vantagens, funcionalidades e limitações, contém também os diagramas de classe completos referentes ao código que foi utilizado, bem como a descrição dos arquivos utilizados para implementação do programa. A documentação técnica do Servidor OPC para o SISAL versão 1.00 está descrita no anexo B deste documento.

4.2.7. Documentação de Uso do Servidor OPC para o SISAL

A documentação de uso do programa Servidor OPC para o SISAL foi desenvolvida para o usuário final do programa e para o administrador do sistema que será monitorado com ajuda do aplicativo. Esta documentação tem como objetivo fornecer um manual do usuário, que permita ao usuário instalar, configurar e executar o aplicativo sem muitas dificuldades. A documentação de uso desenvolvida contém as seguintes partes: os requisitos do sistema, o passo a passo da instalação do aplicativo, configuração do aplicativo e o modo de funcionamento do aplicativo. A documentação de uso do Servidor OPC para o SISAL versão 1.00 está descrita no anexo C deste documento.

5. *Conclusões*

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 tornavam tarefas bastante complicadas. Havia então a necessidade do desenvolvimento de um padrão de comunicação. O OPC veio para acabar com este problema. Com a expansão da automação industrial, um grande volume de informações de processos torna-se disponível, porém, se não for inteligentemente organizado e processado, torna-se inútil. Os servidores bem confeccionados podem disponibilizar este grande volume de informações de maneira adequada para que os clientes possam mostrar na tela estes dados. Um bom servidor, além disso, pode disponibilizar estas informações em tempo real do processo informando com precisão dados coletados nos quais neste projeto são dados de poços de petróleo.

O desenvolvimento do Servidor OPC para o SISAL possibilita que os dados disponibilizados pelo servidor SISAL, através do protocolo PCI, possam ser também disponibilizados através do protocolo padronizado, OPC. Além de resolver o atual problema de sobrecarregamento do banco de dados do servidor SISAL, o servidor OPC oferece mais uma possível forma de acesso aos dados referentes aos poços para fins de monitoramento de variáveis em tempo real.

O desenvolvimento da documentação técnica possibilita que os desenvolvedores que possam possivelmente fazer alguma mudança ou atualização (*upgrade*) do aplicativo, tenham o máximo de informações possíveis sobre o código que foi desenvolvido. O desenvolvimento da documentação de uso possibilita que os usuários finais e os administradores do sistema possam ter acesso às informações referentes à instalação, configuração e modo de funcionamento do Servidor OPC para o SISAL, além do mesmo servir como um guia para responder as possíveis perguntas do usuário do programa.

Referências Bibliográficas

[1] NASCIMENTO, J. M. A.; GOMES, H. P.; SOUZA R. B. (2007). Documentação Técnica SISAL V 2.04 – Revisão 0, Universidade Federal do Rio Grande do Norte.

[2] LEITÃO, G. B. P. (2006). Arquitetura e Implementação de um Cliente OPC para Aquisição de Dados na Indústria do Petróleo, Monografia de graduação, Universidade Federal do Rio Grande do Norte.

[3] FONSECA, M. O. (2002). Comunicação OPC – Uma Abordagem Prática, VI Seminário de Automação de Processos, Associação Brasileira de Metalurgia e Materiais, 9-10 de outubro de 2002, Vitória, ES.

[4] SILVA, C. F.; LOPES, F. A. B.; NOBREGA, J. A.; COSTA, R. P. (2007). Protocolo OPC: Introdução e Aplicações na Automação Industrial, Monografia de graduação, Universidade do Estado do Rio de Janeiro.

[5] CÂNDIDO, R. V. B. (2004). Padrão OPC: Uma Alternativa de Substituição dos *Drivers* Proprietários para Acessar Dados de PLCs, Monografia de graduação, Universidade FUMEC.

[6] OPC FOUNDATION (2003). Data Access Custom Interface Standard Specification Version 3.00.

[7] IWANITZ, F.; LANGE, J. (2001). OLE for Process Control, Hunthig.

[8] SOUZA, R. B.; MEDEIROS, A. A. D.; NASCIMENTO, J. M. A.; MAITELLI, A. L.; GOMES H. P. (2006). SISAL – Um Sistema Supervisório para Elevação Artificial de Petróleo, Rio Oil & Gas Expo and Conference 2006, 11-14 de Setembro de 2006, Rio de Janeiro, RJ.

[9] SOUZA A. J. at. al. (2005). Gerência de Informação de Processos Industriais: Um Estudo de Caso na Produção de Petróleo e Gás, VII SBAI / II IEEE LARS, Setembro de 2005, São Luís, MA.

Anexo

Anexo A – Principais Classes, Métodos, Enumerações, Diagramas de Classe e os Possíveis Mecanismos de Atualização de Valores do Softing OPC Toolbox

1. Principais Classes, Métodos e Enumerações do Toolbox

A classe Application é uma das classes mais importantes do *toolbox* do servidor OPC. Ela fornece métodos para inicializar e encerrar o servidor, detecção e tratamento de objetos *OPCServer* além de manipulação de entrada e saída. A tabela abaixo contém os principais métodos da classe *Application*.

Tabela 1.1 - Principais métodos da classe Application

Métodos	Descrição
Activate	Reponsável pela ativação das funcionalidades do servidor.
getDaAddressSpaceRoot	Retorna o nó raiz do namespace.
Initialize	Inicializa as funcionalidades do servidor.
Ready	Prepara os componentes do toolkit para inicialização de E/S.
Start	Inicializa o servidor. Precede a criação do namespace.
Stop	Finaliza as funcionalidades do servidor.
Terminate	Reponsável pela desativação das funcionalidades do servidor.

A classe *Creator* é utilizada para criar as instancias dos vários objetos que compõem o servidor OPC. A tabela a seguir demonstra os principais métodos da classe *Creator*.

Tabela 1.2 - Principais métodos da classe Creator

Métodos	Descrição
Creator Constructor	Inicializa uma nova instância da classe Creator.
CreateDaAddressSpaceElement	Cria uma nova instância da classe DaAddressSpaceElement.
CreateDaAddressSpaceRoot	Cria uma nova instância da classe DaAddressSpaceRoot.

A classe *ValueQT* representa o valor de uma variável do processo com a sua qualidade e o tempo que foi obtido pelo dispositivo ou pelo *cache* do servidor. Em outras palavras a classe *ValueQT* é responsável por definir o formato dos dados (dos itens) que são trocados entre o cliente OPC e o servidor OPC. O construtor da classe *ValueQT* cria uma nova instância da classe com os seguintes parâmetros de entrada: valor do dado do processo (*value*), a qualidade do valor (*quality*), e o tempo ao qual o dado foi obtido do dispositivo ou do cachê (*time stamp*). A tabela a seguir lista os principais métodos da classe *ValueQT*. Em relação aos tipos dos parâmetros de entrada do construtor desta classe pode-se dizer que: o valor do dado é do tipo criado pela classe *VARIANT*, a qualidade do valor é dado pela enumeração *EnumQuality* e o *time stamp* é dado pelo tipo criado pela classe *DateTime*.

Tabela 1.3 - Principais métodos da classe ValueQT

Métodos	Descrição
ValueQT	Cria uma nova instancia da classe ValueQT.
getData	Retorna o valor do dado do processo.
getQuality	Retorna a qualidade do dado.
getTimeStamp	Retorna o tempo ao qual o dado foi obtido.
setData	Define o valor, a qualidade e o time stamp do dado.
toString	Retorna uma representação em string do valor do dado.

A classe *DaAddressSpaceRoot* é responsável pela criação do nó raiz do *namespace*, criação de elementos (nós ou itens) e gerenciamento destes elementos a partir do nó raiz. A tabela a seguir lista os principais métodos da classe *DaAddressSpaceRoot*.

Tabela 1.4 - Principais métodos da classe DaAddressSpaceRoot

Métodos	Descrição
DaAddressSpaceRoot	Construtor responsável por criar o nó raiz do namespace.
addChild	Adiciona um elemento como filho do nó raiz.
getChildren	Retorna um vetor com todos os filhos do nó raiz.
removeChild	Remove um filho específico do nó raiz.
ValuesChanged	Altera o valor do cache de todos os elementos do namespace.

A classe *DaAddressSpaceElement* é a classe base para o desenvolvimento do espaço de nomes da especificação de acesso de dados do OPC. A tabela a seguir lista os principais métodos da classe *DaAddressSpaceElement*.

Tabela 1.5 - Principais métodos da classe DaAddressSpaceElement

Métodos	Descrição
DaAddressSpaceElement	Construtor de um elemento (nó ou item).
getAccessRights	Retorna o direito de acesso do elemento.
getActive	Retorna se o elemento está ativo ou não no namespace.
getChildren	Retorna um vetor com todos os filhos do elemento.
getDataType	Retorna o tipo de dado deste elemento.
getHasChildren	Retorna se o elemento contém filhos ou não
getIoMode	Retorna o modo de E/S deste elemento.
getIsBrowsable	Retorna se o elemento é um nó ou um item.
getItemId	Retorna o identificador (ID) do elemento.
getName	Retorna o nome do elemento.
getParent	Retorna o nó pai do elemento.
getUpdateRate	Retorna a taxa de atualização para o elemento.
removeChild	Remove um filho específico do elemento.
setAccessRights	Define se os clientes possam ler e/ou escrever.
setActive	Define se o elemento está ativo ou não no namespace.
setDataType	Define o tipo de dado deste elemento.
setHasChildren	Define se o elemento contém filhos ou não
setIoMode	Define o modo de E/S deste elemento.
setIsBrowsable	Define se o elemento é um nó ou um item.
setItemId	Define o identificador (ID) do elemento.
setName	Define o nome do elemento.
valueChanged	Altera o valor do cache deste elemento.

A classe *DaTransaction* gerenciar as requisições de transações. Os principais membros da classe *DaTransaction* estão descritos na tabela a seguir.

Tabela 1.6 - Principais métodos da classe DaTransaction

Métodos	Descrição
DaTransaction	Construtor. Cria uma nova transação.
completeRequest	Completa uma transação e remove-a da solicitação.
completeRequests	Completa todas as transações e remove-as das solicitações.
getKey	Retorna o chave associada à transação.
getSession	Retorna a sessão que está operando a transação.
getRequest	Retorna um vetor com todas as requisições contidas na transação.
getType	Retorna o tipo da transação.
handleReadRequests	Chamado para tratar as requisições de leitura.
handleWriteRequests	Chamado para tratar as requisições de escrita.
removeRequest	Remove uma solicitação específica da lista de solicitações.
valuesChanged	Altera o valor do cache de todos os elementos do namespace.

A classe *DaRequest* armazena os dados relativos aos pedidos de escrita e leitura que ocorrem no servidor OPC. Os principais membros da classe *DaRequest* estão descritos na tabela a seguir.

Tabela 1.7 - Principais métodos da classe DaRequest

Métodos	Descrição
DaRequest	Construtor. Cria uma nova requisição.
complete	Completa a requisição atual.
getDaAddressSpaceElement	Retorna o elemento ao qual está sendo solicitado.
getRequestState	Retorna o estado da requisição.
getResult	Retorna o resultado da requisição.
getTransactionKey	Retorna a chave da transação.
getTransactionType	Retorna o tipo da transação.
getValue	Retorna o valor do item com a qualidade e o timestamp.
serRequestState	Define o estado da requisição.
setResult	Define o resultado da requisição.
setTransactionKey	Define a chave da transação.
setValue	Define o valor do item, a qualidade e o timestamp.

A classe *DaSession* gerencia as sessões de comunicação do servidor OPC com o cliente OPC. Os métodos destas classes não são utilizados no código gerado pelo *Wizard* do *toolbox*. Dessa forma, estes métodos não serão descritos nesta monografia.

As enumerações mais importantes do *toolbox* são: *EnumAccessRights Enumeration, EnumIoMode Enumeration, EnumQuality Enumeration, EnumRequestState e EnumTransactionType*.

A enumeração *EnumAccessRights* descreve o direito de acesso dos elementos que compõem o *namespace*. A tabela abaixo lista os três membros possíveis desta enumeração.

Tabela 1.8 – Enumeração EnumAccessRights

Membros	Descrição
READABLE	Os clientes só podem ler os dados de valor do item.
WRITEABLE	Os clientes só escrever ler os dados de valor do item.
READWRITEABLE	O item pode ser lido e alterado pelo cliente.

A enumeração *EnumIoMode* descreve os possíveis mecanismos de atualização dos valores. A tabela abaixo lista os principais membros desta enumeração.

Tabela 1.9 – Enumeração EnumIoMode

Membros	Descrição
NONE	Sem mecanismo de atualização de valores.
POLL	Utiliza o método Polling.
REPORT	Utiliza o método Report.

A enumeração *EnumTransactionType* lista os possíveis tipos de transações. A tabela abaixo lista os membros possíveis desta enumeração.

Tabela 1.10 – Enumeração EnumTransactionType

Membros	Descrição
READ	O cliente pediu por algum valor.
WRITE	O cliente quer escrever algum valor.

A enumeração *EnumQuality* descreve os possíveis estados (ou qualidade) dos valores que o *toolbox* pode utilizar. A tabela abaixo lista os membros possíveis desta enumeração.

Tabela 1.11 - Enumeração EnumQuality

Membros	Descrição
GOOD	Dado válido.
GOOD_LOCAL_OVERRIDE	O valor foi sobrescrito.
BAD	Dado inválido por motivo desconhecido.
BAD_CONFIG_ERROR	Existe um problema no servidor com a configuração.
BAD_NOT_CONNECTED	A conexão da entrada a um dispositivo não foi
BAD_DEVICE_FAILURE	Falha no dispositivo.
BAD_SENSOR_FAILURE	Falha no sensor.
BAD_LAST_KNOWN	Comunicação falhou. O último valor está disponível.
BAD_COMM_FAILURE	Comunicação falhou. . O último valor está indisponível.
BAD_OUT_OF_SERVICE	Fora de serviço.
BAD_WAITING_FOR_INITIAL_DATA	Servidor ainda esperando por dados iniciais.
UNCERTAIN	Não há razão específica para o qual o valor é incerto.
UNCERTAIN_LAST_USABLE	Mostra o último valor utilizável.
UNCERTAIN_SENSOR_CAL	Valor está nos limites ou fora dos limites do sensor.
UNCERTAIN_EGU_EXCEEDED	O valor retornado está fora dos limites deste parâmetro.
UNCERTAIN_SUB_NORMAL	Valor originado de várias fontes, mas nem todas são boas.
QUALITY_NOT_SET	A qualidade não está definida.

A enumeração *EnumRequestState* lista os possíveis estados das requisições. A tabela abaixo lista os membros possíveis desta enumeração.

