Interacção com uma Arquitectura de Componentes com Independência de Dispositivo e de Localização

Carlos Ângelo Pereira[1], João Correia Lopes[2]

[1] EFACEC Sistemas de Electrónica S.A., Portugal
{capereira}@se.efacec.pt
[2] Faculdade de Engenharia da Universidade do Porto, R. Dr. Roberto Frias, 4200-465 Porto, Portugal
jlopes@fe.up.pt
http://www.fe.up.pt/∼jlopes/

Resumo Recentemente têm surgido equipamentos de pequeno porte e a custos suportáveis que permitem aceder remotamente aos sistemas de informação das empresas. Estes equipamentos, tais como *Personal Digital Assistants* (PDAs) e telemóveis com WAP, tipicamente possuem capacidades de apresentação e de largura de banda de transferência limitadas e, por isso, requerem a produção de interfaces adequadas.
Os sistemas SCADA (*Supervisory Control and Data Acquisition*) são muito usados na indústria, nomeadamente em redes de energia eléctrica, e o acesso à informação por eles tratada, através de sistemas de pequeno porte, pode resultar numa mais valia importante para as empresas.
Neste trabalho propõe-se uma arquitectura, baseada em Serviços Web e em XML, que possibilita a criação e fácil manutenção de interfaces para componentes baseados no paradigma editor/subscritor em uso na Efacec para sistemas SCADA. Os componentes desenvolvidos usando SOAP e WSDL, permitem a interacção a partir de qualquer ponto da Web, mesmo através das *firewalls* instaladas nas empresas por razões de segurança. A camada de apresentação foi conseguida com recurso a um componente desenvolvido seguindo o padrão *Front Controller*, uma variante do padrão MVC (*Model View Controller*).
Por forma a validar a arquitectura proposta foi desenvolvida uma aplicação que permite a realização de ordens de manobra. Os testes conduzidos permitem concluir que os requisitos de interacção deste tipo de aplicações foram satisfeitos em conjunto com o desiderato de independência de localização e de dispositivo.

1 Introdução

A Web permite o acesso através de interfaces universais a informação localizada em qualquer ponto da Internet, usando protocolos universais. XML (*Extensible Markup Language*) estabeleceu-se como formato universal para troca de dados entre aplicações [BPM00]. O protocolo em uso na Web para interacção com a lógica do negócio das aplicações, *Hypertext Transfer Protocol* (HTTP), é actualmente permitido nas *firewalls* instaladas nas empresas para proteger a informação de acessos não autorizados, estando normalmente a porta usada por este protocolo acessível do exterior. No entanto este protocolo simples, leve e sem manutenção de estado, suporta apenas a pedidos de recursos e respectiva devolução de conteúdos estáticos ou produzidos dinamicamente. SOAP (*Simple Object Access Protocol*) usando HTTP, entre outros protocolos, foi desenvolvido para suprir limitações de acesso através de *firewalls* ou *proxys* das aplicações distribuídas desenvolvidas em CORBA, EJB ou outros *middlewares*, que são muito utilizados para interacção entre componentes distribuídos, nomeadamente em Intranets.

Serviços Web (*Web Services*) [STK00] permitem o desenvolvimento de aplicações distribuídas, com interacção entre componentes localizados em qualquer local da Internet, desenvolvidos em qualquer linguagem de programação e disponibilizados por qualquer plataforma computacional. Para isso baseiam-se em protocolos standard em uso na Internet,

permitindo a chamada remota a procedimentos através da troca de mensagens SOAP (uma descrição em XML) usando o protocolo HTTP, que assim podem atravessar *firewalls*.

Os sistemas SCADA (*Supervisory Control and Data Acquisition*) [Boy99] são bastante usados na indústria, nomeadamente em sistemas de geração de energia eléctrica ou em redes de transporte e distribuição de energia eléctrica, gás ou água. Estes sistemas possuem unidades remotas de aquisição e controlo, uma camada de comunicações e um centro de comando. Por cima dos sistema SCADA para redes de distribuição de energia eléctrica podem existir sistemas de suporte à decisão (DMS) e o acesso à informação por eles tratada através de sistemas de pequeno porte pode resultar numa mais valia importante para as empresas.

A implementação de um sistema de suporte a aplicações móveis justifica-se quando o número de utilizadores que precisa de aceder à informação a partir de dispositivos móveis é significativo. A existência deste acesso pode ainda contribuir para eliminar ineficiências, auxiliar o colaborador no exterior da empresa a decidir de forma documentada e com acesso a informação actualizada e pode ainda aumentar o grau de satisfação do cliente final.

