UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE ENGENHARIA ELÉTRICA
COORDENAÇÃO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Relatório do Projeto de Fim de Curso

Comunicação de dispositivos móveis em ambientes de automação industrial

Aluno:
Ademar Virgolino da Silva Netto
ademaree@gmail.com

Orientadora:
Prof.ª Maria de Fátima Queiroz Vieira
fatima@dee.ufcg.edu.br

Campina Grande, Março de 2008.

Biblioteca Setorial do CDSA. Fevereiro de 2021.

Sumé - PB

Sumário

Tabelas

Quadros

Figuras

Glossário

API: *Application Programming Interfaces*
CORBA: *Common Object Request Broker Architecture*
DCOM: *Distributed component object model*
DTD: *Document Type Definition*
HTML: *HyperText Markup Language*
HTTP: *HyperText Transfer Protocol*
IDL: *Interface Definition Language*
IIOP: *Internet Inter-ORB Protocol*
OLE: *Object Linking and Embedding*
OMA: *Object Management Architecture*
OMG: *Object Management Group*
OPC: *OLE for Process Control*
OPC XML-DA: *OLE for Process Control -Data Access*
ORB: *Object Request Broker*
PDA: Personal Digital Assistant
PI: *Instrução de Processamento*
RPC: *Remote Procedure Call*
SGML: *Standard Generalized Markup Language*
SOAP: *Simple Object Access Protocol*
TCP/IP: *Transmission Control Protocol/Internet Protocol*
URI: *Universal Resource Identifier*
W3C: *World Wide Web Consortium*
WSDL: *Web Service Definition Language*
XML: *eXtensible Markup Language*
XML-RPC: *eXtensible Markup Language – Remote Procedure Calling*

1. Introdução

A comunicação nos sistemas industriais automatizados é um fator importante pois a falta ou perda de informação pode prejudicar o processo, ocasionar acidentes e prejuízos para a empresa. Muitas empresas utilizam dispositivos em medições externas à sala de comando, tais como PDAs (Personal Digital Assistant) e *tablets* os quais trocam informações com o sistema central de monitoramento e controle.

Este projeto de conclusão de curso dá continuidade a um trabalho no contexto da representação de interfaces com o usuário de sistemas de supervisão industrial. Este projeto utiliza a notação XML (*Extensible Markup Language*) [1] como base para uma notação capaz de representar a semântica deste contexto e assim simplificar o projeto de telas para os diferentes dispositivos encontrados nestes ambientes. No projeto original, foi adotado como estudo de caso o setor de transporte de produtos em uma planta petroquímica da empresa Petrobrás. Inicialmente a notação XML foi utilizada na comunicação entre um PDA e um computador *desktop* para representar as telas do sistema supervisório da planta.

Neste trabalho serão estudados protocolos e arquiteturas que possibilitem a interoperabilidade, em particular: SOAP [2] , OPC [3] e CORBA [4] visando avaliar sua adequação ao projeto e, mais especificamente, com o propósito de aperfeiçoar a comunicação entre dispositivos em ambientes de automação e a execução de um mesmo aplicativo em plataformas diferentes. Segundo o W3C (*World Wide Web Consortium*) XML é uma linguagem que permite que um mesmo documento possa ser reconhecido em diferentes plataformas.

O SOAP (*Simple Object Access Protocol*) é um protocolo concebido pela IBM que apóia a troca de mensagens com base em XML. SOAP suporta diversos protocolos de transporte e métodos de codificação de dados o que motivou o seu estudo.

CORBA (*Common Object Request Broker Architecture*) [5] é uma arquitetura que possibilita a interação entre plataformas e linguagens, utilizando protocolos próprios de comunicação.

A arquitetura OPC (*OLE for Process Control*) proposta pela OPC *Foundation*, é adotada na comunicação em tempo real e através de redes locais. Serve para simplificar o

compartilhamento e a troca de dados através dos vários níveis de hierarquia de uma planta industrial. Uma evolução da especificação OPC é o OPC XML-DA [6], que utiliza XML. A troca de dados se dá em XML utilizando *schemas* para validar os arquivos de dados. A OPC *Foundation* definiu *schemas* específicos para OPC XML os quais devem ser utilizados para a composição e validação das mensagens enviadas e recebidas. Clientes e servidores que utilizam OPC XML-DA compartilham os mesmos conceitos de XML ao utilizar um mesmo *schema* XML; resultando em alta interoperabilidade.

Este relatório está organizado da seguinte forma. Na seção 2 é apresentada uma revisão bibliográfica sobre as arquiteturas e protocolo estudados, como soluções potenciais para o problema da interoperabilidade de aplicativos nos dispositivos utilizados na automação industrial. Na seção 3, as arquiteturas e o protocolo são avaliados do ponto de vista dos propósitos deste projeto. Na seção 4 é apresentada a solução proposta para o problema e sua validação no contexto de um estudo de caso realizado no âmbito de um projeto da Transpetro – Empresa do Grupo Petrobrás. Finalmente, na seção 5, são apresentadas as conclusões e propostas de continuidade para este trabalho.

2. A notação XML

A notação XML é utilizada em diversos protocolos na comunicação via Internet. Concebida em meados de 1990, XML teve como propósito suprir a necessidade de um padrão que pudesse: ser facilmente utilizado na Internet; suportar uma ampla variedade de aplicações; ser compatível com SGML - *Standard Generalized Markup Language*; facilitar a programação de aplicativos para seu processamento; oferecesse um mínimo de atributos opcionais; gerasse documentos legíveis e razoavelmente claros para o ser humano; apoiasse a formatação de documentos de forma rápida, formal e concisa; que os documentos XML não precisassem sintetizar informações.

Adotada como formato alternativo (e muitas vezes padrão) de diversos pacotes de software para armazenamento de dados (informações), o XML apresenta todas as características desejáveis para promover a interoperabilidade entre aplicativos.

2.1. Componentes XML

Uma característica adicional dos documentos XML é a separação entre informação de conteúdo e de apresentação. Assim, dados em XML possuem uma estrutura composta de até três documentos:

(I) Conteúdo;

(II) Validador (DTD ou XML Schema);

(III) Formato/Apresentação (Instruções de processamento/Folhas de estilo);

sendo os dois primeiros de uso obrigatório e o último opcional.

2.1.1. Conteúdo

Trata-se do documento XML, propriamente dito. É o arquivo que contém as informações dos artefatos de entrada/saída da aplicação. Este arquivo está organizado na forma de um texto estruturado de marcações (*tags*) que delimitam o início e fim dos dados contidos no documento XML.

As *tags* relacionadas a um documento XML podem ser subdivididas em: (I) elementos e atributos; (II) comentários; (III) referências a entidades; (IV) instruções de processamento; (V) declarações de tipos de documentos e (VI) modo de apresentação.

Elementos e Atributos

As marcações do documento XML podem conter elementos e/ou atributos, os elementos permitem representar as informações complexas (mais de uma informação em uma marcação) sendo essas atributos ou outros elementos, enquanto que os atributos são compostos por informações simples (apenas uma informação por atributo). Um aspecto importante na definição de uma especificação XML é a decisão sobre o que deve ser elemento e o que deve ser atributo, pois esta escolha tem implicações sobre expansões da especificação criada. Na Tabela 1 são apresentadas as diferenças entre atributos e elementos.

Tabela 1 – Diferenças entre atributos e elementos

Atributos	Elementos
Não podem conter elementos e/ou atributos	Podem conter elementos e/ou atributos
Não podem ser estruturados	Podem ser estruturados

Comentários

O XML permite ao projetista inserir comentários na estrutura criada, de modo a torná-la mais clara. Por exemplo: explicitando características especificas, exemplificando elementos e atributos. No Quadro 1 apresenta um trecho de código no qual há um comentário para indicar o início de uma nova seção de especificação (em negrito). Cabe observar que comentários são delimitados na notação XML pelos caracteres "<!--" e "-->".

Quadro 1 - Comentários

```
  </xs:simpleType>
<!--Tipos de terminacao-->
  <xs:simpleType name="tipo_terminacao">
```

Referências às entidades

Semelhantemente à HTML, quando é necessária a apresentação de caracteres especiais de marcação, devem-se utilizar comandos que referenciem estes caracteres (Tabela 2).

Tabela 2 – Exemplos de Referencias a Entidades

XML Referência às entidades pré-definidas	
Referência à entidade:	Caractere
&amp;	&
&lt;	<
&gt;	>
&quot;	"
&apos;	'

Instruções de Processamento

Possibilitam a inclusão de instruções de processamento (PI), no documento XML, para as aplicações de forma a orientá-las quanto a forma de manipular os dados contidos no documento.

Uma instrução de processamento inicia com um alvo (*target*) para identificar a aplicação para a qual é direcionada a Instrução de Processamento (PI); seguida da instrução (Quadro 2).

Quadro 2 – Exemplo de Instruções de Processamento

```
<exemplo>
  <?perl lower-to-upper-case  ?>
  <?web-server add-header = "universidade" ?>
  <texto>campina grande </texto>
</exemplo>
```

No Quadro 2 indica para adicionar no cabeçalho com nome universidade o texto campina grande. A PI pode ser inserida em qualquer parte do documento XML, desde que seja inserida fora de uma marcação, porém habitualmente estas se encontram no início do documento, ou seja, antes da marcação raiz do documento.

2.1.2. Declarações de Tipo de Documento / XML Schema

As declaraões de Tipo de Documento podem êstar presentes no Prólogo ou em um documento a parte, o validador. Este documento pode ser escrito em dois formatos

distintos: *Document Type Definition* (DTD) ou XML *Schema* [7], porém com o mesmo propósito: verificar a integridade do documento XML recebido/gerado.