Tabela 1.12 – Enumeração EnumRequestState

Membros	Descrição
CREATED	A requisição acabou de ser criada.
PENDING	A requisição está prestes a ser processada.
COMPLETED	A requisição foi processada e completada.

2. Principais Diagramas de Classe do toolbox

O aplicativo do servidor pode especificar os valores dos elementos através da classe *DaAddessSpaceElement* e as fontes de alarmes através da classe *AeAddressSpaceElement* através da construção de uma árvore de elementos abaixo de um nó de raiz. Essa estrutura pode ser mais bem compreendida através dos diagramas de classe UML. O diagrama de classe abaixo inclui as classes para execução do servidor OPC e criação do *namespace*.

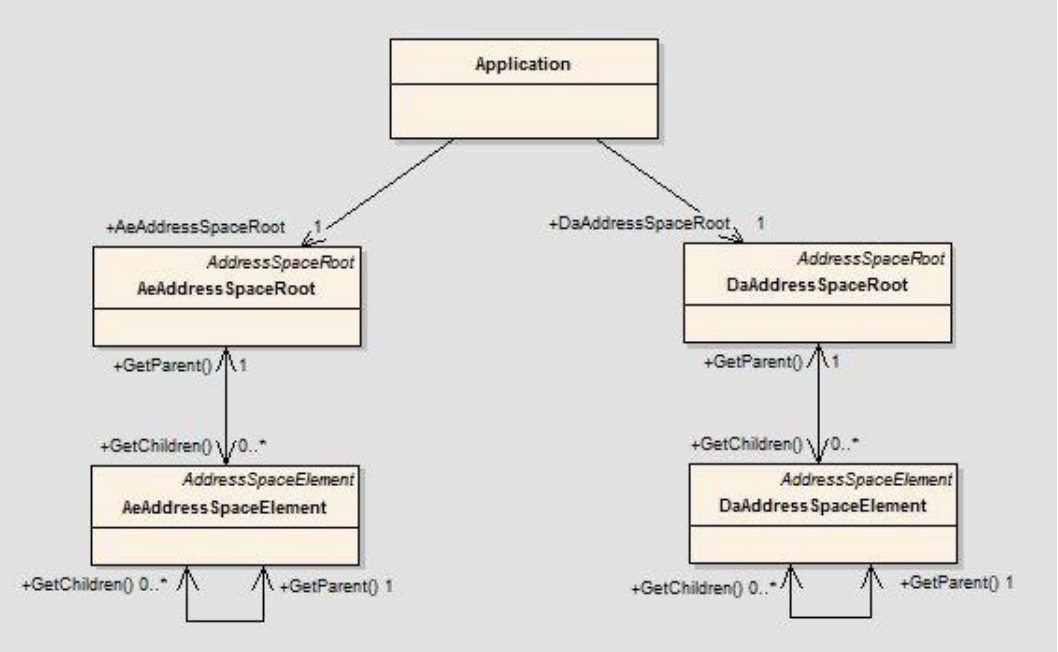


Figura 2.1 - Diagrama de classes UML das pricipais classes do toolbox do servidor OPC

O aplicativo do servidor pode especificar, se as operações de leitura sobre o valor devem ser realizadas pelo mecanismo *polling* ou pelo mecanismo *report* (estes mecanismos serão descritos na sessão a seguir). Se um valor for requisitado através do mecanismo *polling*, o aplicativo cria uma solicitação através da classe *DaRequest* para a leitura do valor. Todas as requisições de chamadas de leitura de um cliente ou criadas pelo toolkit para atualização de um cadastro são coletados em uma transação de leitura ou escrita através da classe *DaTransaction*. O diagrama de classe abaixo inclui as classes para tratar as requisições e transações no *toolkit*.

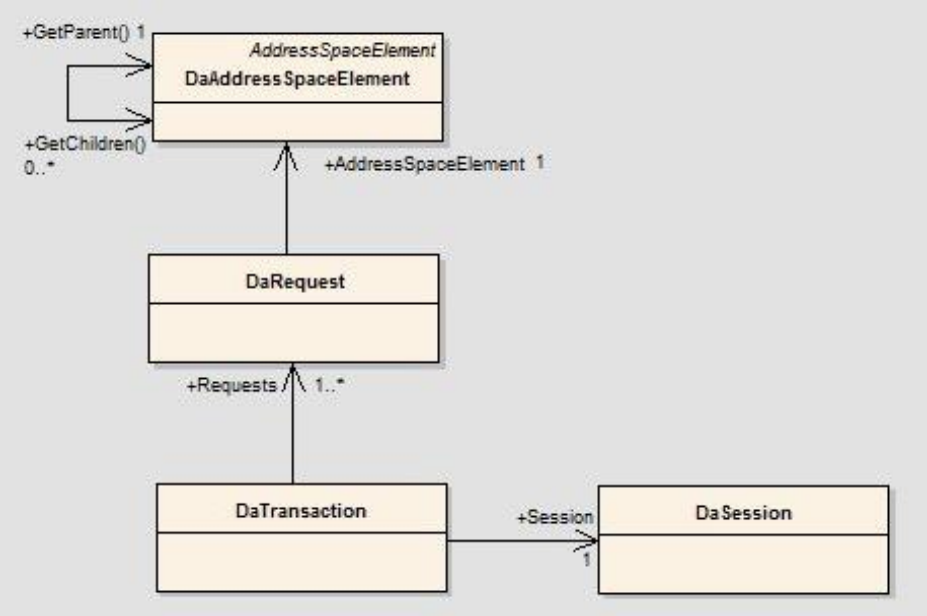


Figura 2.2 - Diagrama de classes com as classes que tratam de requisições e transações no toolkit

3. Mecanismos de Atualização de Valores do Toolbox

A especificação de acesso de dados OPC determina que os servidores OPC verifiquem se os valores dos itens ativos em grupos ativos mudaram de acordo com a taxa de atualização especificada pelo cliente OPC. Esta verificação é realizada independentemente da existência de uma conexão de retorno chamada do cliente para o servidor através do qual as mudanças poderiam ser transmitidas. O *toolbox* dispõe de dois mecanismos de atualização de valores.

- *Pooling* – o *toolkit* implementa um mecanismo ativo para a leitura dos valores dos elementos através de um "cadastro" feito no lado do cliente. Quando um item é ativado pelo cliente, uma solicitação é gerada dentro de uma transação que representa um cadastro do lado do cliente. Para atualizar os itens de um cadastro ativo, a transação é iniciada toda vez que acabar o período de atualização do servidor. Depois que todas as solicitações da operação forem processadas ou o período de atualização do grupo seja atingido, os valores alterados dos itens nos cadastros são transmitidos para todos os clientes OPC que tenham estabelecido conexões de retorno com o servidor. Todas as requisições recebidas dentro de uma transação devem ser concluídas.

- *Report* - O *toolkit* não lê ativamente os valores dos elementos do *namespace*. O servidor é responsável por comunicar os valores de processo modificado para o *toolkit*. Neste caso, o *toolkit* não gera transações ou solicitações. Os valores de *cache* devem ser mudados dentro do aplicativo de servidor. Uma função do *toolkit* (*DaAddressSpaceElement.ValueChanged* ou *DaAddressSpaceRoot.ValuesChanged*) fornecerá o valor de *cache* para todos os cadastros. A transação será gerada por solicitações de leitura e de escrita. Depois de finalizado o período de atualização do cliente, um dos métodos descritos acima verifica se o aplicativo de servidor tem sinalizado mudanças de valores para este item. Os valores dos itens ativos modificados são enviados para todos os clientes OPC que tenham estabelecido conexões de retorno com o servidor**.**

Anexo

Anexo B – Documentação Técnica do Servidor OPC para o SISAL

SERVIDOR OPC

SISAL

Documentação Técnica
Servidor OPC SISAL V 1.00

Julho 2010
Documentação Técnica Servidor OPC SISAL V100 – Revisão 0

Apresentação

Esta documentação técnica foi elaborada pelo aluno de graduação João Teixeira de Carvalho Neto com o objetivo de documentar o *software* Servidor OPC para o SISAL de propriedade da UFRN.

Qualquer dúvida relativa às informações contidas neste documento pode ser esclarecida diretamente através dos telefones:

- (84) 9138-1668
- (84) 8736-1668

Desenvolvimento

Universidade Federal do Rio Grande do Norte – UFRN
Departamento de Computação e Automação – DCA
Email: adelardo@dca.ufrn.br
Email: joaoteixeiracanon828@gmail.com

Julho de 2010

Documentação Técnica Servidor OPC para o SISAL V100

Sumário

1. Apresentação

O Servidor OPC para o SISAL tem como objetivo criar uma interface de comunicação entre um cliente OPC e o servidor SISAL, da forma que o servidor OPC leia os dados referentes à lista de poços disponíveis no servidor SISAL, e disponibilize estes dados para os clientes OPC conectados a ele, utilizando o padrão de comunicação estabelecido pela tecnologia OPC. O aplicativo foi desenvolvido na plataforma *Visual Studio C++ 2005* e foi utilizado o *Softing OPC Toolbox v4.22* para implementação das sua funcionalidades. A versão atual do Servidor OPC para o SISAL (v1.00) contém as seguintes funcionalidades, vantagens e limitações:

- Leitura de variáveis dos poços, contidos no servidor SISAL, em tempo real.
- Comunicação entre cliente OPC e servidor OPC pode ser feita via *Ethernet*, possibilitando a instalação do servidor OPC em campo e do cliente OPC em escritório.
- Criação de um *namespace* dinâmico, onde seu tamanho varia de acordo com a quantidade de poços contidos no servidor SISAL.
- Leitura das seguintes variáveis dos poços: *cod_poço, Estação, End_Controlador* e *Grau API*.
- Tempo de execução limitado a 90 minutos devido à versão do *Softing OPC Toolbox v4.22* ser demonstrativa.

A arquitetura do funcionamento adotado no Servidor OPC para o SISAL e esta documentação técnica foi objeto de trabalho de conclusão de curso do curso de Engenharia de Computação na UFRN e foi projetado de modo a permitir a incorporação da especificação de alarme e eventos do protocolo OPC (*OPC Alarm & Events Specification*).

2. Arquitetura do Sistema

O Servidor OPC para o SISAL funciona coletando as variáveis dos poços disponibilizadas pelo servidor SISAL e disponibilizando estas variáveis em formas de *tags* para os clientes OPC. Os clientes OPC representam a parte responsável por apresentar as informações de supervisão aos usuários através do padrão especificado pela tecnologia OPC. É dos clientes que partem as requisições de dados para o servidor e são eles que devem utilizar essa informação no processo de monitoramento. As informações em questão referem-se às variáveis dos poços. O servidor OPC, além de coletar as variáveis dos poços, é responsável pela criação de um *namespace* dinâmico com nós e itens, onde os nós representam os poços que estão sendo monitorados e os itens representam as variáveis dos poços. O servidor OPC deve receber as requisições dos clientes, e devolver as respostas aos clientes depois de certo tempo estabelecido pelo cliente. O modo de funcionamento do Servidor OPC para o SISAL é discutida na sessão referente ao funcionamento do aplicativo.

A comunicação entre o servidor OPC e o servidor SISAL é dada através de uma interface de comunicação com funções que implementam a mesma comunicação PCI que é utilizada pelo cliente SISAL para coletar os dados do servidor SISAL, ou seja, é utilizado parte do código do cliente SISAL que implementa uma interface de comunicação para disponibilizar a lista de poços e variáveis para o servidor OPC. Esta interface é realizada através do acesso de funções em uma biblioteca de ligação dinâmica (DLL) implementada na plataforma de desenvolvimento *Borland C++ Builder 5.0* chamada "*pci.dll*". Os arquivos do cliente SISAL com extensão "*cpp*" e "*h*" utilizados para implementação da DLL foram: *md5, Unt_Status, Unt_ClassePoco, Unt_ClasseSistema, Unt_PCI, Unt_FuncoesPCI, Unt_Principal, Unt_StringList, Unt_ProcessoPrincipal, Unt_ProcessoReceptor e Unt_ProcessoVerificador*. Foi utilizado também o arquivo *Unt_Buffercircular.h*. O arquivo chamado *Unit1.cpp* contém as funções que serão exportadas da DLL para obtenção e atualização da lista de poços. Mais detalhes sobre os pontos referentes às funcionalidades dos arquivos serão mostrados na sessão referente à descrição dos arquivos do projeto.

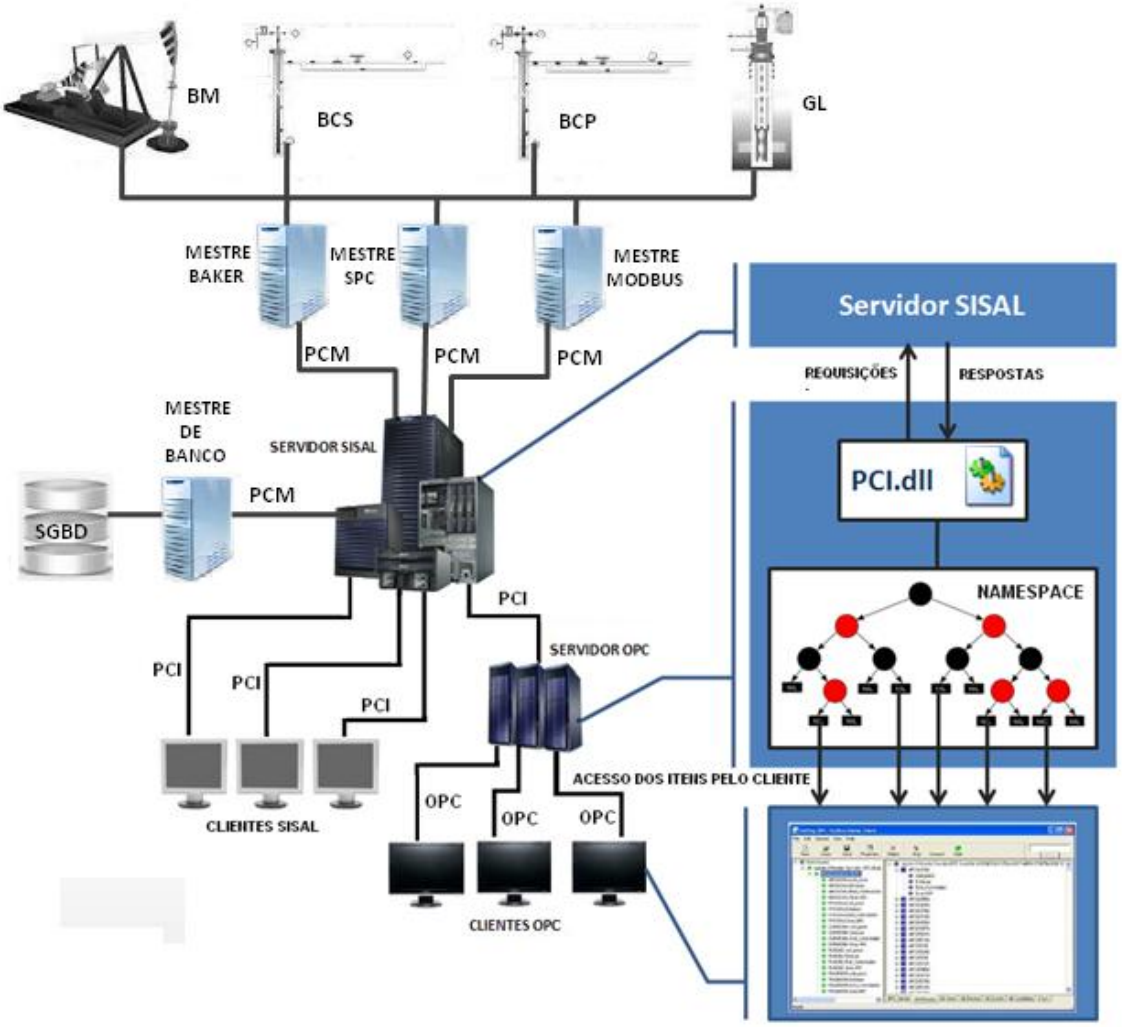


Figura 2.1 - Arquitetura do sistema

3. Funcionamento do Aplicativo

Esta sessão é responsável por descrever o modo de funcionamento do programa. O programa é executado em segundo plano não sendo visível sua interface ao usuário. Os passos a seguir são os principais passos realizados entre momento em que o programa é inicializado e após o momento em que o programa é fechado.