DTD

O formato DTD possui uma sintaxe específica para a definição dos tipos existentes no documento XML.

Declaração de Elementos

Para um documento XML ser válido seus elementos devem estar especificados no DTD correspondente. As declarações de elementos contêm o nome do elemento e uma lista de elementos ou texto que ele pode conter. Esta lista é denominada especificação de conteúdo. Quando não se deseja impor restrições a um elemento usa-se a palavra chave *ANY*. O tipo #PCDATA é usado para elementos simples, que não possuem atributos ou outros elementos (Quadro 3).

Quadro 3 - Declaração de Elementos

```
<!ELEMENT elemento ANY>
<!ELEMENT elemento-de-texto (#PCDATA)>
```

Elementos complexos podem conter textos e elementos filhos. A ordem e o número de elementos filhos podem ser especificados através de uma seqüência na qual os elementos estão entre parênteses e separados por vírgulas (Quadro 4 - a). Para que um elemento filho ocorra uma ou mais vezes utiliza-se o sinal "+" (Quadro 4 - b). Para que um elemento filho não apareça ou apareça apenas uma vez deve-se usar o sinal "?" (Quadro 4 - c). Caso o elemento não possua restrições quanto a sua presença, utiliza-se o símbolo "*" (Quadro 4 - d). Caso seja necessário declarar elementos filhos sem ordem definida deve-se separar a lista com o símbolo "|" (Quadro 4- e). Para elementos vazios, deve-se declara-los com a palavra chave *EMPTY* (Quadro 4- f).

Quadro 4 - Especificação de filhos para elemento

```
(a) <!ELEMENT pessoa (idade, nome, telefone)>
(b) <!ELEMENT pessoa (idade, nome, telefone+)>
(c) <!ELEMENT pessoa (idade, nome, telefone?)>
(d) <!ELEMENT pessoa (idade, nome, telefone*)>
(e) <!ELEMENT veiculo (carro | moto | bicicleta)>
```

```
(f)  <!ELEMENT cor EMPTY>
```

Declaração de Atributos

Assim como os elementos também os atributos devem ser declarados no DTD. Através do comando "<!ATTLIST>" tendo a estrutura exposta no Quadro 5 - a e exemplificada no item b, onde declara-se que o atributo "idade", do elemento "pessoa", é do tipo "CDATA" e de valor padrão "25".

Quadro 5 - Declaração de Atributos

```
(a)  <!ATTLIST>(nome-elemento nome-atributo tipo valor-padrão)
(b)  <!ATTLIST>(pessoa idade CDATA "25")
```

A Tabela 3 lista os dez tipos diferentes de atributos passíveis de uso pelos projetistas e, o Quadro 6 ilustra um trecho de código exemplificando um DTD.

Tabela 3 – Tipos de Atributos

Tipo	Significado
CDATA	qualquer texto que não seja marcação (*string*)
Enumerated	o valor do atributo deve ser um de uma lista de valores pré-definidos
ID	valor único não sendo possível haver repetido (identificador)
IDREF	espécie de chave estrangeira, determina que o valor do atributo seja um identificador (ID) de outro elemento do documento
IDREFS	similar ao IDREF porém neste caso o atributo pode assumir o valor de um ou mais identificadores (ID) de outro(s) elemento(s) do documento
ENTITY	relaciona o valor do atributo a uma entidade declarada no DTD
ENTITIES	similar ao ENTITY porém neste caso o atributo pode assumir o valor de uma ou mais entidade(s) declarada(s) no DTD
NMTOKEN	indica que o atributo pode assumir valores que possuam: caracteres letras, dígitos, pontos, sublinhados, hífens e dois-pontos
NMTOKENS	similar ao NMTOKEN porém neste caso o atributo pode assumir mais de um valor
NOTATION	permite associar aplicativos a documentos diferentes do formato XML

Quadro 6 - Exemplo de DTD

```
<!ELEMENT geral (tanque_cilindrico*, tanque_esferico*, bomba*)>
<!ELEMENT tanque_cilindrico (medidor_num_tanque_cilindrico*, medidor_graf_tanque_cilindrico*,
  bjetos_ilustrativos_cilindrico*)>
<!ATTLIST tanque_cilindrico
  id_tanque_cilindrico CDATA #REQUIRED
  injetor_tanque_cilindrico (true | false | 0 | 1) #REQUIRED
>
```

XML Schema

Diferentemente do DTD um documento Schema permite maior liberdade na definição dos elementos e dos tipos associados a cada atributo e é essencialmente um documento XML.

Declaração de Elementos

O XML Schema classifica os elementos em simples e complexos. Um elemento simples contém apenas texto, não possui atributos e nem elementos filhos. Os elementos complexos podem conter elementos filhos e atributos [15].

Existem três maneiras de agrupar elementos: (I) **xs:all** indica que todos os elementos do grupo devem ocorrer no máximo uma vez, não importando a ordem; (II) **xs:choice** indica que qualquer elemento do grupo pode aparecer, não importando a ordem; (III) **xs: sequence** indica que todos os elementos do grupo devem acorrer exatamente uma vez e na ordem especificada.

Os atributos **minOccurs** e **maxOccurs** de um-elemento determinam quantas vezes o elemento pode aparecer num documento XML, sendo o valor padrão para ambos igual a 1. Para elementos que podem aparecer sem limitação máxima atribui-se o valor **unbounded** ao **maxOccurs.**

No exemplo ilustrado na

Quadro 7, os elementos COMPOSER e PRODUCER são opcionais e podem aparecer infinitas vezes. A declaração <xs:sequence> indica que os elementos devem seguir a mesma seqüência e ocorrer exatamente uma vez. O elemento SongType é complexo e os demais são simples.

Quadro 7 - Exemplo de Declaração de Elementos

```
<?xml version="1.0"?>
  <xs:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xs:element name="SONG" type="SongType"/>
    <xs:complexType name="SongType">
      <xs:sequence>
        <xs:element name="TITLE" type="xsd:string"/>
        <xs:element name="COMPOSER" type="xsd:string" maxOccurs="unbounded"/>
        <xs:element name="PRODUCER" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:schema>
```

Declaração de Atributos

Diferentemente do DTD, no XML Schema há um conjunto mais amplo de tipos simples (Figura 1), além de permitir a definição de novos tipos através da derivação, e podendo ou não serem definidas restrições.

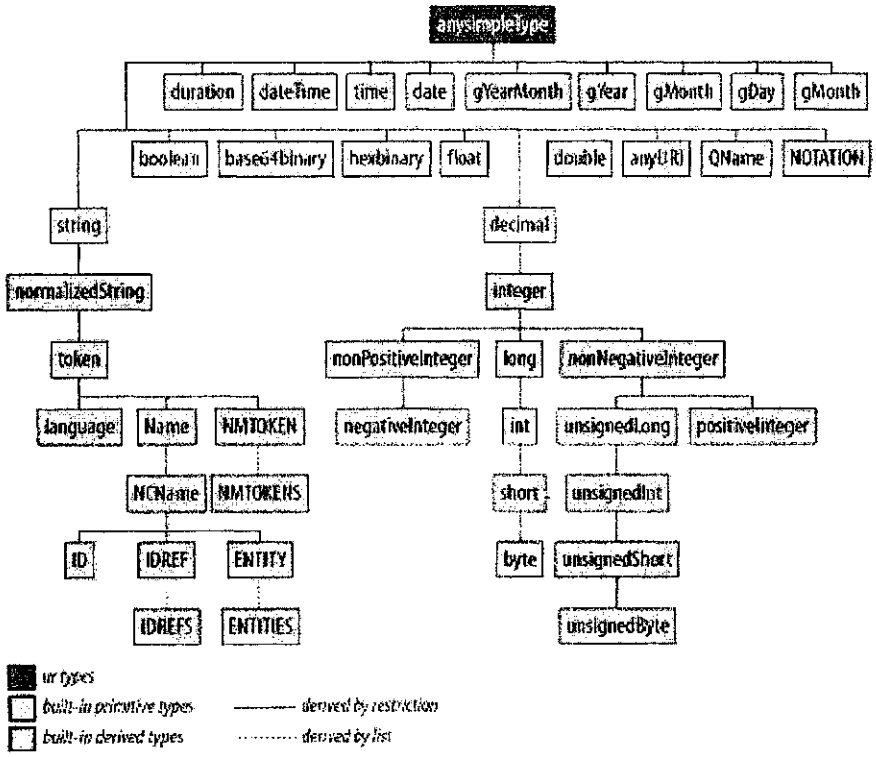

Figura 1 – Tipos pré-definidos de XML Schema

Existem duas formas de declarar atributos (Quadro 9): (I) globalmente, logo abaixo da declaração *xs:schema*, podendo o atributo ser referenciado em qualquer parte do documento Schema ou (II) localmente, não podendo o atributo ser referenciado em outras partes do documento.

Quadro 9 - Declaração de atributos

```
<xs:attribute name="age" type="xs:positiveInteger"/>
```

Comparativo DTD x XML Schema

Como exposto nas seções anteriores tanto DTD quanto XML Schema podem ser usados para gerar o documento de validação, entretanto há diferenças significativas entre eles. Na Tabela 4 – Comparativo entre DTD e XML Schema são comparadas as duas abordagens .

Tabela 4 – Comparativo entre DTD e XML Schema

DTD	XML Schema
- de simples codificação	- codificação mais complexa
- codificação diferente de XML	- codificação em XML
- não permite definição de tipos novos	- permite definição de tipos novos
- não aceita uso de namespaces	- aceita uso de namespaces
- documento tende a ficar menor	- documento tende a ficar maior

Apesar dos documentos XML Schema tenderem a ocupar mais espaço em disco, optou-se por sua utilização dadas suas vantagens, principalmente devido a maior liberdade de especificação. Em alguns protocolos tais como SOAP, o DTD está em desuso. No projeto realizado para a indústria petroquímica foi criada uma notação XML para comunicação entre dispositivos móveis, um desktop executando o software supervisório e um servidor de banco de dados.

Na seção a seguir será apresentada uma síntese do estudo realizado sobre as arquiteturas e protocolos considerados na solução do problema proposto.

3. Arquiteturas e protocolo estudados

Neste capítulo são apresentados as arquiteturas e o protocolo estudados com o propósito de verificar sua adequação enquanto solução de comunicação no contexto do estudo. Nele é feita uma descrição da arquitetura e do modo de funcionamento do SOAP, CORBA e de OPC.

3.1. SOAP – Protocolo de Acesso à Objeto Simples

SOAP – Simple Object Access Protocol é um protocolo de comunicação que surgiu no ano de 1998. Foi desenvolvido pelo grupo W3C[1] e objetiva apoiar a comunicação entre diferentes aplicações. Possui uma linguagem independente baseada no XML, portanto é simples e extensível e utiliza a Internet como meio de comunicação (HTTP – *HyperText Transfer Protocol*).

O SOAP permite a comunicação entre sistemas protegidos por *firewalls*, sem precisar abrir portas adicionais o que poderia comprometer a segurança do sistema. SOAP utiliza (na maioria dos servidores) a porta 80. Possui uma descrição de cada elemento na mensagem facilitando o entendimento e a proteção contra erros.

O SOAP pode ser implementado por: um servidor RPC (*Remote Protocol Call*) [13], objeto COM (*Component Object Model*), Java *servlet*, Perl *script* – e pode executar nos sistemas operacionais Windows e Linux.

3.1.1. Estrutura do protocolo SOAP

De acordo com o W3C, a estrutura da mensagem SOAP é definida em um documento XML que contém um envelope com a seguinte estrutura: cabeçalho, corpo e falhas. No Quadro 10 é apresentada a estrutura do SOAP [14].

Quadro 10 - Estrutura do SOAP.

```
<SOAP-ENV:envelope>
<!– Elemento raiz do SOAP e define que essa é uma mensagem SOAP-->
<SOAP-ENV:header>
<!–Especifica informações especificas como autenticação (opcional)-->
</SOAP-ENV:header>
<SOAP-ENV:body>
<!–O elemento BODY contém o corpo da mensagem-->
<SOAP-ENV:fault>
<!–O elemento FAULT contém os erros que podem ocorrer-->
</SOAP-ENV:fault>
</SOAP-ENV:body>
</SOAP-ENV:envelope>
```

O envelope é obrigatório e é responsável por definir o conteúdo da mensagem. O *header* (cabeçalho) é opcional e contém os dados do cabeçalho (Quadro 12).

[1] O W3C (*World Wide Web Consortium*) cria padrões para a *Web*, com o objetivo de aumentar ao máximo o potencial da *Web* a partir de especificações, diretrizes, software e ferramentas.

Quadro 12 - Exemplo de cabeçalho

```
<?xml version='1.0' ?>
<env:Envelope xmlns: env=":://www.w3.org/2003/05/soap-envelope">
<env:Header>
  <a:autenticar xmlns:a="http://www.lihm.ufcg/teste/autenticar">
    <a:usuario>Ademar</a:usuario>
    <a:senha>lihm</a:senha>
  </a:autenticar>
</env:Header>
...
```

O *body* (corpo) é obrigatório e contém a codificação atual de uma chamada a um método com todos os argumentos de entrada ou uma resposta codificada que contém o resultado da chamada de um método. O Quadro 13 mostra o corpo de um arquivo SOAP.

Quadro 13 - Exemplo de corpo

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 ...
  <env:Body>
    <p:nodo xmlns:p="http://www.lihm.ufcg/teste/tanque_cilindrico ">
      <p:objeto>tanque_cilindrico</p:objeto>
      <p:id_tanque_cilindrico >A-001</p: id_tanque_cilindrico >
      <p:id_medidor>001</p:id_medidor>
      <p:valor_medidor>100</p:valor_medidor>
      <p:unidade>m3</p:unidade>
    </p:nodo>
  </env:Body>
</env:Envelope>
```

As mensagens do cabeçalho possuem atributos, a exemplo de: *encodingStyle* (*xmlns*) que especifica como as informações devem ser codificadas, *actor* especifica qual receptor que deve processar o elemento do cabeçalho e *mustUnderstand* que especifica se uma entrada de cabeçalho é obrigatória ou opcional (booleano). O Quadro 14 apresenta um exemplo completo de uma mensagem SOAP.

Quadro 14 - Exemplo completo

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
 <env:Header>
   <a:autenticar xmlns:t="http://www.lihm.ufcg/teste/autenticar"
                 env:encodingStyle="http://www.lihm.ufcg/encoding"
                 env:mustUnderstand="true/>
     <a:usuario>Ademar</a:usuario>
     <a:senha>lihm</a:senha>
   </a:autenticar>
 </env:Header>
 <env:Body>
  <p:nodo xmlns:p="http://www.lihm.ufcg/teste/tanque_cilindrico ">
      <p:objeto>tanque</p:objeto>
      <p:id_tanque_cilindrico >A-001</p: id_tanque_cilindrico >
      <p:id_medidor>001</p:id_medidor>
```

```
      <p:valor_medidor>100</p:valor_medidor>
      <p:unidade>m3</p:unidade>
    </p:nodo>
  </env:Body>
</env:Envelope>
```

O SOAP também possui comandos pré-definidos para controle de falhas tais como *fault* (falhas) que contém as informações dos erros ocorridos no envio da mensagem (apenas mensagens de resposta do servidor).

3.2. CORBA – Arquitetura intermediária de requisição à objeto comum

O CORBA – Common Object Request Broker foi criado pela OMG (*Object Management Group*) [12] e consititui a especificação de um *middleware*[1]. Seu propósito é estabelecer e simplificar a troca de dados entre sistemas distribuídos heterogêneos. Resumidamente, CORBA propicia a interoperabilidade local ou remota entre aplicações, independentemente da linguagem de programação em que foram desenvolvidas e da plataforma onde serão executadas.

CORBA é uma arquitetura orientada a objeto, definida apenas através da interface das classes, ficando a implementação a cargo dos programadores. Entretanto como as interfaces usadas como base são sempre as mesmas (ou seja, aquelas definidas pela OMG) é possível garantir a interoperabilidade entre as implementações de programadores diferentes.

A primeira versão, CORBA 1.0, surgiu em 1991, quando foi definida a IDL (*Interface Definition Language* – Interface de definição de linguagem) e a API (*Application Programming Interfaces* – Interface de programação de aplicativo). Porém a interoperabilidade entre os objetos desenvolvidos em linguagens de diferentes fabricantes, só foi possível em 1994, com a segunda versão, CORBA 2.0, quando foi implementada no mesmo o IIOP (*Internet Inter-ORB Protocol*) [11]. Existem outros protocolos para comunicação entre ORB, porém o IIOP é o mais popular deles.

3.2.1. OMA - *Object Management Architecture*

Para aplicar CORBA é necessário adotar a Object Management Architecture – OMA que consiste de um *framework* o qual define as funções suportadas pela arquitetura com base nas especificações da OMG. A OMA constitui a base para a construção de aplicações com objetos distribuídos e da interoperabilidade entre aplicações em ambientes heterogêneos e homogêneos.

Para compreender a estrutura da arquitetura do CORBA deve-se compreender o modelo de referência da OMA (Figura 2), o qual consiste dos seguintes componentes:

[1] Código que desempenha o papel de intermediário entre a plataforma de hardware e as aplicações distribuídas dos usuários

- ✓ **ORB (Object Request Broken)**: camada responsável pela comunicação entre os objetos utilizados nos pedidos e respostas das requisições, em ambientes distribuídos. É o principal componente na comunicação entre objetos e clientes.
- ✓ **Object Services**: É o conjunto de serviços disponíveis para implementação e uso dos objetos em uma aplicação. Um exemplo de serviço é o *Naming Service*: o qual permite ao cliente encontrar objetos através do nome.
- ✓ **Common Facilities**: consiste em um conjunto de serviços compartilhados por várias aplicações, porém menos essenciais que os Object Services.
- ✓ **Domain Interfaces**: interface específica para cada domínio de aplicação.
- ✓ **Application Interfaces:** interfaces não-padronizadas, específicas de cada aplicação.

A outra parte do modelo de referência dedica-se ao uso das *Application Interfaces* e introduz a noção dos *frameworks* de objetos específicos por domínio. Esses *frameworks* são coleções de objetos que fornecem uma solução integrada dentro de uma aplicação ou domínio de aplicação, podendo ser caracterizado pelo desenvolvedor. Todas as interfaces são definidas através de IDL.

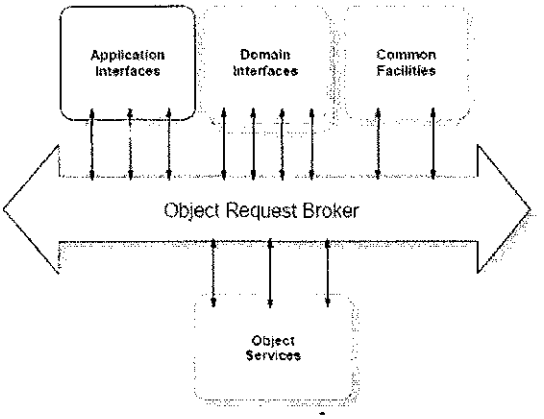

Figura 2 – Modelo de referência OMA

A função da arquitetura CORBA inserida na OMA é implementar a função ORB, a partir de um mecanismo padronizado para a definição das interfaces entre os componentes.