Passos do funcionamento do aplicativo

- Passo 1 – Criação de uma nova instancia do objeto *OpcServer*, inicialização parâmetros do servidor através da função *OpcServer.initialize()* e início de execução do servidor através da função *OpcServer.start()*.
- Passo 2 – Criação uma nova instância da interface entre o servidor OPC e o servidor SISAL *Unt_PciInterface* e inicialização da comunicação com servidor SISAL através da função *Unt_PciInterface.init()*.
  - Passo 2.1 – Verificação da versão do servidor SISAL e *login* com o servidor SISAL através da função *StartFunc* importada da DLL "*pci.dll*".
  - Passo 2.2 – Obtenção da lista de poços através da função *getVecPocosFunc* importada da DLL "*pci.dll*".
  - Passo 2.3 – Atualização da lista de poços através da função *UpdateWell* importada da DLL "*pci.dll*".
- Passo 3 – Armazenamento da lista de poços em um vetor de poços (classe *Poco*) através da função *Unt_PciInterface.UpdateVec()*.
- Passo 4 – Servidor sinaliza que está pronto para construção do *namespac*e através da função *OpcServer.ready()*.

- Passo 5 – Inicialização de threads para simulação de valores na chamada da função *startSimulationThread()*.

- Passo 6 – Criação do *namespace* e atualização dos valores de seus itens através da função *OpcServer.UpdateWells()*.

  - Passo 6.1 - Para a criação do espaço de nomes e atualização dos valores de seus itens é necessário primeiramente varrer o vetor de poços, adicionando um poço com suas variáveis por vez no *namespace* para cada iteração no vetor, através da função *OpcServer.AddWell(const Poco &p)*.

  - Passo 6.2 - Para adicionar um novo poço, primeiramente um *namespace* temporário é criado através do construtor *creator* da classe *MyCreator*. O construtor cria uma nova instância do objeto *MyDaAddressSpaceElement*. O construtor é responsável por criar um nó para representar o poço e a quantidade de itens deste nó é equivalente a quantidade das *tags* que foram definidas (no caso desta versão do servidor OPC, foram definidas quatro *tags* – *cod_poco, Estacao, End_Controlador* e *Grau API*) – neste momento os nós e itens estão "em branco" sem nome e sem valores definidos.

    - Passo 6.2.1 - Quanto aos direitos de acesso, os itens foram configurados para serem somente lidos, através da função *DaAddressSpaceElement.setAccessRights(READ)*.

    - Passo 6.2.2 - A determinação de se o elemento é um nó ou um item, através da função *DaAddressSpaceElement.setIsBrowsable()*.

    - Passo 6.2.3 – A determinação do nome dos elementos, através da função *DaAddressSpaceElement.setName(string nome)* da forma que os nós representam os poços e os itens representam as variáveis dos poços.

    - Passo 6.2.4 – Quanto ao tipo de dado, todos os elementos do espaço de nomes são do tipo string definidos pela função *DaAddressSpaceElement.setType(VT_BSTR)*.

    - Passo 6.2.5 – Todos os elementos do *namespace* do servidor OPC são atualizados segundo o mecanismo de atualização de valor *polling*, através da função *DaAddressSpaceElement.setIoMode(POLL)*.

- Passo 6.3 - Após a construção do nó e dos itens, estes itens são conectados ao nó (estes itens são configurados como filhos do nó), através da função *DaAddressSpaceElement.addChild(DaAddressSpaceElement elemento)* e são armazenados em um mapa de elementos no qual a chave para cada elemento do mapa é o código do poço (*cod_poco*).

- Passo 6.4 - Após cada adição de poço, a função define os valores para aqueles itens "em branco" que foram criados na iteração através das funções:

  - Passo 6.4.1 – Definição do valor do item Estação, através da função *OpcServer.setValue(MyDaAddressSpaceElement ele, EnumQuality qualidade, const char * valor)*;
  - Passo 6.4.2 – Definição do valor do item Grau API, através da função *OpcServer.setValue(MyDaAddressSpaceElement ele, EnumQuality qualidade, float * valor)*;
  - Passo 6.4.3 – Definição do valor dos itens *End_Controlador* e *cod_poco*, através da função *OpcServer.setValue(MyDaAddressSpaceElement ele, EnumQuality qualidade, int * valor)*;

  A adição de poços no *namespace* é realizada até que o iterador chegue ao final do vetor de poços.

- Passo 6.5 - Quando o iterador chegar ao final do vetor, ou seja, não tiver mais poços a serem adicionados até o momento da atual atualização, os valores são atualizados através da função *Unt_PciInterface.UpdateVec()* até que a aplicação seja finalizada.

- Passo 6.6 - Se um novo poço for adicionado ao sistema e conseqüentemente for adicionado ao vetor de poços, a função *OpcServer.UpdateWells()* ficará responsável por tratar de criar um novo nó com itens referentes a um poço com suas variáveis e adicioná-los ao espaço de nomes do servidor OPC.

- Passo 7 – A atualização de todos os valores do *namespace* é dada de acordo com o mecanismo de *polling*. Este passo não pode ser debugado, pois ele acontece no momento quando o cliente requisita alguma *tag* ao *namespace* do servidor.

- Passo 7.1 – Quando um item é requisitado pelo cliente é realizado um cadastro do servidor no cliente através de uma solicitação para uma transação.

- Passo 7.2 – Na medida em que os valores estiverem disponíveis no servidor ou o tempo de atualização do servidor chegar ao final, os valores são atualizados nos cadastros através de uma transação de escrita.

- Passo 7.3 – Quando o tempo de atualização do cliente for finalizado, os valores dos cadastros são utilizados para atualizar os dados nos cliente.

Para uma operação de leitura em um item do espaço de nomes é utilizada a função *DaTransaction.handleReadRequests()*. Para uma operação de escrita em um item do espaço de nomes é utilizada a função *DaTransaction.handleWriteRequests()*. Os itens que compõem o *namespace* nesta versão do programa só têm direito de acesso a leitura.

- Passo 8 – Quando o programa é fechado pelo usuário, a thread de simulação de valores é finalizada, o servidor se prepara para ser desligado, fechando suas funcionalidades, através da função *OpcServer.stop()* e, após isso, é desativado chamando a função *OpcServer.terminate()*.

4. *Diagramas de Classe do Servidor OPC SISAL V 1.00*

Esta sessão é responsável pela descrição das principais classes do aplicativo, e por ilustrar os diagramas de classe referentes ao código do servidor OPC para o SISAL. Os diagramas foram desenvolvidos no *toolbox class designer* do *Visual Studio C++ 2005*. As descrições das classes, que estão presentes no código gerado pelo *Wizard* e utilizadas pelo *toolbox* do servidor OPC para o SISAL estão listadas abaixo:

- Classe *Application* – Responsável por fornecer métodos para inicializar e encerrar o servidor, detecção e tratamento de objetos do *toolbox* do servidor OPC, além de manipulação de entrada e saída.

- Classe *MyCreator* – Herda os membros da classe *Creator*.

- Classe *Creator* – Responsável pela criação dos principais objetos do *toolbox* do servidor OPC.

- Classe *OpcServer* – Responsável pela inicialização do servidor OPC, construção dos *namespace* da especificação de acesso de dados do servidor OPC e atualização dos valores de seus itens, criação do *areaspace* e do *eventspace* (espaço de eventos) da especificação de alarme e eventos do servidor OPC e atualização de seus eventos, dentre outras funcionalidades.

- Classe *Unt_PciInterface* – Responsável pela comunicação do servidor OPC com o servidor SISAL e armazenamento dos Poços por um vetor de objetos da classe *Unt_ClassePoco*.

- Classe *Poco* – Responsável criar uma nova instância com as variáveis do poço.

- Classe *MyRequest* – Herda os membros da classe *DaRequest*.

- Classe *DaRequest* – Responsável pelo tratamento das requisições originadas dos clientes OPC.

- Classe *MyTransactions* - Herda os membros da classe *DaTransactions*.

- Classe *DaTransaction*s – Responsável pelo tratamento das transações de leitura e escrita entre o cliente OPC e o servidor .

- Classe *MyDaAddressSpaceRoot* – Herda os membros da classe *DaAddressSpaceRoot*.

- Classe *DaAddressSpaceRoot* – Responsável pela definição e configuração do nó raiz do espaço de nomes do servidor OPC. Herda os membros da classe *AddressSpaceRoot*.

- Classe *AddressSpaceRoot* – Representa por tratar a definição e configuração do nó raiz das especificações DA e AE. Herda os membros da classe *IAddressSpaceElement*.

- Classe *MyDaAddressSpaceElement* – Herda os membros da classe *DaAddressSpaceElement*.

- Classe *DaAddressSpaceElement* – Responsável pela definição e configuração dos elementos do espaço de nomes do servidor OPC. Herda os membros da classe *AddressSpaceElement*.

- Classe *AddressSpaceElement* – Representa o componente base para definição e configuração de elementos das especificações DA e AE. Herda os membros da classe *IAddressSpaceElement*.

- Classe *MyAeAddressSpaceElement* – Herda os membros da classe *AeAddressSpaceElement*.

- Classe *AeAddressSpaceElement* – Responsável pela definição e configuração de elementos do espaço de nomes referentes à especificação de alarme e eventos.

- Classe *IAddressSpaceElement* – Interface responsável por tratar do *namespace* (nó raiz e itens) das especificações DA e AE.

- Classe *DaSession* – Representa a sessão de comunicação do servidor OPC com o cliente OPC.

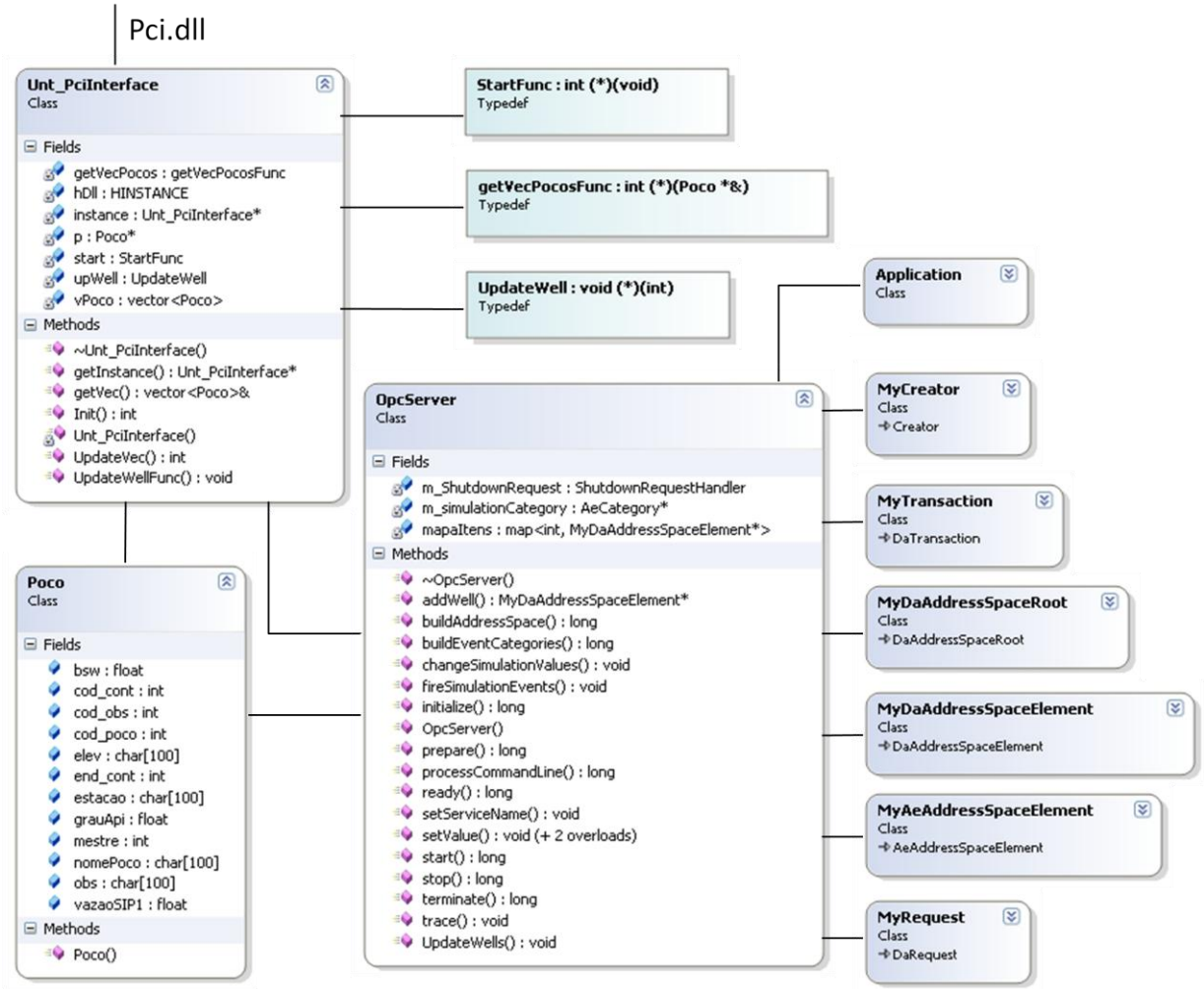
- Classe *ValueQT* – Representa o valor de uma variável do processo com a sua qualidade e o tempo que foi obtido pelo dispositivo ou pelo *cache* do servidor. Herda os membros da classe *ValueData*.