3.2.2. Implementação do objeto CORBA

Dado que CORBA é uma arquitetura orientada a objeto, é necessário um modelo de implementação do objeto e esse consiste de duas partes: a execução e a construção do modelo. A execução é o modo como os serviços são realizados e a construção descreve como o serviço é definido.

Para execução de um serviço é feita uma requisição na qual o parâmetro de entrada é enviado para o destinatário, uma vez recebido retorna, em seguida, um valor de saída. O código executado para realizar um serviço é denominado método.

Na construção do modelo são previstos os mecanismos para realização de uma requisição, esses mecanismos incluem definições do estado do objeto e definições de

métodos. Além disso, é necessário fornecer as informações necessárias para criar um objeto e como associar esse novo objeto ao método apropriado.

A Tabela 5 compara um modelo de objetos convencional e um modelo de objetos CORBA.

Tabela 5 – Modelo de objetos convencional e modelo de objetos CORBA

Modelo de objetos convencional	Modelo de objetos CORBA
Classe: Conjunto de dados ou objetos	Não suporta o conceito de classe. O Conjunto de objetos é uma função apoiada pela interface.
Tipo	*Interface*
Tipo de herança: Inclui "herança" de operações e de implementações,	Suporta apenas a herança de *interface*
Suporta: Exceções, sobrecarga, atraso de envio	Não suporta: Exceções, sobrecarga, atraso de envio
Identidade do objeto: Contém identificação dos objetos	*Object reference*: é uma extensão do *object identity* e inclui informações adicionais
Encapsulamento	Encapsulamento

A implentação de CORBA consiste em implementar o objeto(ORB). Para isso é necessária uma linguagem de descrição (IDL) e analisar a relação entre cliente e servidor (*Stub* e *Skeleton*).

ORB – Intermediário da requisição de objeto

O *Object Request Broker* – ORB [8], componente fundamental da arquitetura CORBA, consiste em um elemento de software cujo propósito é realizar a comunicação entre objetos. A especificação (www.omg.org) propõe entre outras capacidades a de localizar um objeto remoto a partir de uma referência.

Utilizando o ORB, os objetos fazem requisições e recebem respostas uns dos outros, seja em uma mesma máquina, seja através de uma rede. Com o ORB, o cliente não precisa estar ciente dos mecanismos usados para a comunicação ou ativação de um objeto, as propriedades de sua implementação ou mesmo sua localização. O ORB forma, portanto, a base para se construir aplicações distribuídas e para a interoperabilidade entre aplicações em ambientes homogêneos ou heterogêneos.

IDL – Linguagem de definição de interface

A Linguagem para Definição de Interface (*Interface Definition Language)* – IDL [9], tem como função especificar as interfaces entre os objetos CORBA. Esta interface é responsável pelas possíveis operações do cliente ao objeto. Com a utilização da IDL, CORBA assegura a independência de linguagem.

IDL não é uma linguagem procedural, ou seja, serve para definir apenas as interfaces, e não as implementações. Comparada à linguagem de programação C++ é equivalente aos cabeçalhos. A especificação da IDL é responsável por assegurar que os dados sejam corretamente trocados entre linguagens de programação diferentes.

A IDL, da mesma maneira que a maioria das linguagens de programação define tipos primitivos que incluem números inteiros, caracteres, números de ponto flutuante, *strings*, tipos booleanos e constantes, entre outros. Além disso, os tipos primitivos podem ser agregados em novos tipos, a partir de *unions* e *structs* (Figura 3).

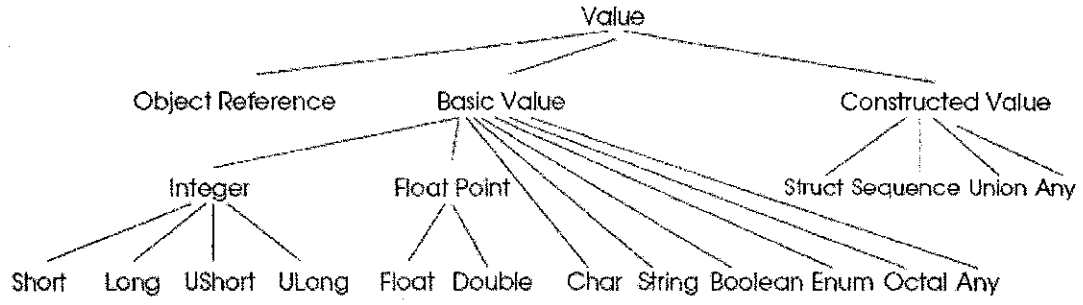

Figura 3 – Hierarquia dos tipos de CORBA IDL

No Quadro 15 é apresentada parte de um código, que exemplifica o uso da IDL. Este exemplo foi extraído do projeto que precedeu este estudo. O código apresenta um trecho da interface de um tanque cilíndrico, no qual constam a identificação do tanque e alguns de seus atributos. Em seguida a informação é apagada, e é criado outro tanque.

Quadro 15 - Exemplo tanque cilíndrico

```
interface tanque_cilindrico{
    readonly attribute string id_tanque_cilindrico;
    readonly attribute boolean injetor_tanque_cilindrico;
    void medidor_num_tanque(in float valor_medidor);
    void medidor_graf_tanque(in float nível_medidor);
};
interface checar_tanque_cilindrico: tanque_cilindrico{
    readonly attribute float valores_tanque;
};
interface supervisorio
    tanque_cilindrico newtanque_cilindrico (in string id_tanque);
    void deletetanque_cilindrico(in tanque_cilindrico P-800A);
    checar_tanque_cilindrico newchecar_tanque_cilindrico (in string id_tanque, in float
valor_medidor);
};
```

Stubs e Skeletons

CORBA mantém as noções de cliente e servidor, porém qualquer objeto CORBA pode trabalhar nas duas condições. O objeto será servidor quando estiver disponibilizando serviços para outros objetos e será cliente, quando solicitar serviços de outro.

Stubs são as implementações do cliente. No caso é um trecho de código que permite ao cliente ter acesso a um componente do servidor. Já os *Skeletons* são trechos de código que devem ser preenchidos quando um servidor é implementado. Os *Stubs* e *Skeletons* são gerados quando a interface IDL é compilada, não sendo necessário codifica-los manualmente.

3.3. OPC – OLE para controle de Processo

Um grupo de empresas OPC *Foundation*, com o intuito de melhorar a interoperabilidade e a comunicação entre dispositivos propôs a arquitetura OPC – *OLE for Process Control*. Essa arquitetura permite a comunicação entre produtos de diferentes fabricantes e define um conjunto de interface padrões, aumentando assim a eficiência na troca de dados entre componentes de softwares.

Desde sua criação diversas especificações foram publicadas. A primeira especificação, publicada em agosto de 1996, foi denominada OPC *Specification*, que após sua primeira atualização, passou a ser denominada OPC *Data Access Specification*.

As especificações do padrão OPC são subdivididas em *Custom Interfaces* e *Automation Interfaces* (Figura 4). As *Custom Interfaces* são usadas com linguagens de programação que utilizam o conceito de apontadores de função, como C e C++ . E as linguagens que não suportam esse conceito são as *Automation Interface*, que utilizam os métodos que chamam as funções através do nome, ao invés de utilizarem apontadores.

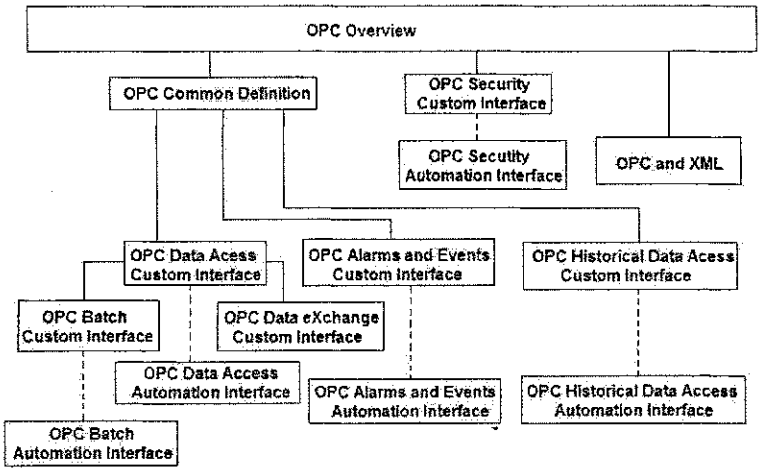

Figura 4 – Visões da especificação do padrão OPC

A base do OPC é o OLE e o DCOM. O OLE - *Object Linking and Embedding* foi desenvolvido pela Microsoft como modelo para comunicação entre aplicativos, visando suprir a necessidade de integração entre diferentes aplicações dentro da plataforma *Windows*. E o DCOM consiste em um modelo de objetos para a implementação de aplicações distribuídas através da relação de comunicação Cliente-Servidor. Um cliente pode utilizar vários servidores ao mesmo tempo e um servidor pode fornecer sua funcionalidade a vários clientes ao mesmo tempo.

Com o uso do DCOM os objetos são instanciados de forma distribuída e seus serviços e métodos são acessíveis por diferentes programas, independentemente do ambiente de programação (linguagem, compilador, versão, etc.) em que os mesmos foram criados. Para isso é utilizada a linguagem IDL.

Com o crescente uso da Internet, a fundação OPC criou o OPC-XML acabando assim com a limitação de comunicação apenas para rede local e permitindo a construção de aplicações através da Internet. O XML-OPC consiste na integração entre a linguagem XML e o padrão OPC.

3.3.1. Visão geral, definições comuns e interfaces OPC

A visão geral do padrão OPC ilustrada na Figura 4, especifica um conjunto de definições e interfaces comuns aos servidores e clientes.

A especificação OPC *Overview* define os campos de aplicação, o sistema de desenvolvimento e as especificações existentes no padrão. A especificação de definição comum e interfaces de *OPC* descreve as interfaces IOPCCommon, IOPCShutdown e IOPCServerBrowser, além das definições para instalação e configuração de registro do sistema.

3.3.2. Arquitetura OPC

A arquitetura OPC é composta por três tipos básicos de objetos: item, grupo e servidor.

Os itens são os elementos mais simples na especificação e representam conexões a pontos de entrada ou saída. São os meios de acesso a valores. Uma única variável de entrada ou saída pode ser representada por itens diferentes, com propriedades distintas, e compartilhada por mais de um cliente.

Um conjunto de itens com características comuns forma um grupo. Os grupos têm como função, efetuar as operações de leitura e escrita, enviando as atualizações para seus clientes, periodicamente ou por exceção. A definição dos grupos presentes em um servidor é realizada pelos clientes OPC. Na maioria das vezes, os grupos são privados e o acesso a eles é um privilégio do criador. Entretanto, em alguns casos, o servidor pode disponibilizar grupos públicos, que podem ser compartilhados por vários clientes. Os grupos assumem um papel importante na interação cliente-servidor. Esta interação é realizada através de interfaces, que reúnem conjuntos de funções relacionadas.

As principais interfaces de um grupo OPC são:

- ✓ *IOPCGroupStateMgt*: Responsável pela administração do grupo e de suas propriedades;
- ✓ *IOPCPublicGroupStateMgt*: Interface que complementa a IOPCGroupStateMgt, em servidores que suportam grupos públicos;
- ✓ *IOPCSyncIO*: Interface com funções de leitura e escrita síncronas;
- ✓ *IOPCAsyncIO*: Interface com funções de leitura e escrita assíncronas;
- ✓ *IOPCItemMgt*: Responsável pela adição, remoção e alteração de itens;
- ✓ *IDataObject*: Interface que estabelece as conexões entre servidores e clientes;

Os servidores podem ser vistos como estruturas de armazenamento de grupos e apresentam como função básica, o gerenciamento das conexões com as fontes de dados. Além disso, é o servidor que implementa a estrutura de endereçamento capaz de associar itens com variáveis reais. Assim como nos grupos, as funcionalidades dos servidores são acessadas a partir de interfaces, dentre as quais, podemos citar:

- ✓ *IOPCServer*: Interface que objetiva a criação e a remoção de grupos, possibilitando o acesso às propriedades globais de um servidor;
- ✓ *IOPCServerPublicGroups*: Interface complementar a IOPCServer, em servidores que suportam grupos públicos;
- ✓ *IOPCBrowseServerAddressSpace*: Responsável pelas funções de navegação através da estrutura de endereçamento do servidor, permitindo a criação automática de itens sintaticamente corretos;
- ✓ *IPersistFile*: Interface que permite aos clientes selecionarem configurações de servidores armazenados em disco;

Os servidores podem ser implementados como bibliotecas, *Dynamic-Link Libraries* (.DLL), ou como programas executáveis independentes (.EXE). As DLLs, executam no mesmo processo que o cliente, desta maneira, a configuração do servidor como DLL proporciona um alto desempenho.

Quando implementado como programas independentes, os servidores podem ser executados tanto na mesma máquina que os clientes - servidor local, quanto em outro nó de rede - servidor remoto. Esta última alternativa utiliza mecanismos DCOM que, quando apropriadamente configurados, tornam a infra-estrutura de rede completamente transparente para clientes e servidores.

Outra possibilidade de implementação são os servidores híbridos, constituidos em parte por DLL's e em parte como programas executáveis independentes. Os servidores híbridos são capazes de equilibrar desempenho e flexibilidade à custa de uma maior complexidade de desenvolvimento.

3.3.3. Acesso de dados OPC – OPC DA

OPC DA surgiu logo após OPC e consiste em um mecanismo de comunicação cliente/servidor, mas não provê meios de implementação de algoritmos de controle. Um servidor DA terá normalmente acesso a dispositivos utilizando *drivers* e protocolos apropriados e permite que várias aplicações clientes DA acessem estes dispositivos de um modo uniforme. Por exemplo em um sistema industrial através da comunicação em tempo real.

A especificação DA se diferencia da primeira versão ao utilizar os conceitos: espaço de nomes – *namespace* e a hierarquia de objetos OPC (Figura 5) [6].

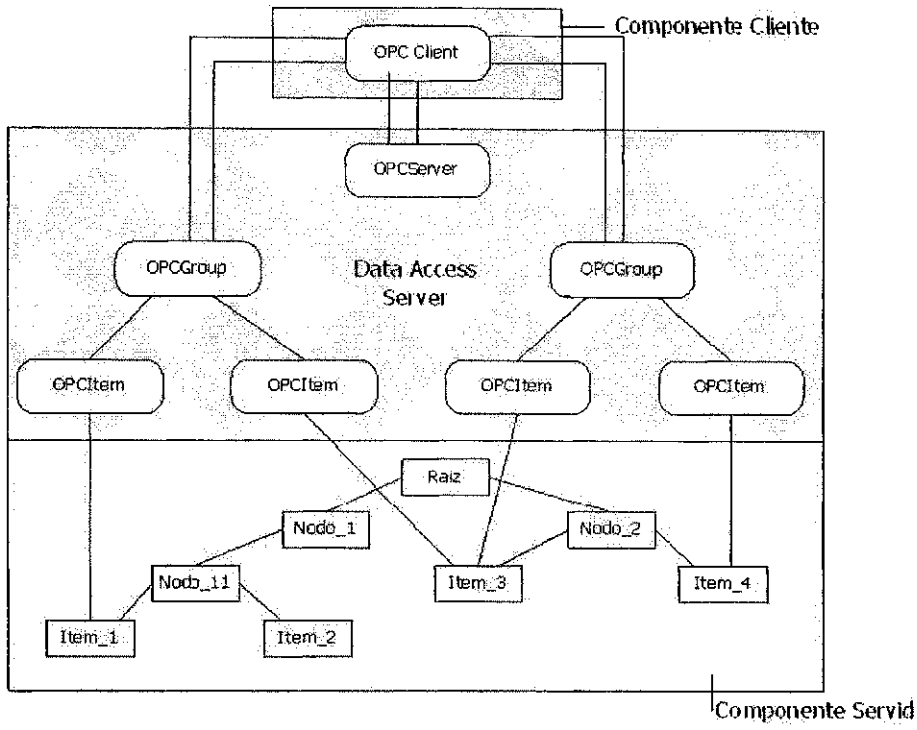

Figura 5 – Servidor DA – *Namespace* e Hierarquia de Objetos

Mecanismos de comunicação:

No OPC DA a leitura dos dados pode ocorrer de três maneiras:

- ✓ Leitura síncrona: Uma nova leitura é realizada, após confirmação da execução da leitura anterior. Só é utilizada para acesso a dados rápidos, para evitar bloqueios ao cliente;
- ✓ Leitura assíncrona: O cliente chama o método e imediatamente recebe a resposta do servidor. É utilizada por garantir melhor desempenho;
- ✓ *Refresh*: o cliente lê todos os objetos OPCItem ativos de um objeto OPCGroup ativo.

Formato dos dados OPC:

Os dados OPC consistem de três componentes:

- ✓ O dado atual: Todas as especificações OPC usam tipos de dados DCOM denominados VARIANT, que podem conter número, *strings*, tipos de dados definidos pelo usuário, ou vazio e nulo. O tamanho do VARIANT é de 16 bytes;
- ✓ Marcação de tempo ou *Timestamp*: Esta informação é fornecida pelo servidor, após a leitura dos dispositivos de campo ou por geração interna. O tamanho equivale a um valor inteiro longo de 8 bytes;
- ✓ Informação do estado do dado: compreende o estado da informação que será transmitida, é um dado de 2 bytes.

Criação de Objetos OPC:

No início o cliente de acesso aos dados – *Data Access Client*, acessa o objeto OPCServer que está no topo da hierarquia de objetos. Em seguida o cliente procura no *namespace* as informações sobre a estrutura e captura os identificadores que serão usados na criação dos objetos OPCItem.

O servidor é responsável por criar o *namespace* e por especificar os parâmetros de configuração necessários. Ao criar um Item é necessário fazer a descrição e identificação. As propriedades e os detalhes de acesso podem ser livremente definidos. Como DCOM suporta UNICODE é possível criar identificadores do *namespace* com qualquer conjunto de caracteres.

Em seguida o cliente pode criar OPCGroup e OPCItem, todo dado OPC é denominado Item e pode ser agrupado em Grupos OPC. Quando o cliente faz a requisição da criação de um OPCGroup, ele transmite valores para os seguinte parâmetros do servidor:

- ✓ Symbolic Name: Nome simbólico;
- ✓ RequesedUpdateRate: A taxa de atualização que influência na velocidade de leitura dos valores dos objetos OPCItem será repetida;
- ✓ PercentDeadband: A banda morta indica qual a variação máxima do valor de um dado antes que ele seja considerado alterado e, consequentemente, enviado para o cliente;
- ✓ ActiveState: O estado ativo informa o estado do objeto. Se um objeto OPCGroup não está ativo, as variáveis do processo não são lidas automaticamente;

Os valores passados para o servidor para criação dos objetos OPCItem são:

- ✓ ItemID: Identifica claramente o item no *namespace*;
- ✓ ActiveState: O estado ativo informa o estado do objeto. Apenas objetos ativos têm seu valor lido automaticamente;
- ✓ RequestDataType: Tipo de dado do item;
- ✓ AccessPath: Campo opcional com o detalhamento do caminho de acesso;
- ✓ ClientHandle: Indica o cliente que está criando o item.