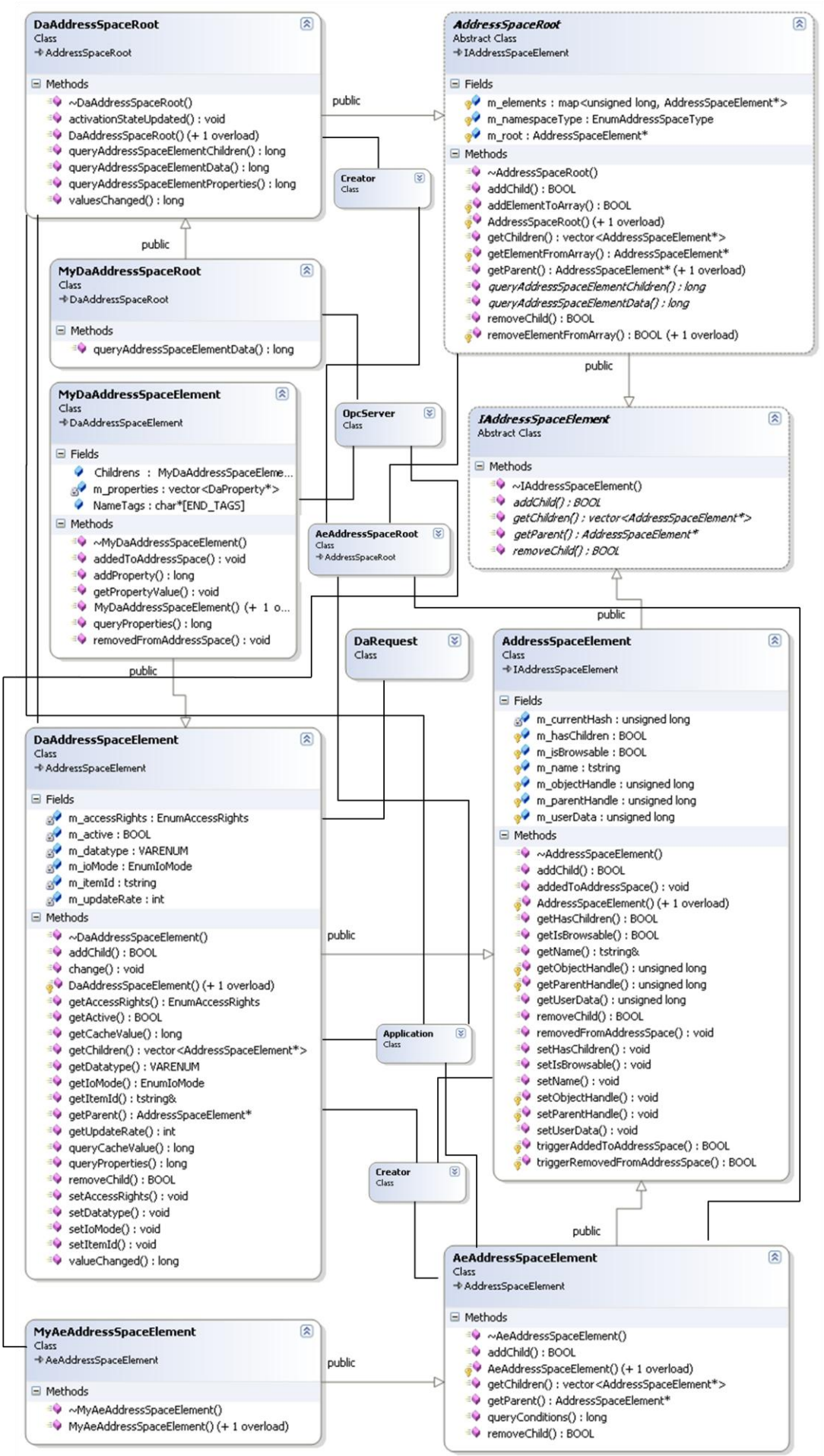
- Classe *ValueData* – Representa somente o valor de uma variável do processo fornecidos pelo servidor OPC.

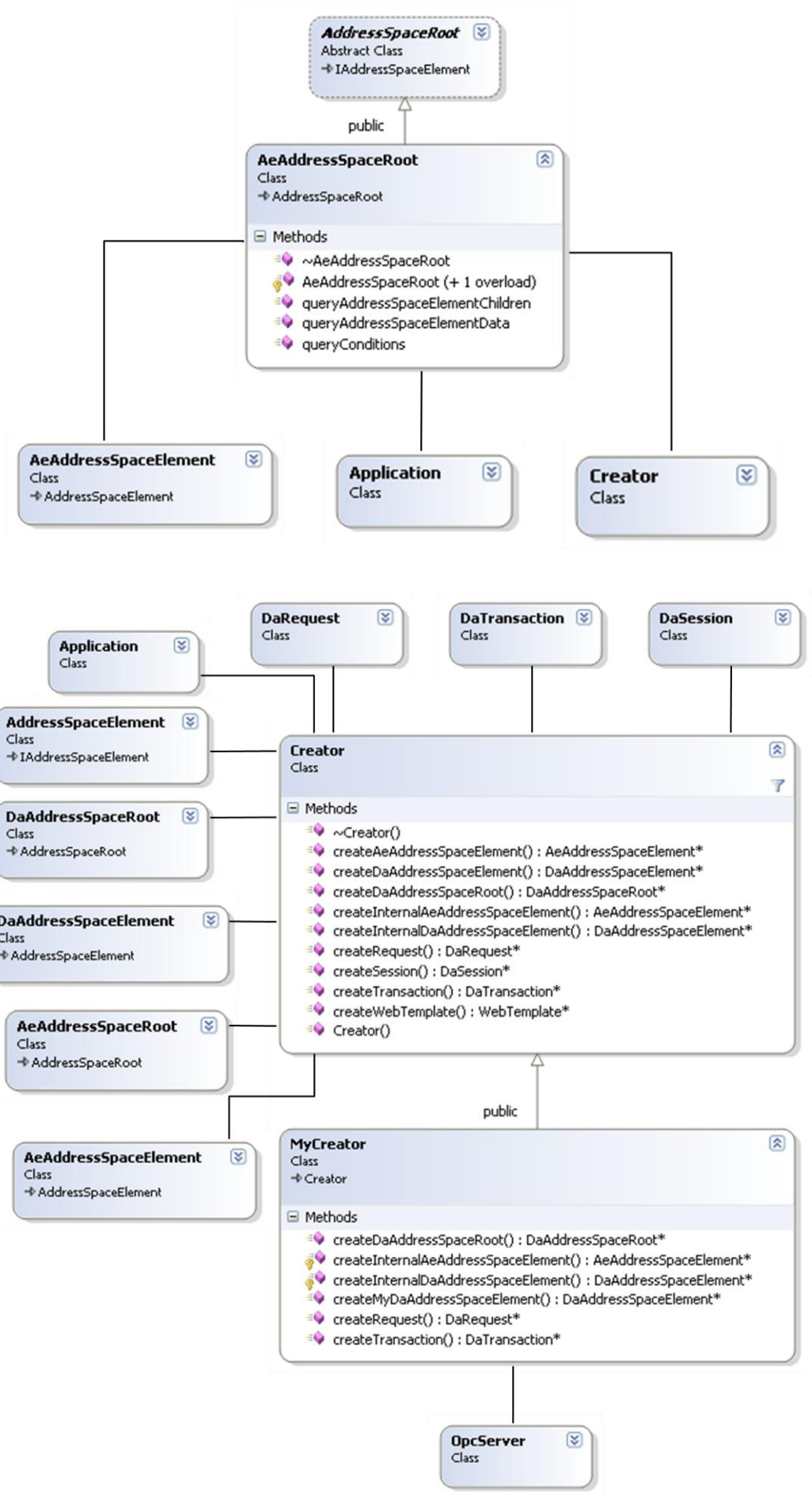
- Classe *DateTime* – Representa um instante no tempo, normalmente expressado com a data e a hora.

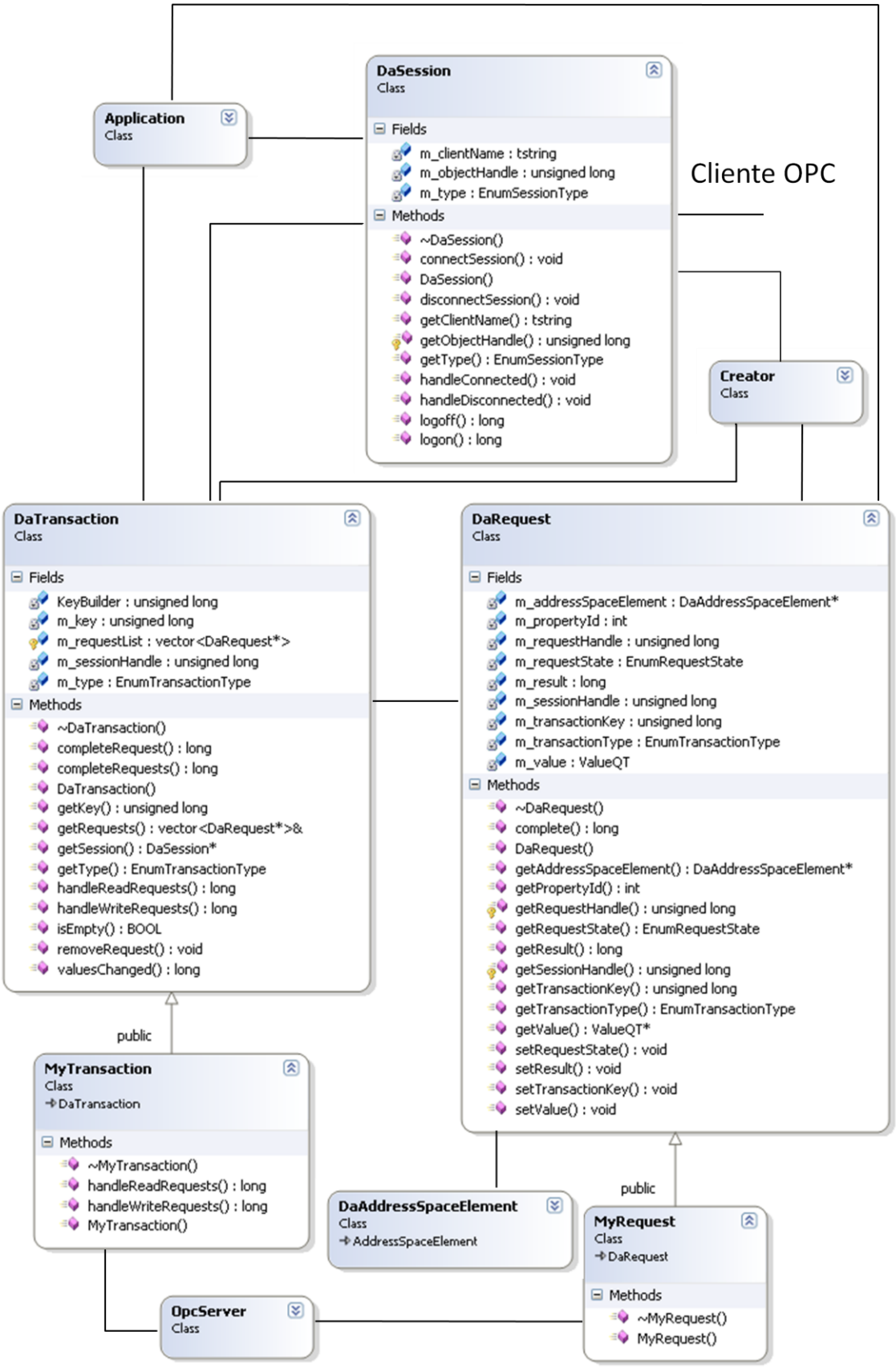
- Classe *Variant* – Representa o tipo de dado *VARIANT*.

A seguir são ilustrados os diagramas de classe do código do servidor OPC para o SISAL, e do código referente às bibliotecas do toolbox utilizadas.

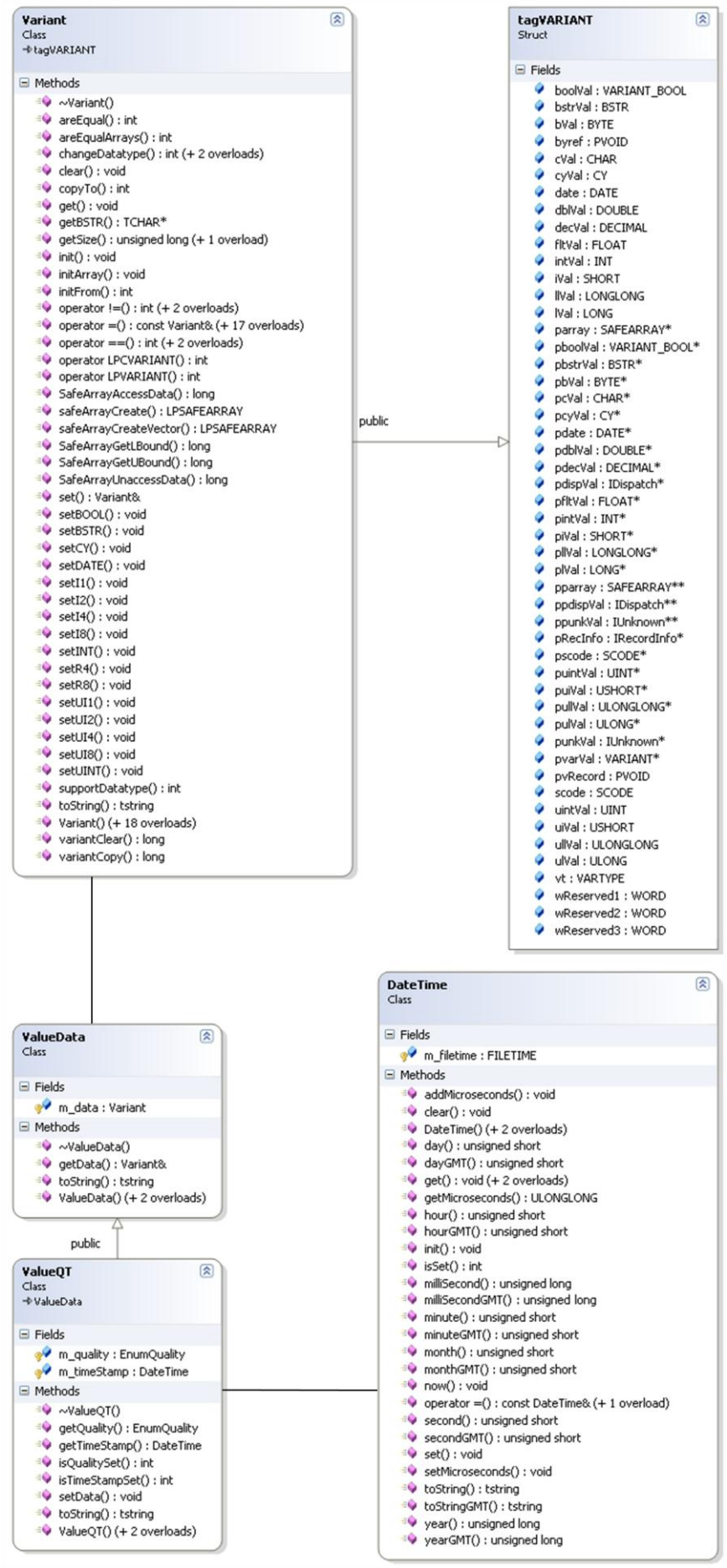



DaAddressSpaceRoot
Class
AddressSpaceRoot
Methods
~DaAddressSpaceRoot()
activationStateUpdated() : void
DaAddressSpaceRoot() (+ 1 overload)
queryAddressSpaceElementChildren() : long
queryAddressSpaceElementData() : long
queryAddressSpaceElementProperties() : long
valuesChanged() : long
public
AddressSpaceRoot
Abstract Class
IAddressSpaceElement
Fields
m_elements : map<unsigned long, AddressSpaceElement*>
m_namespaceType : EnumAddressSpaceType
m_root : AddressSpaceElement*
Methods
~AddressSpaceRoot()
addChild() : BOOL
addElementToArray() : BOOL
AddressSpaceRoot() (+ 1 overload)
getChildren() : vector<AddressSpaceElement*>
getElementFromArray() : AddressSpaceElement*
getParent() : AddressSpaceElement* (+ 1 overload)
queryAddressSpaceElementChildren() : long
queryAddressSpaceElementData() : long
removeChild() : BOOL
removeElementFromArray() : BOOL (+ 1 overload)
Creator
Class
public
MyDaAddressSpaceRoot
Class
DaAddressSpaceRoot
Methods
queryAddressSpaceElementData() : long
public
MyDaAddressSpaceElement
Class
DaAddressSpaceElement
Fields
Childrens : MyDaAddressSpaceEleme...
m_properties : vector<DaProperty*>
NameTags : char*[END_TAGS]
Methods
~MyDaAddressSpaceElement()
addedToAddressSpace() : void
addProperty() : long
getPropertyValue() : void
MyDaAddressSpaceElement() (+ 1 o...
queryProperties() : long
removedFromAddressSpace() : void
OpcServer
Class
IAddressSpaceElement
Abstract Class
Methods
~IAddressSpaceElement()
addChild() : BOOL
getChildren() : vector<AddressSpaceElement*>
getParent() : AddressSpaceElement*
removeChild() : BOOL
public
AeAddressSpaceRoot
Class
AddressSpaceRoot
DaRequest
Class
public
AddressSpaceElement
Class
IAddressSpaceElement
Fields
m_currentHash : unsigned long
m_hasChildren : BOOL
m_isBrowsable : BOOL
m_name : tstring
m_objectHandle : unsigned long
m_parentHandle : unsigned long
m_userData : unsigned long
Methods
~AddressSpaceElement()
addChild() : BOOL
addedToAddressSpace() : void
AddressSpaceElement() (+ 1 overload)
getHasChildren() : BOOL
getIsBrowsable() : BOOL
getName() : tstring&
getObjectHandle() : unsigned long
getParentHandle() : unsigned long
getUserData() : unsigned long
removeChild() : BOOL
removedFromAddressSpace() : void
setHasChildren() : void
setIsBrowsable() : void
setName() : void
setObjectHandle() : void
setParentHandle() : void
setUserData() : void
triggerAddedToAddressSpace() : BOOL
triggerRemovedFromAddressSpace() : BOOL
public
DaAddressSpaceElement
Class
AddressSpaceElement
Fields
m_accessRights : EnumAccessRights
m_active : BOOL
m_datatype : VARENUM
m_ioMode : EnumIoMode
m_itemId : tstring
m_updateRate : int
Methods
~DaAddressSpaceElement()
addChild() : BOOL
change() : void
DaAddressSpaceElement() (+ 1 overload)
getAccessRights() : EnumAccessRights
getActive() : BOOL
getCacheValue() : long
getChildren() : vector<AddressSpaceElement*>
getDatatype() : VARENUM
getIoMode() : EnumIoMode
getItemId() : tstring&
getParent() : AddressSpaceElement*
getUpdateRate() : int
queryCacheValue() : long
queryProperties() : long
removeChild() : BOOL
setAccessRights() : void
setDatatype() : void
setIoMode() : void
setItemId() : void
valueChanged() : long
public
Application
Class
Creator
Class
public
AeAddressSpaceElement
Class
AddressSpaceElement
Methods
~AeAddressSpaceElement()
addChild() : BOOL
AeAddressSpaceElement() (+ 1 overload)
getChildren() : vector<AddressSpaceElement*>
getParent() : AddressSpaceElement*
queryConditions() : long
removeChild() : BOOL
MyAeAddressSpaceElement
Class
AeAddressSpaceElement
Methods
~MyAeAddressSpaceElement()
MyAeAddressSpaceElement() (+ 1 overload)
public


AddressSpaceRoot
Abstract Class
IAddressSpaceElement
public
AeAddressSpaceRoot
Class
AddressSpaceRoot
Methods
~AeAddressSpaceRoot
AeAddressSpaceRoot (+ 1 overload)
queryAddressSpaceElementChildren
queryAddressSpaceElementData
queryConditions
AeAddressSpaceElement
Class
AddressSpaceElement
Application
Class
Creator
Class
DaRequest
Class
DaTransaction
Class
DaSession
Class
Application
Class
AddressSpaceElement
Class
IAddressSpaceElement
DaAddressSpaceRoot
Class
AddressSpaceRoot
DaAddressSpaceElement
Class
AddressSpaceElement
AeAddressSpaceRoot
Class
AddressSpaceRoot
Creator
Class
Methods
~Creator()
createAeAddressSpaceElement() : AeAddressSpaceElement*
createDaAddressSpaceElement() : DaAddressSpaceElement*
createDaAddressSpaceRoot() : DaAddressSpaceRoot*
createInternalAeAddressSpaceElement() : AeAddressSpaceElement*
createInternalDaAddressSpaceElement() : DaAddressSpaceElement*
createRequest() : DaRequest*
createSession() : DaSession*
createTransaction() : DaTransaction*
createWebTemplate() : WebTemplate*
Creator()
public
AeAddressSpaceElement
Class
AddressSpaceElement
MyCreator
Class
Creator
Methods
createDaAddressSpaceRoot() : DaAddressSpaceRoot*
createInternalAeAddressSpaceElement() : AeAddressSpaceElement*
createInternalDaAddressSpaceElement() : DaAddressSpaceElement*
createMyDaAddressSpaceElement() : DaAddressSpaceElement*
createRequest() : DaRequest*
createTransaction() : DaTransaction*
OpcServer
Class


Application
Class
DaSession
Class
Fields
m_clientName : tstring
m_objectHandle : unsigned long
m_type : EnumSessionType
Methods
~DaSession()
connectSession() : void
DaSession()
disconnectSession() : void
getClientName() : tstring
getObjectHandle() : unsigned long
getType() : EnumSessionType
handleConnected() : void
handleDisconnected() : void
logoff() : long
logon() : long
Cliente OPC
Creator
Class
DaTransaction
Class
Fields
KeyBuilder : unsigned long
m_key : unsigned long
m_requestList : vector<DaRequest*>
m_sessionHandle : unsigned long
m_type : EnumTransactionType
Methods
~DaTransaction()
completeRequest() : long
completeRequests() : long
DaTransaction()
getKey() : unsigned long
getRequests() : vector<DaRequest*>&
getSession() : DaSession*
getType() : EnumTransactionType
handleReadRequests() : long
handleWriteRequests() : long
isEmpty() : BOOL
removeRequest() : void
valuesChanged() : long
DaRequest
Class
Fields
m_addressSpaceElement : DaAddressSpaceElement*
m_propertyId : int
m_requestHandle : unsigned long
m_requestState : EnumRequestState
m_result : long
m_sessionHandle : unsigned long
m_transactionKey : unsigned long
m_transactionType : EnumTransactionType
m_value : ValueQT
Methods
~DaRequest()
complete() : long
DaRequest()
getAddressSpaceElement() : DaAddressSpaceElement*
getPropertyId() : int
getRequestHandle() : unsigned long
getRequestState() : EnumRequestState
getResult() : long
getSessionHandle() : unsigned long
getTransactionKey() : unsigned long
getTransactionType() : EnumTransactionType
getValue() : ValueQT*
setRequestState() : void
setResult() : void
setTransactionKey() : void
setValue() : void
public
MyTransaction
Class
DaTransaction
Methods
~MyTransaction()
handleReadRequests() : long
handleWriteRequests() : long
MyTransaction()
DaAddressSpaceElement
Class
AddressSpaceElement
public
MyRequest
Class
DaRequest
Methods
~MyRequest()
MyRequest()
OpcServer
Class


Application
Class
Fields
m_addressSpaceDelimiter : TCHAR
m_AeAddressSpaceRoot : AeAddressSpaceRoot*
m_appType : EnumApplicationType
m_buildNumber : short
m_categoryList : map<unsigned long, AeCategory*>
m_clientCheckPeriod : long
m_clsidAE : tstring
m_clsidDA : tstring
m_conditionList : map<unsigned long, AeCondition*>
m_creator : Creator*
m_DaAddressSpaceRoot : DaAddressSpaceRoot*
m_description : tstring
m_instance : Application*
m_ipPortHTTP : unsigned short
m_majorVersion : short
m_minorVersion : short
m_minUpdateRateDA : long
m_patchVersion : short
m_progIdAE : tstring
m_progIdDA : tstring
m_propertyDelimiter : TCHAR
m_serviceName : tstring
m_sessionList : map<unsigned long, DaSession*>
m_supportDisableConditions : unsigned char
m_tpCredentialsNumber : long
m_tpPasswords : vector<tstring>
m_tpPort : unsigned short
m_tpUsers : vector<tstring>
m_transactionJanitor : Mutex
m_transactionList : map<unsigned long, DaTransaction*>
m_urlDA : tstring
m_vendorInfo : tstring
m_verIndProgIdAE : tstring
m_verIndProgIdDA : tstring
m_versionOTB : short
m_webAdministratorPassword : tstring
m_webOperatorPassword : tstring
m_webRootDirectory : tstring
m_webRootFile : tstring
m_webTemplate : WebTemplate*
ShutdownRequest : ShutdownRequestHandler
TraceCompleted : TraceEventHandler
Methods
~Application()
activate() : long
addAeCategory() : long
addSession() : void
addTpCredentials() : long
addTransaction() : void
Application()
changeCondition() : long
changeConditions() : long
enableTracing() : void
findTransaction() : DaTransaction*
fireEvents() : long
forceDemoVersion() : long
getAddressSpaceDelimiter() : TCHAR
getAeAddressSpaceRoot() : AeAddressSpaceRoot*
getAeConditionList() : vector<AeCondition*>
getAppType() : EnumApplicationType
getBuildNumber() : short
getClientCheckPeriod() : long
getClsidAe() : tstring&
getClsidDa() : tstring&
getCreator() : Creator*
getDaAddressSpaceRoot() : DaAddressSpaceRoot*
getDescription() : tstring&
getIpPortHTTP() : unsigned short
getMajorVersion() : short
getMinorVersion() : short
getMinUpdateRateDa() : long
getPatchVersion() : short
getProgIdAe() : tstring&
getProgIdDa() : tstring&
getPropertyDelimiter() : TCHAR
getServiceName() : tstring&
getSession() : DaSession*
getSupportDisableConditions() : unsigned char
getTpPort() : unsigned short
getUrlDa() : tstring&
getVendorInfo() : tstring&
getVerIndProgIdAe() : tstring&
getVerIndProgIdDa() : tstring&
getVersionOtb() : unsigned short
getWebAdministratorPassword() : tstring&
getWebOperatorPassword() : tstring&
getWebRootDirectory() : tstring&
getWebRootFile() : tstring&
getWebTemplate() : WebTemplate*
initialize() : long
Instance() : Application*
processCommandLine() : long
ready() : long
Release() : void
releaseCondition() : void
releaseTransaction() : void
removeSession() : void
setAddressSpaceDelimiter() : void
setAppType() : void
setBuildNumber() : void
setClientCheckPeriod() : void
setClsidAe() : void
setClsidDa() : void
setDescription() : void
setIpPortHTTP() : void
setMajorVersion() : void
setMinorVersion() : void
setMinUpdateRateDa() : void
setPatchVersion() : void
setProgIdAe() : void
setProgIdDa() : void
setPropertyDelimiter() : void
setServiceName() : void
setSupportDisableConditions() : void
setTpPort() : void
setUrlDa() : void
setVendorInfo() : void
setVerIndProgIdAe() : void
setVerIndProgIdDa() : void
setVersionOtb() : void
setWebAdministratorPassword() : void
setWebOperatorPassword() : void
setWebRootDirectory() : void
setWebRootFile() : void
start() : long
stop() : long
terminate() : long
trace() : void
DaTransaction
Class
DaSession
Class
DaRequest
Class
OpcServer
Class
AeAddressSpaceElement
Class
AddressSpaceElement
DaAddressSpaceElement
Class
AddressSpaceElement
Creator
Class
DaAddressSpaceRoot
Class
AddressSpaceRoot
AeAddressSpaceRoot
Class
AddressSpaceRoot