O servidor retorna os seguintes parâmetros para o cliente após terminar a execução do método:

- ✓ CanonicalDataType: indica o tipo de dado;
- ✓ ServerHandle: indica o servidor que está recebendo o método.

Propriedades do Item:

No processo de comunicação além do valor das variáveis o sistema necessita de outras informações como o nome dos dispositivos e calibração. Por esta razão, propriedades foram introduzidas a partir da versão 2.0 da Especificação de Acesso aos Dados - *Data Access Specification*. Toda propriedade é composta por três partes:

- ✓ PropertyID: identificador numérico da propriedade;
- ✓ Tipo de dados da propriedade;
- ✓ Descrição em Inglês.

A faixa de valores do identificador de propriedades é subdividida em três sub-faixas:

- ✓ 1-99 OPC-Specific Properties: propriedades especificadas pelo padrão OPC;
- ✓ 100-4999 Recommended Properties: propriedades recomendadas pelo padrão e que podem ser suportadas pelos servidores OPC;
- ✓ 5000-65535 Vendor-specific Properties: propriedades definidas pelos fabricantes de servidores, usuários e outros.

3.3.4. OPC XML-DA

A especificação OPC XML-DA é semelhante à especificação OPC DA. Para o estudo da especificação OPC XML-DA é necessário o conhecimento das interfaces:

- ✓ OPC *Data Access Custom Interface Standard*;
- ✓ OPC *Common Definitions and Interfaces*;
- ✓ OPC *Security Custom Interface Standard.*

Além dos serviços originalmente oferecidos pelo OPC DA o padrão OPC XML-DA oferece:

- ✓ Status (GetStatus, GetStatusResponse): pode ser utilizado pelo cliente para checar a disponibilidade do protocolo de transporte e do próprio serviço. Ainda, serve para questionar sobre informações de serviço como, por exemplo, informação sobre o desenvolvedor do produto ou o estado do serviço;
- ✓ Read (Read, ReadResponse): leitura dos itens de dados, retorna a qualidade do dado e o *timestamp* para um ou mais itens;
- ✓ Write (Write, WriteResponse): escrita de um ou mais itens de dados;
- ✓ Subscription (Subscribe, SubscribeResponse): estabelece uma conexão para troca automática de dados entre cliente e servidor;
- ✓ Subscription Polled Refresh (SubscriptionPolledRefresh, SubscriptionPolledRefreshResponse): o cliente inicia um pedido pelos itens de dados de uma assinatura;
- ✓ Subscription Cancel (SubscriptionCancel, SubscriptionCancelResponse): uma assinatura deve ser cancelada;
- ✓ Browse (Browse, BrowseResponse): pesquisa o *namespace* do servidor por informações dos nomes de todos os itens (*tags*) disponíveis;
- ✓ Get Properties (GetProperties, Get PropertiesResponse): pesquisa por propriedades dos valores.

A interface que fornece estes serviços é definida utilizando WSDL[1] – *Web Service Definition Language*. Por convenção, o WSDL define o *namespace* OPC XML-DA e o *namespace* do *Schema* XML. Os *schemas* base são definidos por OPC XML-DA. Estes são contidos por outros *schemas* para descrever as mensagens a serem transportadas. Todos os atributos são opcionais, a menos que sejam explicitados como necessários. A descrição do serviço explicita o comportamento esperado para os atributos que não estão incluídos. *Schemas* são utilizados para definir tanto os pedidos quanto suas respostas e o formato dos tipos utilizados pela especificação.

A próxima seção apresenta uma análise das arquiteturas e protocolo estudos, levando em consideração a interoperabilidade, velocidade de comunicação, segurança, complexidade, dentre outros critérios.

[1] É uma linguagem baseada em XML utilizada para descrever Web Services. Trata-se de um documento escrito em XML que além de descrever o serviço, especifica como acessá-lo e quais as operações ou métodos disponíveis.

4. Abordagem crítica das arquiteturas e protocolo

Após o estudo das arquiteturas e protocolo apresentados na seção precedente foi realizada uma análise de sua adequação aos propósitos deste trabalho. Este capítulo apresenta esta análise, que não representa um estudo comparativo uma vez que SOAP é um protocolo e para implementá-lo é necessário enquadrá-lo em uma arquitetura com *web service*, enquanto que CORBA e OPC são arquiteturas completas.

Na análise foram considerados um conjunto de critérios, os quais são apresentados a seguir em paralelo com a discussão da adequação ou não dos conceitos estudados.

4.1. Interoperabilidade

O protocolo SOAP e a arquitetura CORBA são independentes de plataforma e de sistema operacional. Já OPC apresenta dificuldades na independência de sistema operacional, pois como foi desenvolvido pela *Microsoft* baseado em DCOM o sistema operacional utilizado é o *Windows* [10].

SOAP e OPC podem ser escritos em diversas linguagens já CORBA não admite a linguagem C#, desenvolvida pela *Microsoft*, a qual detém uma grande fatia do mercado na área de automação.

SOAP apresenta problemas de interoperabilidade nas suas ferramentas de desenvolvimento, pois embora tenha amplo suporte ainda há incompatibilidades entre suas diferentes implementações.

CORBA e OPC suportam grandes aplicações, diferentemente de SOAP.

OPC-DA não suporta a comunicação através da Internet; sendo restrito à comunicação em redes locais.

4.2. Velocidade de comunicação

No SOAP a velocidade de transferência de informação é menor que no CORBA [18]. O tempo de resposta e a perda de informações no SOAP são maiores que no CORBA [10].

Em OPC a velocidade de comunicação vai depender da aplicação. OPC DA é mais rápido que CORBA, porém OPC XML-DA é mais lento, pois é necessário processar o arquivo XML.

4.3. Segurança

SOAP pode ser utilizado potencialmente, em combinação com vários protocolos de transporte de dados, como HTTP, SMTP e FTP. Dessa forma a implementação da segurança depende do usuário. Quando bem utilizado oferece um bom nível de segurança

e pode atravessar *firewalls*. Sua limitação é não definir um mecanismo para criptografia do conteúdo de uma mensagem, o que evitaria o acesso ao conteúdo da mensagem [10].

Em CORBA existe uma falha de segurança no *firewall*, pois para fazer a comunicação é necessário abrir uma porta para cada serviço, tornando-o vulnerável a ataques Em OPC existe uma especificação para definições de segurança, sendo considerada uma arquitetura segura. [6]

4.4. Complexidade

Dado que SOAP é estruturado usando XML, é leve e de fácil compreensão. Por outro lado, CORBA apresenta um alto nível de complexidade de implementação devido às suas API's e sua interface é definida de modo complexo. Em muitas situações suas linhas de comando são em excesso e possuem pouca funcionalidade [8], OPC é fácil de entender e bem aceito (popular) devido á relação com a *Microsoft*, porém apresenta um grau elevado de complexidade na implementação.

4.5. Outros critérios analisados

Quando existe mais de um servidor de dados o cliente SOAP envia a solicitação a todos os servidores, porém não a um especificamente. Em contraste CORBA utiliza *Stubs* e *Skeleton* que podem enviar diretamente para um servidor específico e em OPC utiliza o namespace para referenciar servidor desejado. Em SOAP não existe garantia quanto à entrega da mensagem. Quando uma mensagem estiver sendo transferida, se o sistema falhar, não há meios de saber se é necessário reenviar a mensagem.

Da revisão bibliográfica foi constatado que CORBA está em desuso [8], pois não permite atualizações graduais. A cada nova versão é necessário mudar o cliente e o servidor. Por fim a depuração de sua programação é difícil.

OPC DA é mais utilizado para comunicação de sistemas industriais em tempo real, porém uma limitação é que alarmes e eventos fazem parte de outra especificação, sendo necessário na implementação ter acesso a diferentes especificações.

OPC XML-DA oferece a vantagem de permitir a comunicação via Internet, em relação à OPC DA. No entanto, existem desvantagens em relação ao desempenho que é inferior em relação à OPC DA devido ao processamento exigido na utilização dos arquivos XML. Além disso, por operar em rede, além dos limites do *firewall*, apenas o *firewall* não poderia garantir a segurança necessária como em OPC DA.

A principio a adoção de OPC XML-DA poderia ser recomendada, porém os requisitos do projeto, não exigem tantos recursos. Ao longo deste estudo surgiu a possibilidade de adoção de XML-RPC que será tratado na próxima sessão como a solução mais adequada e, portanto a que foi escolhida.

Na próxima seção será descrito a solução utilizada para o estudo de caso, além apresentar a nova arquitetura do projeto após a implementação do protocolo XML-RPC.

5. Proposta de solução

A partir do estudo apresentado na sessão anterior e considerando o requisito inicial do projeto que propunha a utilização da notação XML em todas as etapas da comunicação entre cliente e servidor, buscou-se uma solução que contemplasse este requisito.

O protocolo XML-RPC se apresentou adequado ao cumprimento deste requisito além de exigir pequenas alterações na arquitetura no estudo de caso proposto. Esta sessão apresenta a aplicação deste protocolo na formulação das mensagens trocadas entre cliente e servidor em um ambiente de automação.

5.1. O protocolo XML-RPC

RPC - *Remote Procedure Call* [20] define um protocolo para execução remota de procedimentos em computadores ligados em rede. O RPC não especifica como a mensagem é enviada de um processo para outro, mas somente especifica e interpreta a mensagem.