Variant
Class
tagVARIANT
Methods
~Variant()
areEqual() : int
areEqualArrays() : int
changeDatatype() : int (+ 2 overloads)
clear() : void
copyTo() : int
get() : void
getBSTR() : TCHAR*
getSize() : unsigned long (+ 1 overload)
init() : void
initArray() : void
initFrom() : int
operator !=() : int (+ 2 overloads)
operator =() : const Variant& (+ 17 overloads)
operator ==() : int (+ 2 overloads)
operator LPCVARIANT() : int
operator LPVARIANT() : int
SafeArrayAccessData() : long
safeArrayCreate() : LPSAFEARRAY
safeArrayCreateVector() : LPSAFEARRAY
SafeArrayGetLBound() : long
SafeArrayGetUBound() : long
SafeArrayUnaccessData() : long
set() : Variant&
setBOOL() : void
setBSTR() : void
setCY() : void
setDATE() : void
setI1() : void
setI2() : void
setI4() : void
setI8() : void
setINT() : void
setR4() : void
setR8() : void
setUI1() : void
setUI2() : void
setUI4() : void
setUI8() : void
setUINT() : void
supportDatatype() : int
toString() : tstring
Variant() (+ 18 overloads)
variantClear() : long
variantCopy() : long
public
tagVARIANT
Struct
Fields
boolVal : VARIANT_BOOL
bstrVal : BSTR
bVal : BYTE
byref : PVOID
cVal : CHAR
cyVal : CY
date : DATE
dblVal : DOUBLE
decVal : DECIMAL
fltVal : FLOAT
intVal : INT
iVal : SHORT
llVal : LONGLONG
lVal : LONG
parray : SAFEARRAY*
pboolVal : VARIANT_BOOL*
pbstrVal : BSTR*
pbVal : BYTE*
pcVal : CHAR*
pcyVal : CY*
pdate : DATE*
pdblVal : DOUBLE*
pdecVal : DECIMAL*
pdispVal : IDispatch*
pfltVal : FLOAT*
pintVal : INT*
piVal : SHORT*
pllVal : LONGLONG*
plVal : LONG*
pparray : SAFEARRAY**
ppdispVal : IDispatch**
ppunkVal : IUnknown**
pRecInfo : IRecordInfo*
pscode : SCODE*
puintVal : UINT*
puiVal : USHORT*
pullVal : ULONGLONG*
pulVal : ULONG*
punkVal : IUnknown*
pvarVal : VARIANT*
pvRecord : PVOID
scode : SCODE
uintVal : UINT
uiVal : USHORT
ullVal : ULONGLONG
ulVal : ULONG
vt : VARTYPE
wReserved1 : WORD
wReserved2 : WORD
wReserved3 : WORD
ValueData
Class
Fields
m_data : Variant
Methods
~ValueData()
getData() : Variant&
toString() : tstring
ValueData() (+ 2 overloads)
public
ValueQT
Class
ValueData
Fields
m_quality : EnumQuality
m_timeStamp : DateTime
Methods
~ValueQT()
getQuality() : EnumQuality
getTimeStamp() : DateTime
isQualitySet() : int
isTimeStampSet() : int
setData() : void
toString() : tstring
ValueQT() (+ 2 overloads)
DateTime
Class
Fields
m_filetime : FILETIME
Methods
addMicroseconds() : void
clear() : void
DateTime() (+ 2 overloads)
day() : unsigned short
dayGMT() : unsigned short
get() : void (+ 2 overloads)
getMicroseconds() : ULONGLONG
hour() : unsigned short
hourGMT() : unsigned short
init() : void
isSet() : int
milliSecond() : unsigned long
milliSecondGMT() : unsigned long
minute() : unsigned short
minuteGMT() : unsigned short
month() : unsigned short
monthGMT() : unsigned short
now() : void
operator =() : const DateTime& (+ 1 overload)
second() : unsigned short
secondGMT() : unsigned short
set() : void
setMicroseconds() : void
toString() : tstring
toStringGMT() : tstring
year() : unsigned long
yearGMT() : unsigned long

5. *Descrição dos arquivos do Servidor OPC SISAL V 1.00*

Os arquivos da tabela abaixo, constituem o código do projeto *ServidorOPCSISAL.sln* desenvolvido no *Visual Studio C++ 2005*.

Tabela 5.1 – Arquivos do projeto ServidorOPCSISAL.sln

Arquivos	Descrição
OpcServer.cpp	Arquivo principal do projeto. Responsável pela inicialização e finalização, criação do namespace e atualização dos seus itens.
OpcServer.h	Definição das funções do arquivo principal do projeto.
OutProc.cpp	Arquivo responsável pela inicialização da janela principal do aplicativo e tratamento da thread de simulação.
stdafx.cpp	Arquivo de compilação de projetos no Visual C++.
stdafx.h	Definição de cabeçalhos de compilação de projetos no Visual C++.
Unt_PciInterface.cpp	Arquivo que trata da interface entre o servidor OPC e o servidor SISAL.
Unt_PciInterface.h	Definição dos arquivos que tratam da interface entre o servidor OPC e o servidor SISAL.
Unt_ClassePoco.h	Definição das variáveis e do objeto Poco.
Resource.h	Arquivo contendo recursos para compilação do projeto.
MyCreator.h	Descrição das funções responsáveis pela criação do objetos do projeto.
MyDaAddressSpaceRoot.h	Descrição das funções responsáveis pela criação do nó raiz do namespace da especificação DA.
MyDaAddressSpaceElement.h	Descrição das funções responsáveis pela criação dos elementos do namespace da especificação DA.
MyAeAddressSpaceElement.h	Descrição das funções responsáveis pela criação dos elementos do namespace da especificação AE do projeto.
MyRequest.h	Descrição das funções que tratam das requisições de origem do cliente OPC.
MyTransaction.h	Descrição das funções que tratam das transações de leitura ou escrita entre o cliente OPC e o servidor OPC.

A tabela abaixo contém os principais arquivos utilizados pela biblioteca do *toolbox* para compilação do programa. Alguns arquivos que constituem a implementação da especificação de alarmes e eventos não estão inclusos nesta tabela. Mais detalhes sobre estes arquivos estão na documentação do *Softing OPC toolbox*.

Tabela 5.2 – Arquivos da biblioteca do toolbox utilizados no projeto ServidorOPCSISAL

Arquivos	Descrição
ServerAddressSpaceElement.cpp	Contém funções que implementam os componentes básicos do namespace das especificações DA e AE.
ServerAddressSpaceElement.h	Definição das funções que implementam os componentes básicos do namespace das especificações DA e AE.
ServerApplication.cpp	Contém funções que tratam das principais funcionalidades do servidor OPC.
ServerApplication.h	Definição das funções que tratam das principais funcionalidades do servidor OPC.
ServerCommon.h	Responsável pela declaração da instância da aplicação.
ServerCreator.cpp	Arquivo que trata da interface entre o servidor OPC e o servidor SISAL.
ServerCreator.h	Responsável por criar as instâncias dos objetos que compõem os componentes do servido OPC.
ServerEnums.h	Definição das enumerações utilizadas pelo toolbox.
ServerDaAddressSpaceElement.cpp	Contém funções que implementam os componentes básicos do namespace da especificação DA.
ServerDaAddressSpaceElement.h	Descrição das funções que implementam os componentes básicos do namespace da especificação DA.
ServerDaProperty.cpp	Trata funções referentes à criação e configuração das propriedades dos elementos de um namespace.
ServerDaProperty.h	Descrição das funções referentes à criação e configuração das propriedades dos elementos de um namespace.
ServerDaRequest.cpp	Contém funções que tratam das requisições de origem do cliente OPC.
ServerDaRequest.h	Descrição das funções que tratam das requisições de origem do cliente OPC.
ServerDaSession.cpp	Responsável por tratar as sessões de comunicação entre o servidor OPC e o cliente OPC.
ServerDaSession.h	Descrição das funções referentes às sessões de comunicação entre o servidor OPC e o cliente OPC.
ServerDaTransaction.cpp	Responsável por tratar as funções das transações de leitura ou escrita entre o cliente OPC e o servidor OPC.

ServerDaTransaction.h	Descrição das funções das transações de leitura ou escrita entre o cliente OPC e o servidor OPC.
ServerAeAddressSpaceElement.cpp	Contém funções que implementam os componentes básicos do namespace da especificação AE.
ServerAeAddressSpaceElement.h	Descrição das funções que implementam os componentes básicos do namespace da especificação AE.
Enums.h	Contém as enumerações utilizadas pelo toolbox.
Mutex.cpp	Trata funções referentes aos mutexes.
Mutex.h	Definição das funções referentes aos mutexes.
OSCompat.h	Trata das inclusões (include) e definições (define) específicas do sistema operacional.
SOModule.h	Definições referentes ao módulo da versão do toolbox. Incui as DLL de funcionamento do toolbox.
SOVersion.h	Definições referentes à versão do toolbox.
Trace.cpp	Trata de funções e propriedades que ajudam a traçar (rastreando) a execução do código.
Trace.h	Definição das funções e propriedades que ajudam a traçar (rastreando) a execução do código.
ValueQT.cpp	Trata funções referentes ao tipo ValueQT.
ValueQT.h	Definição das funções referentes ao tipo ValueQT.
Pci.dll	Biblioteca de ligação dinâmica responsável por tratar da interface entre o servidor OPC e o servidor SISAL

Os arquivos da tabela abaixo constituem o código que compõe a DLL (“pci.dll”) referente à comunicação PCI entre o servidor SISAL e o servidor OPC.

Tabela 5.3 – Arquivos da DLL pci.dll

Arquivos	Descrição
Unt_Status.cpp	Arquivo com o corpo das funções necessárias a verificação de Status de Funcionamento do protocolo PCI.
Unt_Status.h	Arquivo com as definições da Classe STATUS.
Unt_ClassePoco.cpp	Arquivo com o corpo das funções de inicialização de parâmetros de controle e métodos da classe poço.
Unt_ClassePoco.h	Definição da Classe Poço.
Unt_ClasseSistema.cpp	Principal classe de interface do Cliente SISAL. Neste arquivo estão disponíveis muitos métodos de aquisição e tratamento de valores ligados aos poços.

Unt_ClasseSistema.h	Arquivo com definições da classe sistema.
Unt_PCI.cpp	Arquivo com o corpo das funções de controle do Protocolo PCI, referentes à Classe PCI.
Unt_PCI.h	Arquivo com as definições das Classes de Controle do Protocolo PCI.
Unt_FuncoesPCI.cpp	Define a implementação do protocolo PCI e os métodos da classe FuncoesPCI de comunicação.
Unt_FuncoesPCI.h	Descreve a classe FuncoesPCI do protocolo PCI, seus métodos e atributos.
Unt_Principal.cpp	Arquivo principal de execução do cliente contendo todos os componentes necessários à sua execução.
Unt_Principal.h	Arquivo de cabeçalho do principal arquivo de execução do cliente.
Unt_StringList.cpp	Implementa a classe StringList, um dos tipos de dados utilizado como parâmetro de comunicação nos protocolos PCI.
Unt_StringList.h	Define a classe StringList, seus métodos e atributos.
Unt_ProcessoPrincipal.cpp	Arquivo com o corpo do método de execução da Thread Principal.
Unt_ProcessoPrincipal.h	Definição das funções que compõem a Thread Principal.
Unt_ProcessoReceptor.cpp	Arquivo com o corpo do método de execução da Thread de Recepção.
Unt_ProcessoReceptor.h	Arquivo com as definições necessárias a Thread de Recepção.
Unt_ProcessoVerificador.cpp	Arquivo com o corpo do método de execução da Thread de Verificação de Timeout's.
Unt_ProcessoVerificador.h	Arquivo com as definições necessárias a Thread de Verificação de Timeout's.
Unt_Buffercircular.h	Implementa um buffer circular de uso geral através da classe Buffer.
md5.cpp	Contém a implementação da criptogradia md5.
md5.h	Contém a definição das funções da criptogradia md5.

6. Sobre o Aplicativo

O Servidor OPC para o SISAL é resultado do trabalho de conclusão de curso referente ao curso de Engenharia de Computação da Universidade Federal do Rio Grande do Norte através do projeto AUTOPOC localizado no Laboratório de Automação em Petróleo (LAUT) e do trabalho de conclusão referente ao programa de especialização em Engenharia de Processo de Plantas de Petróleo e Gás Natural vinculado ao Programa de Recursos Humanos da Agência Nacional de Petróleo, Gás Natural e Biocombustíveis nº 14 (PRH ANP – 14).

O projeto AUTOPOC (Automação de Poços) tem a coordenação do Prof. Adelardo Adelino Dantas de Medeiros (adelardo@dca.ufrn.br) na UFRN e o programa PRH ANP – 14 tem a coordenação do Prof. Osvaldo Chiavone Filho (osvaldo@eq.ufrn.br) na UFRN.

Equipe de desenvolvimento:

- Alan Diego Dantas Protasio – alanprot@gmail.com
- João Teixeira de Carvalho Neto – joaoteixeiracanon828@gmail.com

---

anp
Agência Nacional
do Petróleo,
Gás Natural e Biocombustíveis

PRH ANP-14
PROGRAMA DE RECURSOS HUMANOS ANP-14

Anexo

Anexo C – Documentação de Uso do Servidor OPC para o SISAL

SERVIDOR OPC
SISAL

Documentação de Uso
Servidor OPC SISAL V 1.00

Julho 2010
Documentação de Uso Servidor OPC SISAL V100 – Revisão 0

Apresentação

Esta documentação é destinado ao usuário final do aplicativo Servidor OPC SISAL v1.0 e ao administrador do sistema que será monitorado. Este manual também tem como objetivo fornecer um guia, que permita aos usuários encontrar as respostas para as dúvidas mais comuns referentes à instalação do programa, procedimento de execução do programa e os requisitos necessários para sua instalação. Este manual deve ser consultado sempre que surgirem dúvidas quanto ao procedimento de execução de uma função do produto, ou quanto ao procedimento de sua instalação.

Qualquer dúvida relativa às informações contidas neste documento pode ser esclarecida diretamente através dos telefones:

- João Teixeira de Carvalho Neto - (84) 9138-1668
- João Teixeira de Carvalho Neto - (84) 8736-1668

Desenvolvimento

Universidade Federal do Rio Grande do Norte – UFRN
Departamento de Computação e Automação – DCA
Email: adelardo@dca.ufrn.br
Email: joaoteixeiracanon828@gmail.com

Julho de 2010

Documentação de Uso Servidor OPC SISAL V100

Sumário

1. Introdução

O Servidor OPC SISAL tem como objetivo apresentar uma interface em segundo plano para disponibilizar os dados dos poços disponíveis no servidor SISAL para serem visualizados em um cliente OPC. A versão atual do Servidor OPC SISAL (v1.00) contém as seguintes funcionalidades, vantagens e limitações:

- Leitura de variáveis dos poços, contidos no servidor SISAL, em tempo real.
- Comunicação entre cliente OPC e servidor OPC pode ser feita via Ethernet, possibilitando a instalação do servidor OPC em campo e do cliente OPC em escritório.
- Leitura de 4 variáveis dos poços: *cod_poço, Estação, End_Controlador* e *Grau API*.
- Tempo de execução limitado a 90 minutos.

Para monitoramento das variáveis dos poços é necessário um cliente OPC instalado na máquina. Cada cliente OPC, tem sua interface diferente, porém têm praticamente as mesmas funcionalidades. Para demonstrar as funcionalidades do aplicativo Servidor OPC SISAL V1.00 neste manual, será utilizado o cliente *Softing OPC Toolbox Demo Client*.

2. *Requisitos do Sistema*

Os requisitos do sistema recomendados para a instalação e uso da aplicação são os seguintes:

- Sistema operacional:
  - Microsoft ® Windows ® XP.
- *Microsoft Visual Studio 2005 (Express Edition)*.
- 26 MB livres no disco rígido.
- *Postgresql* compatível com banco de dados do SISAL.
- Servidor SISAL.
- Mestre de Banco (Mestre BD SISAL).
- Cliente OPC.

O instalador do aplicativo tem aproximadamente 4 MB.

3. *Instalação do Aplicativo*

Os passos a seguir devem ser realizados após a instalação da plataforma *Microsoft Visual Studio 2005* (*Express Edition*).

- Passo 1: Clique no ícone do instalador do aplicativo Servidor OPC SISAL V1.00 para iniciar a instalação do aplicativo. Selecione o idioma para instalação.

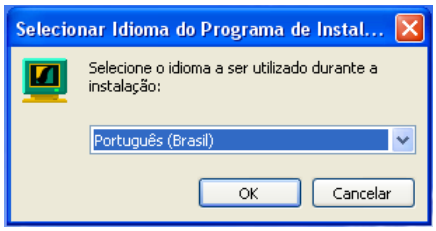


Figura 3.1 - Passo 1: Escolha do idioma para instalação do aplicativo

- Passo 2: Clique em "avançar" e leia as informações referentes à instalação. Este arquivo poderá ser encontrado no diretório de instalação do aplicativo. Este arquivo contém a URL a ser utilizada para cadastrar o servidor OPC no cliente OPC.

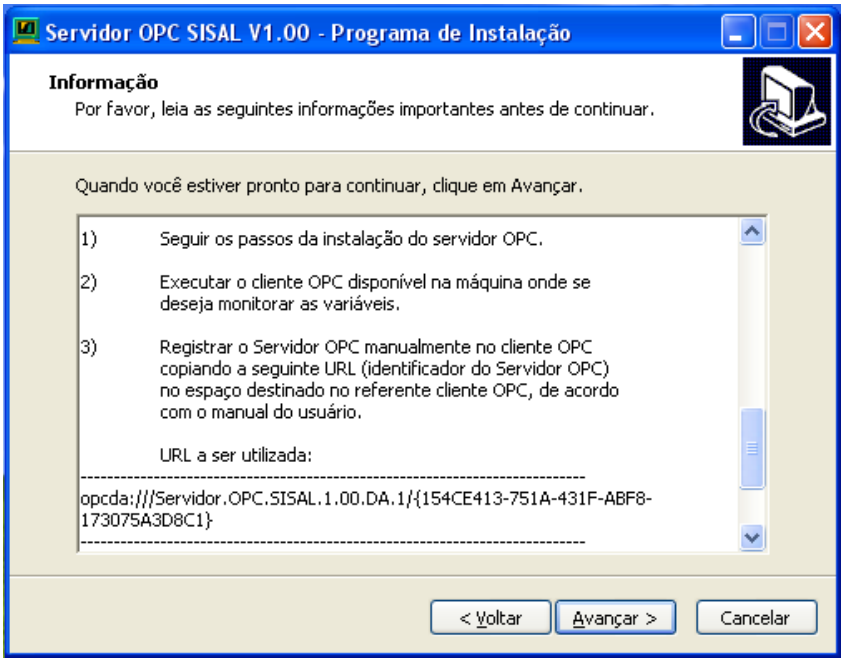


Figura 3.2 - Passo 2: Informações para instalação

- Passo 3: Escolha o diretório para instalação do aplicativo.

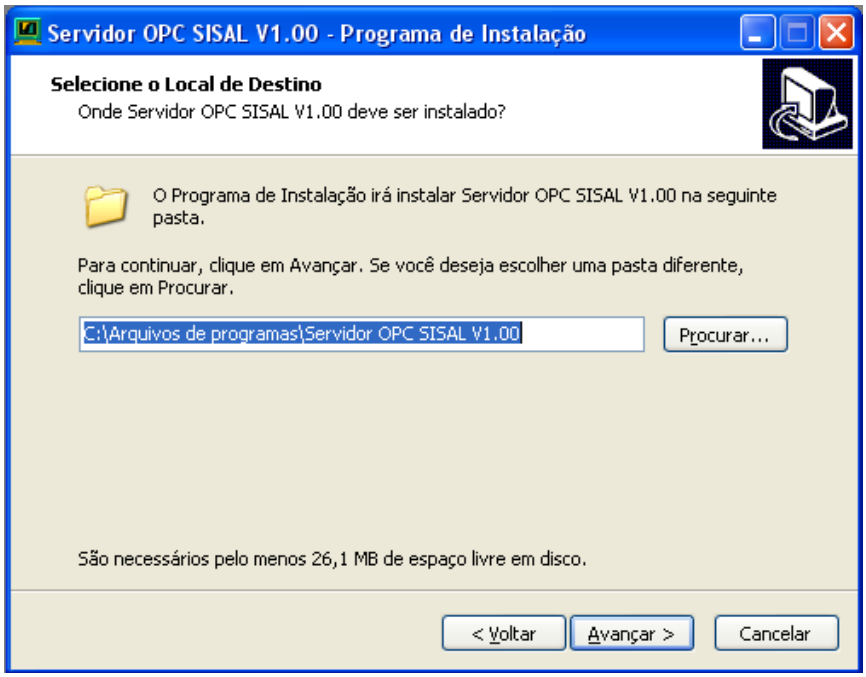


Figura 3.3 - Passo 3: Escolha do diretório para instalação do aplicativo

- Passo 4: Escolha o diretório para localização de atalhos.

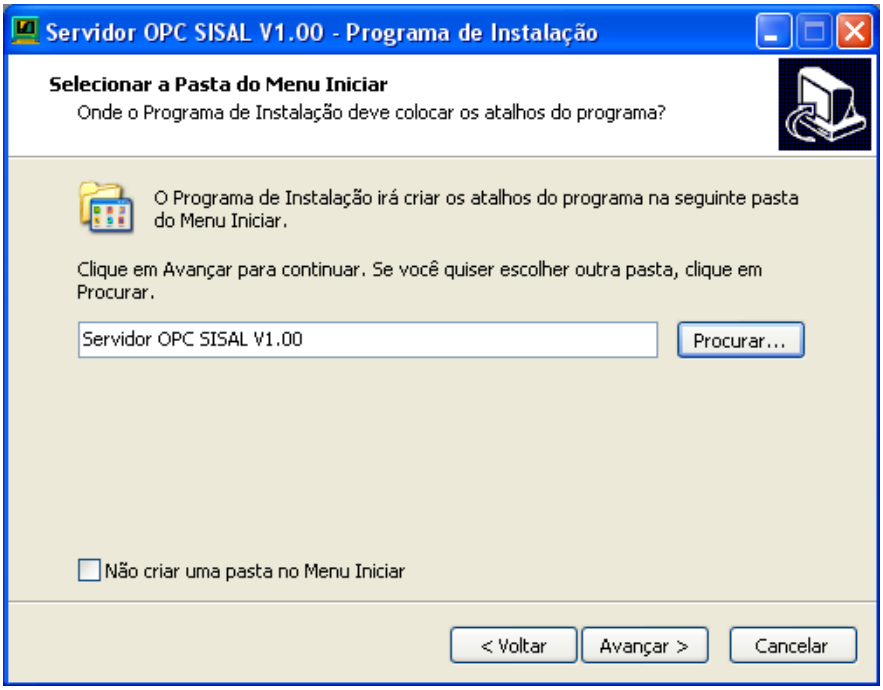


Figura 3.4 - Passo 4: Escolha de diretório para localização dos atalhos

- Passo 5: Selecione os ícones adicionais se desejar.

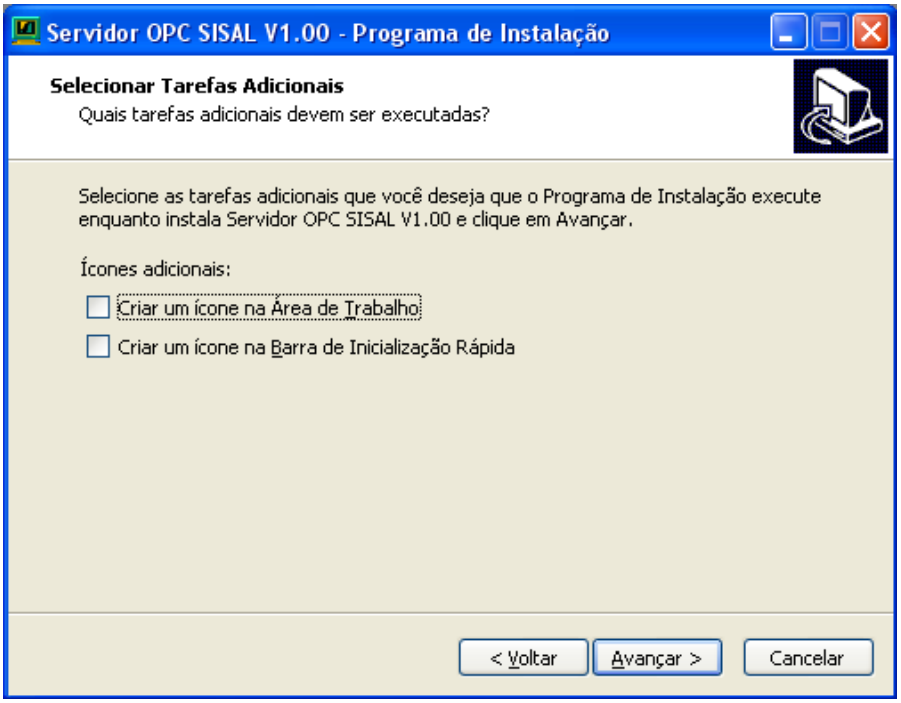


Figura 3.5 - Passo 5: Seleção de ícones adicionais

- Passo 6: Clique em “avançar” e na próxima janela em “instalar” e aguarde a instalação do programa.

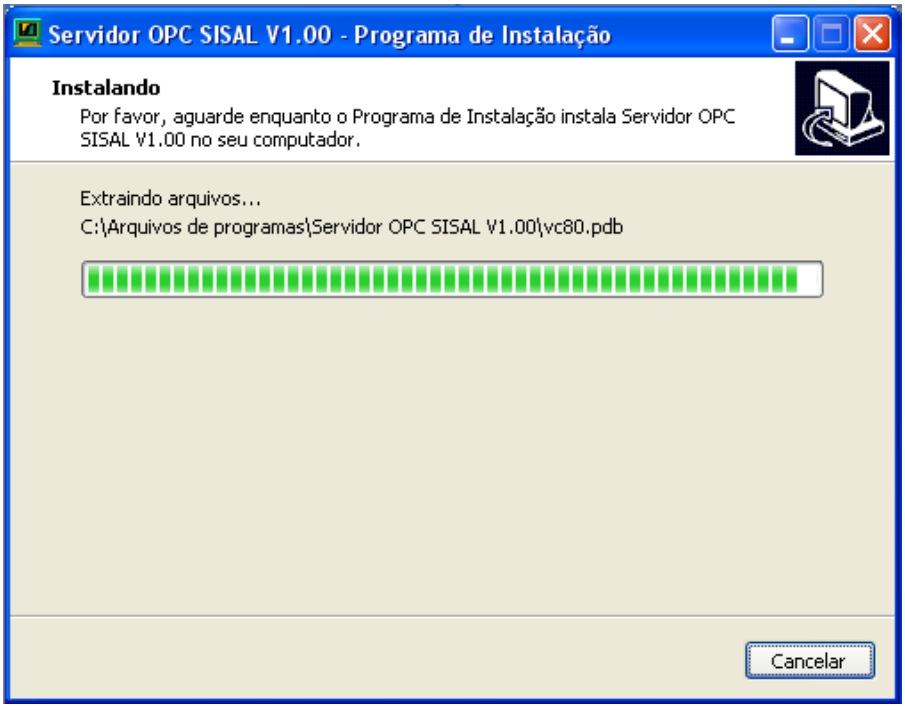


Figura 3.6 - Passo 6: Barra de instalação do aplicativo

- Passo 7: Se desejar executar o aplicativo, é necessário seguir os passos seguidos no de configuração e execução do aplicativo descritos na próxima sessão, Configuração e execução.

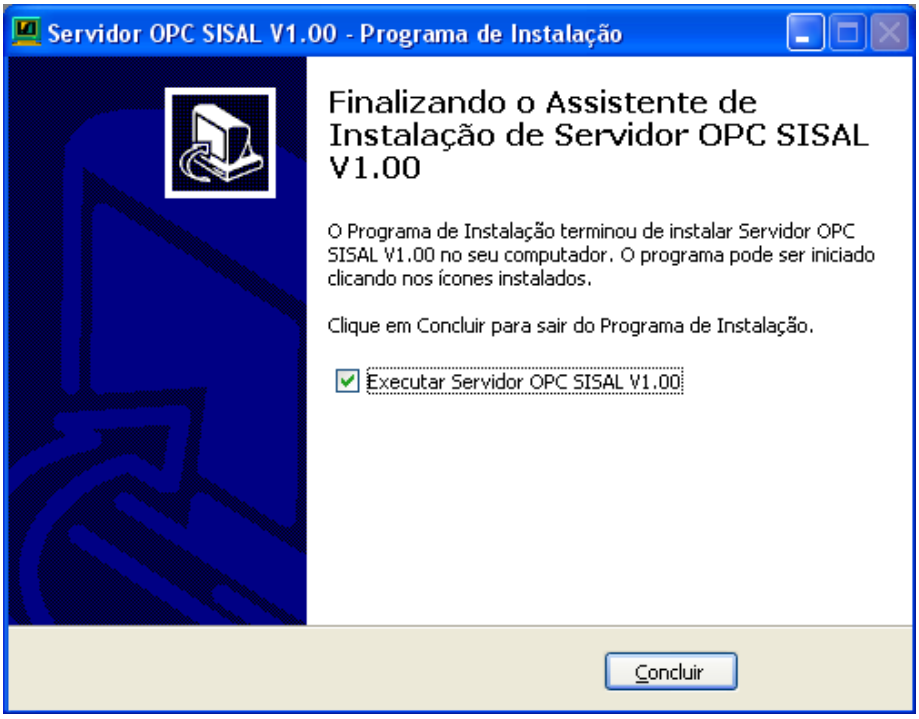


Figura 3.7 - Passo 7: Finalização do assitente de instalação do aplicativo

4. Iniciando o Aplicativo

Os passos a seguir devem ser realizados após a instalação do aplicativo Servidor OPC SISAL V 1.00.

- Passo1: O primeiro passo a ser realizado após a instalação, é executar o arquivo de texto LEIA-ME que está presente no diretório de instalação do aplicativo Servidor. Veja a figura a seguir.