XML-RPC [16] consiste em um procedimento de chamada remota que utiliza HTTP – *HyperText Transfer Protocol* como transporte e XML como o corpo da requisição (codificação). O XML-RPC foi projetado para ser simples e permitir que estruturas de dados complexas possam ser transmitidas, processadas e devolvidas consecutivamente.

Quando uma chamada é realizada o processo é executado no servidor e retorna valores em XML. Os parâmetros do processo podem ser simples como escalares, números, palavras, entre outros e podem ser mais complexo como conjuntos e lista de estruturas.

Exemplo de requisição:

No Quadro 16 apresenta-se um exemplo de requisição em XML-RPC:

Quadro 16 - Requisição XML-RPC

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
   <methodName>examples.getStateName</methodName>
   <params>
      <param>
         <value><i4>41</i4></value>
         </param>
      </params>
   </methodCall>
```

Cabeçalho:

Toda mensagem XML-RPC apresenta um cabeçalho contendo o formato da URI - *Universal Resource Identifier* na primeira linha, que é opcional. Entretanto se o servidor receber uma mistura de requisições HTTP, a URI ajuda a distribuir as requisições. No exemplo, a URI é /RPC2, indicando ao servidor que a requisição é para "RPC2" responder. O cabeçalho também contém:

- ✓ *User- Agent:* onde o cliente é especificado.
- ✓ *Content-Type*: tipo de conteúdo (no caso é texto/xml).
- ✓ *Content-Length:* tamanho do conteúdo.

O formato do arquivo:

A linguagem utilizada é XML. A estrutura é simples e denominada: <methodCall>. O <methodCall> pode conter o sub-item <methodName>, a palavra (*String*), e o nome do método a ser chamado. A palavra pode conter apenas caracteres, letras maiúsculas e minúsculas A-Z, caracteres numéricos 0-9, ponto, dois pontos e barra. O servidor é responsável por interpretar as chamadas (methodName).

O <methodCall> pode conter parâmetros (<params>) que podem ser do tipo escalar (<value>). A Quadro 17 ilustra os tipos que são aceitos nos parâmetros.

Quadro 17 – Tipos de valores escalares

Tag	Tipo	Exemplo
<i4> ou <int>	Inteiro de quatro bytes	15, -15
<boolean>	0 (falso) ou 1(verdadeiro)	1
<string>	Palavra	Tanque cilindrico
<double>	Real	15.34,-18.99
<dateTime.iso8601>	Data/Tempo	20080318T16:06:59
<base64>	Base64- Decodificação Binário	GNhbid0IHJlYWQ=

XML-RPC também pode conter as estruturas (*struct*), membros (*member*) e cada membro possui um nome e um valor. Ver exemplo no Quadro 18 [17].

Quadro 18 - Exemplo de uma estrutura.

```
<struct>
   <member>
      <name>lowerBound</name>
      <value><i4>18</i4></value>
      </member>
   <member>
      <name>upperBound</name>
      <value><i4>139</i4></value>
      </member>
   </struct>
```

Pode conter os arranjos (*arrays*). Os arranjos contêm elementos (*data*) e esses por sua vez podem conter valores. Ver exemplo no Quadro 19. Os *arrays* não têm nomes.

Quadro 19 – Exemplo de *array*

```
<array>
   <data>
      <value><i4>12</i4></value>
      <value><string>Egypt</string></value>
      <value><boolean>0</boolean></value>
      <value><i4>-31</i4></value>
      </data>
   </array>
```

Exemplo de Resposta:

O Quadro 20 apresenta um exemplo de resposta a uma requisição.

Quadro 20 – Exemplo de resposta à requisição

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
         <value><string>South Dakota</string></value>
         </param>
      </params>
   </methodResponse>
```

Como não ocorreu nenhum erro, retorna 200 OK. Os campos *Content-Length* e *Content-Type* são iguais ao da requisição. O Corpo da resposta é uma estrutura simples no formato XML, o <methodResponse>, pode conter parâmetros e valores. A resposta também pode conter uma falha <fault>. Normalmente isso ocorre quando o valor devolvido é de um tipo diferente do esperado. No Quadro 21 é apresentado um exemplo de falha.

Quadro 21 – Exemplo de falha

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 426
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:02 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
   <fault>
      <value>
```

```
        <struct>
          <member>
            <name>faultCode</name>
            <value><int>4</int></value>
            </member>
          <member>
            <name>faultString</name>
            <value><string>Too many parameters.</string></value>
            </member>
          </struct>
        </value>
      </fault>
  </methodResponse>
```

A opção por XML-RPC deveu-se à simplicidade de implementação e à facilidade de executar em qualquer ambiente uma vez que neste projeto os clientes e o servidor estão em plataformas e ambientes diferentes.

5.2. *Estudo de caso*

Com o propósito de validar a solução adotada, esta foi aplicada a um estudo de caso no ambiente de automação industrial do setor petroquímico. O estudo de caso consistiu em apresentar uma tela do supervisório em diferentes dispositivos, tanto do tipo *desktop* quanto dispositivos móveis, utilizando na comunicação a linguagem XML.

Para realizar este trabalho foi necessário criar um servidor contendo um banco de dados com as informações das telas do supervisório. Após cada consulta ao banco de dados os dados eram enviados em um arquivo XML, de acordo com a notação proposta no projeto original [19]. A seguir, na Figura 7, é apresentada a arquitetura original adotada na comunicação cliente-servidor.

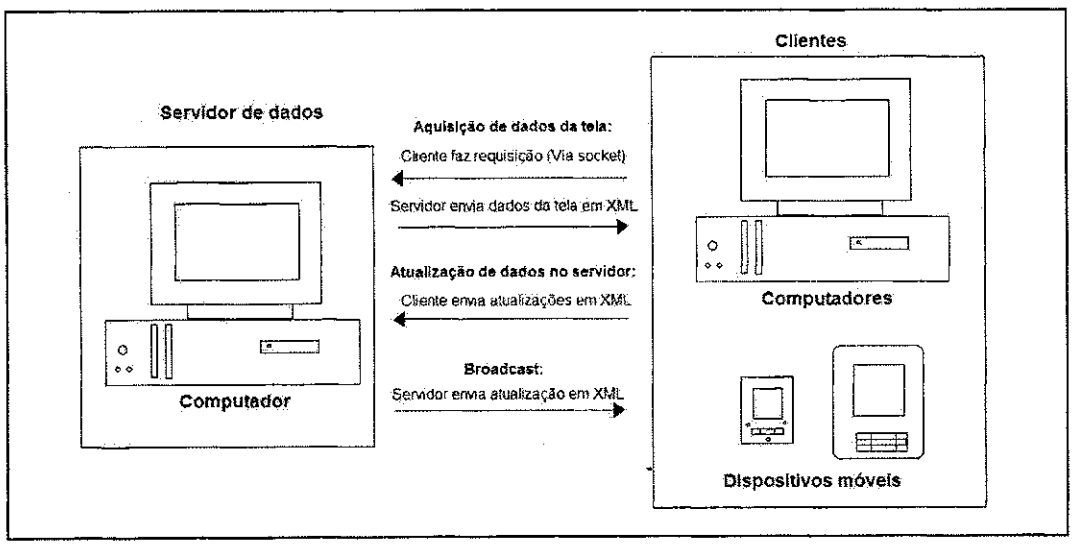

Figura 6 – Comunicação inicial entre cliente e servidor

Como parte do projeto original foi necessário propor e implementar uma arquitetura de comunicação alternativa que enviasse arquivos XML utilizando a notação proposta. Na ocasião a comunicação foi feita através de *socket* [1] e a comunicação consiste em:

- ✓ O cliente enviava o pedido XML de uma tela ao banco de dados (a requisição se dava através de um arquivo PHP);
- ✓ O servidor retornava um arquivo XML com os dados da tela. Esse arquivo era interpretado pelo cliente e apresentado na tela;
- ✓ Sempre que um valor era alterado no cliente, o mesmo criava um arquivo XML com essa atualização e o enviava ao servidor;
- ✓ Por fim, após atualizar o banco de dados, o servidor enviava a atualização para todos os clientes (*Broadcast*);

Desvantagem:

Nessa arquitetura o servidor e o cliente necessitavam ficar com a comunicação entre *sockets* sempre em espera.

Neste projeto, a pesquisa por um novo protocolo de comunicação visava assegurar a comunicação entre o cliente e o servidor através de arquivos XML. O estudo de SOAP e XML-RPC [17] levou à escolha do segundo dado a sua efetividade e simplicidade.

Com sua adoção, a comunicação entre o servidor e o cliente passa a ser através de arquivos XML's e é iniciada através de uma chamada XML-RPC. A nova arquitetura é ilustrada na Figura 8 e a comunicação consiste em:

- ✓ Inicialmente o cliente envia uma chamada XML-RPC solicitando os dados da tela a ser apresentada;
- ✓ O servidor cria uma conexão (via *socket*) e envia o arquivo XML para o cliente;
- ✓ Sempre que o cliente fizer uma alteração, ele gera o novo arquivo XML e faz uma chamada XML-RPC para o servidor indicando que existe uma atualização;
- ✓ O servidor faz uma nova conexão e solicita o arquivo de atualização;
- ✓ Após alterar os dados no banco de dados o servidor envia a atualização para todos os clientes (*broadcast*).

A partir do estudo de caso, constatou-se que a implementação do protocolo XML-RPC e a nova arquitetura de comunicação possibilitaram atingir os objetivos deste trabalho.

Um dos ganhos mais importantes é que todos os arquivos enviados na comunicação entre cliente e servidor são do tipo XML. Os arquivos gerados com informações do banco de dados utilizam à notação proposta no projeto anterior e os arquivos utilizados na comunicação XML-RPC utilizam a especificação do protocolo.

O segundo ganho é que na proposta anterior todas as aplicações do sistema implementavam interface de cliente e servidor, agora cada aplicação implementa apenas

[1] É canal de comunicação que utiliza TCP/IP entre cliente e servidor para enviar e receber dados.

umas das interfaces e todo o gerenciamento da comunicação é feito através de XML-RPC. Com essa modificação a requisição de processamento de informação torna-se menor.

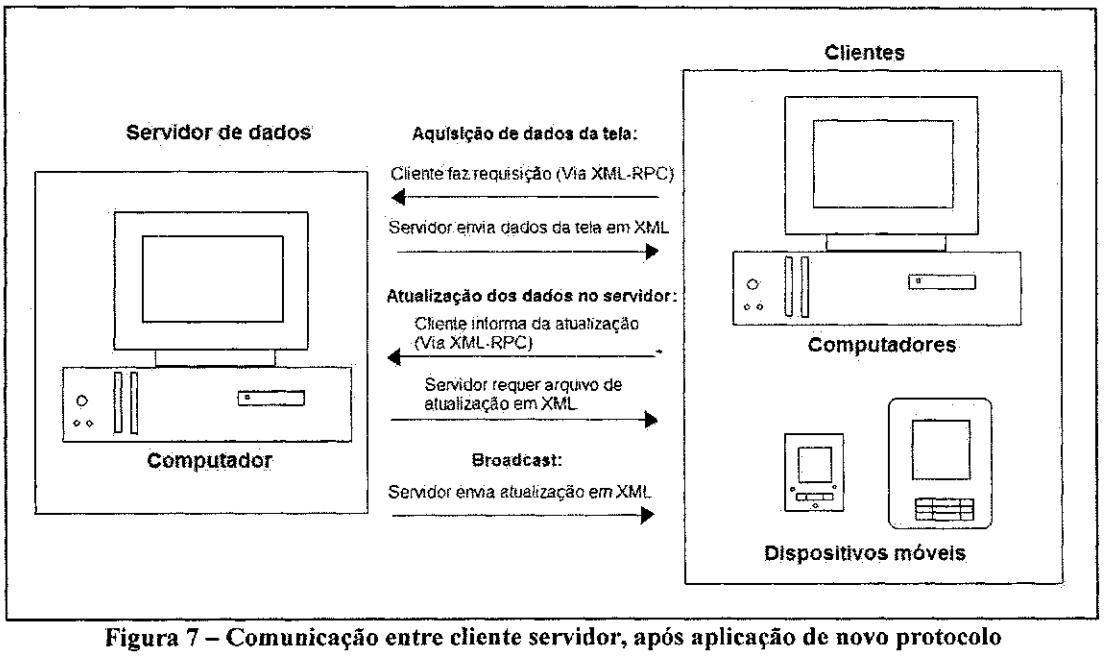

Figura 7 – Comunicação entre cliente servidor, após aplicação de novo protocolo

Códigos em XML

Para melhor exemplificar os arquivos XML que são utilizados na comunicação, o Quadro 22 ilustra o XML gerado na chamada do cliente ao servidor, solicitando os dados de uma tela do supervisório a ser apresentada. O envio desse XML é feito através de uma chamada de procedimento remoto.

Quadro 22 – XML gerado na chamada XML-RPC