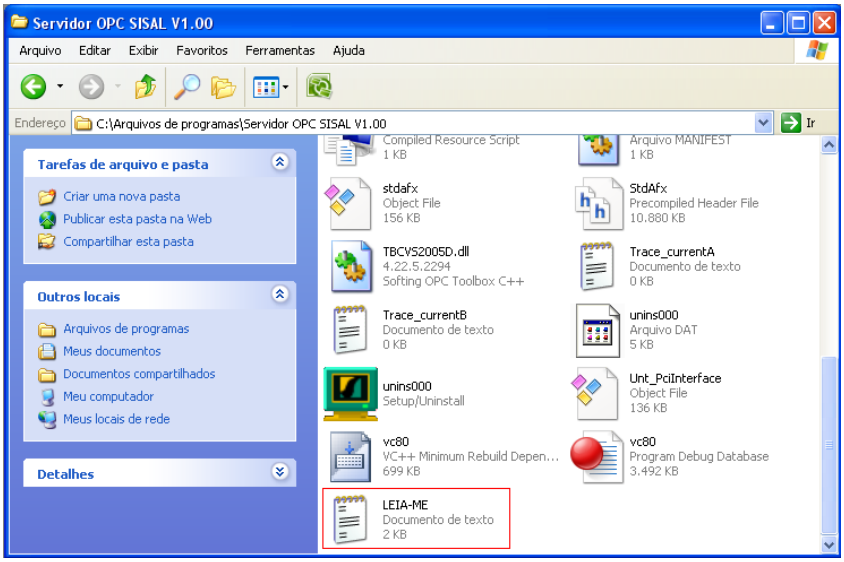


Figura 4.1 - Localização do arquivo de configuração do aplicativo

Este arquivo contém o identificador do servidor que deverá ser registrado colocado manualmente no cliente OPC, como é ilustrado na figura abaixo.

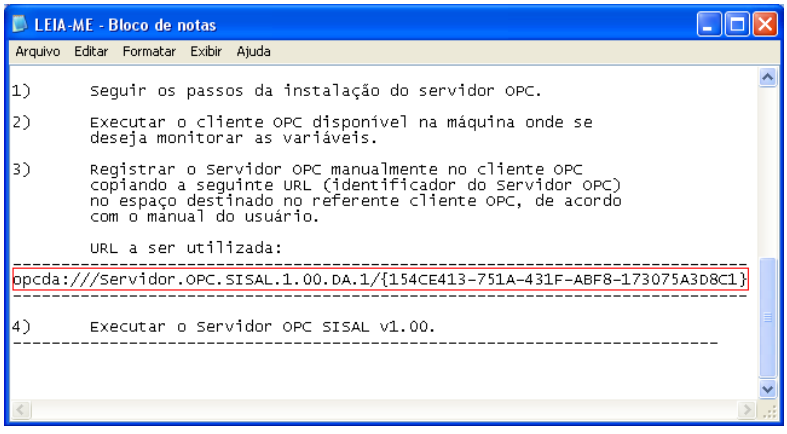


Figura 4.2 - Identificador do servidor OPC

- Passo 2: Digite a URL do servidor OPC no cliente OPC escolhido, conforme a figura a seguir. Observe que o servidor OPC ainda está desativado. Isto é percebido de acordo com o ícone que estará com as cores verde e vermelho ao lado do nome do servidor OPC.

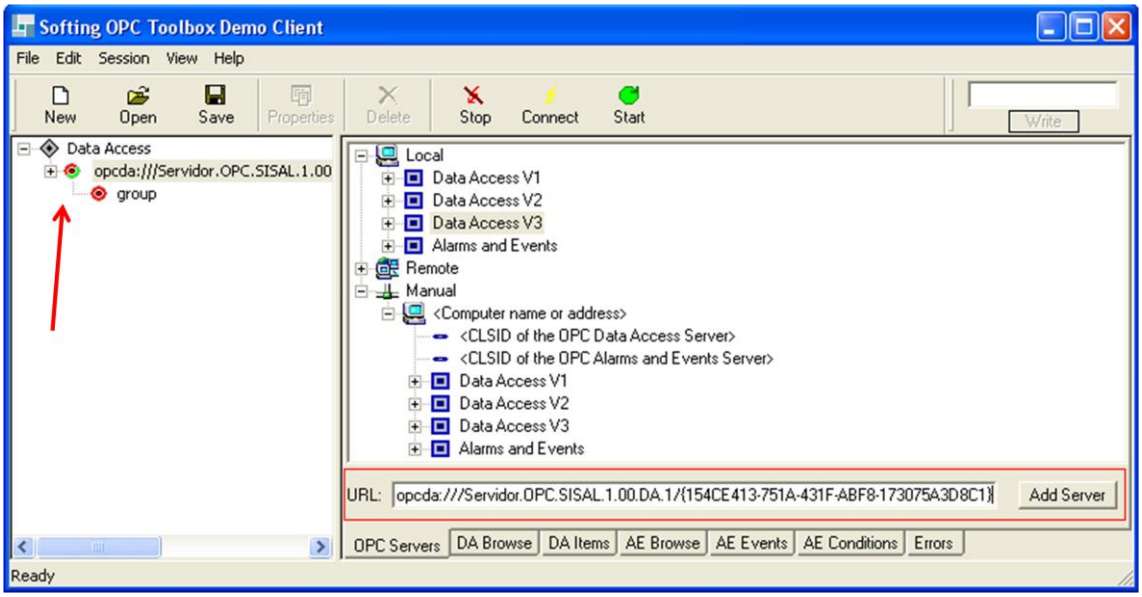


Figura 4.3 - Configuração do servidor OPC no cliente OPC

- Passo 3: A seguir é necessário executar o aplicativo Servidor OPC SISAL 1.00. Fazendo isso, será aberta a seguinte janela.

Figura 4.4 - Aplicativo Servidor OPC SISAL 1.00 em execução

O ícone, ao lado do nome do servidor, que antes estava vermelho e verde, ficará totalmente verde (veja a figura 30), indicando que o servidor OPC está conectado ao respectivo cliente. Após isso, pode ser realizada a navegação na lista de poços e suas variáveis através do cliente OPC e escolher as variáveis a serem monitoradas.

Figura 4.5 - Servidor OPC conectado ao cliente OPC

5. *Modo de Funcionamento*

Após iniciado o servidor OPC e conectado o mesmo, ao cliente OPC, pode-se fazer a navegação na lista de poços e escolher as variáveis a serem monitoradas. Nesta versão do servidor OPC existem 5 variáveis que servem para ser monitoradas ou simplesmente para descrever alguma característica referente ao poço. Esta lista pode ser vista na aba "*DA Browser*". A lista montada no cliente OPC tem a seguinte estrutura: cada nó é constituído do nome de todos os poços que estão disponíveis pelo servidor SISAL (por exemplo: *ARG0216U, ARG0249U, FP0300U, MAG0020S*). Cada nó contém 4 informações referentes a cada poço: *cod_poco, Estacao, End_Controlador, GRAU API*. Além destas 4 variáveis pode-se ainda ter a informação do tipo de poço através do nó com o nome do poço. Para expandir a árvore, basta clicar no símbolo "+". A figura a seguir mostra a forma como é mostrada a lista de poços e variáveis disponibilizadas pelo servidor OPC.

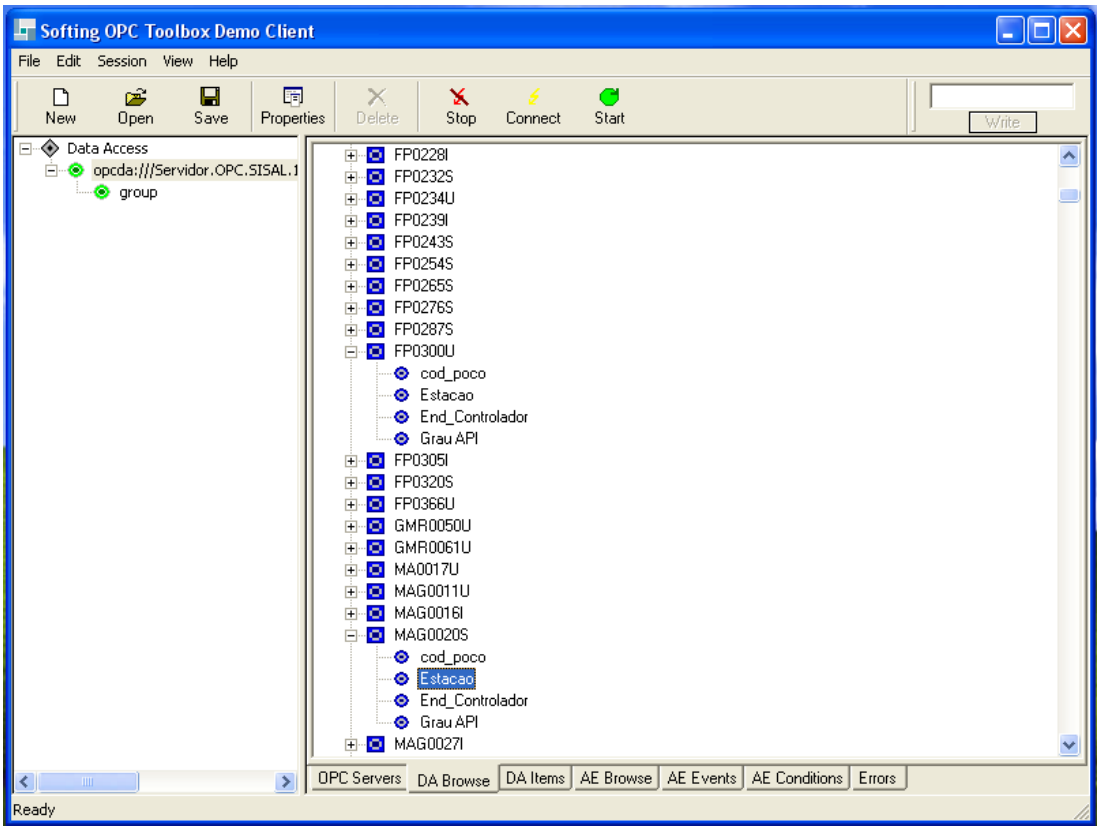


Figura 5.1 - Modo como a lista de poços e suas variáveis são mostradas no cliente OPC

Antes de monitorar as variáveis é necessária a criação de um grupo. Para realizar este passo neste cliente, é preciso criar um grupo clicando com o botão direito do mouse no nome do servidor e escolhendo a opção "*Add Group*". Para se ter a informação referente a cada nó ou a cada item, basta clicar duas vezes no respectivo item ou nó. Na aba "*DA Items*", são mostradas as variáveis que foram selecionadas. A figura a seguir mostra as informações (Valor ou *Value*, Qualidade ou *Quality* e *TimeStamp*) referentes às variáveis do poço de nome *SCR0062I* incluído no grupo "Monitoramento 1".

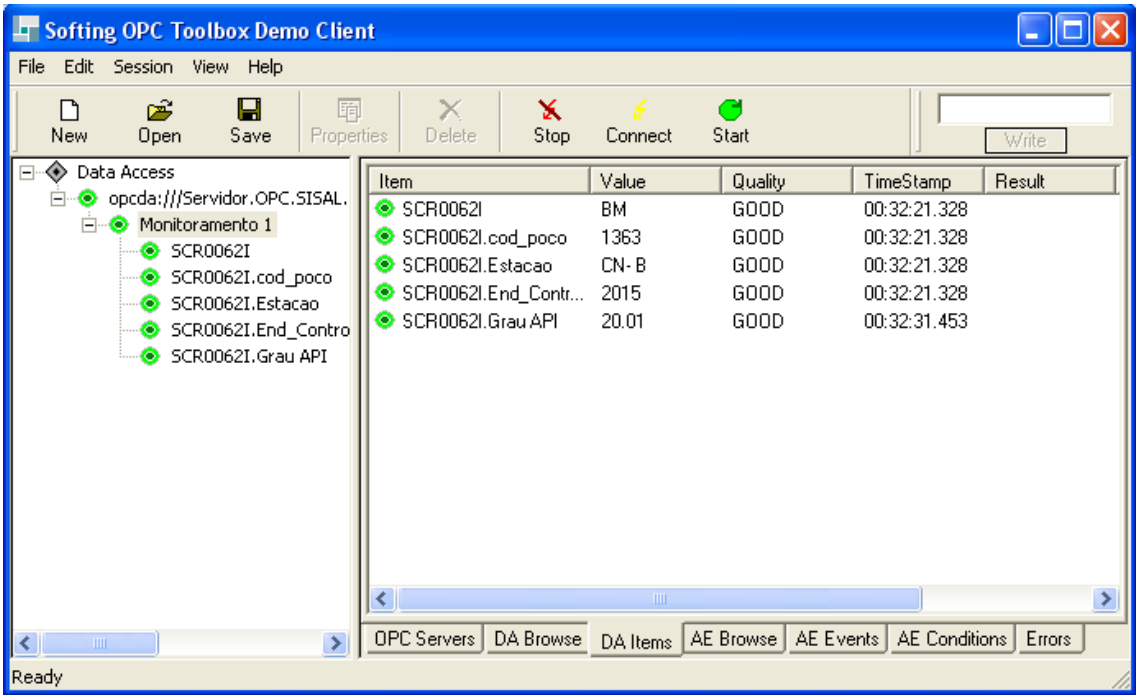


Figura 5.2 - Itens do servidor SISAL sendo monitorados por um cliente OPC

Para listar todas as variáveis para monitoramento, é necessário clicar no nome do servidor localizado na parte superior da aba "*DA Browser*" e escolher a opção "*Add Items for All Tags*". Esta opção deixará o processamento da máquina muito lento.

7. Sobre o Aplicativo

O Servidor OPC SISAL V1.00 é resultado do trabalho de conclusão de curso referente ao curso de Engenharia de Computação da Universidade Federal do Rio Grande do Norte através do projeto AUTOPOC localizado no Laboratório de Automação em Petróleo (LAUT) e do trabalho de conclusão referente ao programa de especialização em Engenharia de Processo de Plantas de Petróleo e Gás Natural vinculado ao Programa de Recursos Humanos da Agência Nacional de Petróleo, Gás Natural e Biocombustíveis nº 14 (PRH ANP – 14).

O projeto AUTOPOC (Automação de Poços) tem a coordenação do Prof. Adelardo Adelino Dantas de Medeiros (adelardo@dca.ufrn.br) na UFRN e o programa PRH ANP – 14 tem a coordenação do Prof. Osvaldo Chiavone Filho (osvaldo@eq.ufrn.br) na UFRN.

Equipe de desenvolvimento:

- Alan Diego Dantas Protasio – alanprot@gmail.com
- João Teixeira de Carvalho Neto – joaoteixeiracanon828@gmail.com

---

anp
Agência Nacional
do Petróleo,
Gás Natural e Biocombustíveis