```
<?xml version="1.0"?>
<methodCall>
    <methodName>tela.Nafta</methodName>
    <params>
        <param>
            <value><string>numeroIP</string></value>
        </param>
    </params>
</methodCall>
```

- ✓ "tela.Nafta": é o nome do método, no qual a tela é a função e Nafta é o nome da tela desejada;
- ✓ "numeroIP": é um parâmetro da chamada e indica o número IP do cliente para o servidor saber qual cliente fez o pedido do arquivo XML.

O Quadro 23 apresenta o XML gerado após uma consulta ao banco de dados, utilizando a notação proposta no projeto anterior.

Quadro 23 – XML gerado após consulta ao banco

```
<geral xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="completo2.1.xsd">
  <cabecalho titulo_cabecalho="NAFTA" simbolo_br="true"/>
  ...
  <nodo id_nodo="49">
      <tanque_cilindrico injetor_tanque_cilindrico="false" id_tanque_cilindrico="P-801K"
      revestimento_tanque_cilindrico="false" faixa_verde_amarela="true">
          <medidor_num_tanque_cilindrico tipo_medidor_num="temperatura"
          titulo_medidor_num="Temp." valor_medidor_num="35"
              unidade_medidor_num="oC"/>
          <medidor_num_tanque_cilindrico tipo_medidor_num="nivel"
          titulo_medidor_num="Nível" valor_medidor_num="125"
          unidade_medidor_num="m3"/>
          <medidor_num_tanque_cilindrico tipo_medidor_num="vazao"
              titulo_medidor_num="Vazão" valor_medidor_num="75"
              unidade_medidor_num="m3/s"/>
          <medidor_graf_tanque_cilindrico conteudo_medidor_graf="azul"
              nivel_medidor_graf="50"/>
      </tanque_cilindrico>
</nodo>
```

A principal diferença consiste na chamada inicial do cliente, que agora é feita através de um arquivo XML. O ganho ao utilizar XML nessa comunicação é que tanto o servidor quanto os clientes independem de plataforma. Neste estudo de caso, o servidor está na plataforma Linux e os clientes utilizam sistema operacional *Windows* (versão XP e versão CE).

6. Conclusão

O projeto da indústria petroquímica utilizado como estudo de caso e propor uma nova solução de comunicação foram à motivação para esse trabalho de conclusão. O estudo do XML foi importante, pois apresenta todas as características desejáveis para promover a interoperabilidade entre aplicativos.

O estudo do protocolo SOAP, serviu para entender, conhecer e exemplificar a utilização do XML, já que o SOAP possui uma notação XML própria. Através do estudo de CORBA tomou-se conhecimento de sua arquitetura e da utilização do ORB e da linguagem de descrição IDL. No estudo de OPC, a mais robusta das arquiteturas estudadas, permitiu conhecer como funciona sua arquitetura (item, grupo, servidor), suas especificações e como os dispositivos são integrados (dll's). Esta poderia ter sido uma solução adotada, porém implicaria em modificações profundas na arquitetura original.

O estudo de arquiteturas e protocolos de comunicação agregou conhecimentos ao projeto na forma de idéias e conceitos.

A solução XML-RPC, se mostrou válida, pois possibilitou utilizar diferentes sistemas operacionais, ambientes e fazer chamadas de procedimentos através da Internet, além de preservar a maior parte das características da arquitetura do projeto anterior. Sua interoperabilidade foi comprovada a partir do estudo de caso que utilizou na comunicação tanto o ambiente Windows quanto Linux.

Como proposta para trabalhos futuros sugere-se a implementação de mecanismos de segurança, tais como criptografia das mensagens; uma vez que estas carregam dados críticos de ambientes industriais. Outra sugestão é o desenvolvimento de um protocolo de gerenciamento de rede que possibilite a definição de controles de acessos e de prioridades.

Do ponto de vista do aluno este trabalho de conclusão complementou sua formação de engenheiro e ofereceu a oportunidade de implementar soluções para problemas reais.

7. Referências

[1] Site: http:// www.w3c.org/xml, W3C descrevendo XML acessado em Fevereiro 2008;

[2] Site: http://www.w3.org/TR/soap12-part1/, W3C descrevendo SOAP acessado em fevereiro de 2008;

[3] Site: http://www.opcfoundation.org, OPC Foundation, acessado em Janeiro 2008.

[4] The Common Object Request Broker: Architecture and Specification, Revisão 2.0, Julho 1995;

[5] CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, Artigo publicado na revista IEEE vol.35 nº02 em Fevereiro de 1997, Steve Vinoski;

[6] OPC XML-DA Specification, versão 1.0, Julho 2003;

[7] Harold, E.R.. XML 1.1 Bible. Wiley Publishing Inc. 2004.

[8] Dissertação de mestrado, tema Arquitetura de um sistema MÊS baseado em Corba, Paulo Marcelo Porto Alves Blanco, Escola politécnica da Universidade de São Paulo.

[9] Orfali, Robert et al. - Client/Server Programming with Java and CORBA – 2ª edição John Wiley & Sons, Inc. – 1998

[10] ACM Queue vol. 4, no. 5 - June 2006 by Michi Henning, ZeroC, http://acmqueue.com/modules.php?name=Content&pa=showpage&pid=396, Acessado em Fevereiro de 2008.

[11] CORBA Developer's Guide, release 3 (8.1.7), Julho de 2000.

[12] Site: http://www.omg.org, Object Management Group acessado em Fevereiro 2008.

[13] Site: http://kr.fujitsu.com/products/solutions/ebiz_platform/was/technical/tutorial/soap/index.html, acessado em fevereiro de 2008.

[14] Site: http://www.w3schools.com/soap/default.asp,W3Schools, acessado em fevereiro de 2008.

[15] Van der Vlist, E.. XML Schema. Ed. O'Reilly. 2002.

[16] http://www.xmlrpc.com, Especificações do XML-RPC, acessado em Março 2008

[17] http://ws.apache.org/xmlrpc, Apache XML-RPC, acessado em março de 2008.

[18] Artigo Performance of SOAP in Web Service Environment Compared to CORBA, Robert Elfwing, Ulf Paulsson, and Lars Lundberg do Department of Software Engineering and Computer Science, Blekinge Institute of Technology.

[19] Relatório de Projeto da Petrobrás – Representação de interface de sistemas de supervisão utilizando a notação XML, UFCG 2007.

[20] RFC 1057 RPC Remote Procedure Call Specification Version 2 Sun Microsystems, Inc. June 1988.