A empresa "EFACEC — Sistemas de Electrónica S.A." desenvolve sistemas SCADA para a indústria eléctrica há alguns anos e recentemente desenvolveu um SCADA/DMS de nova geração denominado GENESys. Para suportar o desenvolvimento destas aplicações distribuídas, foi especificada e implementada uma arquitectura de software — BUS [MV01], que utiliza o paradigma editor/subscritor para troca de informação entre componentes CORBA desenvolvidos em C^{++}. Nesta empresa decorreram três teses de mestrado que procuraram alargar a utilização desta arquitectura a outras linguagens de programação e constituir pontes para a Web e para outras utilizações [Per03,Gom03,Fer03]. Na primeira destas teses, um dos autores deste artigo desenvolveu e demonstrou uma arquitectura que, utilizando XML e Serviços Web, permite interagir com independência de dispositivo e de localização, com o sistema SCADA/DMS GENESys.

São muito reduzidas as referências bibliográficas relacionadas com o tema da interacção com sistemas SCADA com independência de localização e de dispositivo, nomeadamente através de dispositivos de pequeno porte. Na área de trabalho cooperativo suportado por computador, o WebSplitter [HPN00] oferece uma arquitectura XML que permite a navegação multi-dispositivo e multi-utilizador, permitindo a participação na mesma sessão de vários tipos de dispositivos como computadores portáteis, PDAs, projectores de vídeo. Na área de SCADA, o trabalho reportado em [TT02] tem por objectivo permitir a utilização de aplicações de controlo e aquisição de informação em clientes leves, possuindo interfaces sofisticadas, com um custo de implementação e utilização reduzido.

Na Secção 2 será apresentada a arquitectura de componentes distribuídos BUS que suporta o desenvolvimento de sistemas SCADA/DMS na EFACEC. Na Secção 3 são enumerados os requisitos a cumprir e é apresentado o desenho da arquitectura proposta neste trabalho, constituída por um "Mediador Web" (Web Mediator) e por uma "Zona Desmilitarizada" (DMZ). Na Secção 4 descreve-se a implementação do servidor SOAP, que suporta a interacção da zona desmilitarizada com a arquitectura editor/subscritor — BUS, e o "Mediador Web" que, para além de ser um servidor SOAP, é também um componente que recebe e envia eventos para o BUS. Na Secção 5 é descrita a implementação do cliente SOAP e a lógica do negócio que envolve a validação e transformação XSLT do conteúdo XML vindo do BUS para WML ou HTML, dependendo do cliente Web em utilização. Na Secção 6 é apresentada a a camada de interacção com o utilizador e o seu desenvolvimento usando o padrão *Front Controller*. Na Secção 7 descreve-se uma aplicação pensada com o objectivo de validar e avaliar a arquitectura desenvolvida, sendo apresentadas imagens retiradas

da sua execução. Finalmente, na Secção 8 serão apresentadas as conclusões deste trabalho juntamente com algumas possibilidades de melhorias futuras.

2 BUS — Arquitectura de Software

Nesta secção descreve-se a arquitectura de componentes distribuídos — BUS — que suporta a implementação do GENESys, um sistema SCADA/DMS de nova geração desenvolvido conjuntamente pela "EFACEC — Sistemas de Electrónica S.A." e pela "EDP — Electricidade de Portugal".

Os sistemas SCADA (*Supervisory Control and Data Acquisition*) são complexos e distribuídos e caracterizam-se pela dispersão de utilizadores e de dispositivos que supervisionam, normalmente à distância. Estes sistemas devem garantir elevada disponibilidade e segurança, dada a natureza sensível dos sistemas controlados (com riscos económicos mas também, e mais importantes, com riscos de integridade pessoal). Num sistema SCADA não se pode perder informação, todos os eventos reportados têm de ser tratados, quer para controlo em tempo real dos processos, quer para posterior estudo do comportamento dos equipamentos. Para sistemas de grande dimensão é usual associar ao sistema SCADA sistemas de apoio à decisão, como por exemplo o DMS (*Distribution Management System*) para redes de energia eléctrica.

O BUS é uma arquitectura de software, especificada e implementada de forma a dar suporte às características do sistema SCADA/DMS GENESys, que utiliza o paradigma editor/subscritor para troca de informação entre componentes CORBA desenvolvidos em C++ [MV01]. Um sistema distribuído desenvolvido com recurso ao BUS é constituído por um conjunto de componentes que colaboram entre si através da troca de eventos. Um componente só envia ou recebe eventos de acordo com os tópicos que publica ou que subscreve, sendo os eventos entregues ao componente numa *callback*. A relação entre tópicos e eventos é especificada num meta-modelo de eventos.

Um componente é a unidade lógica de processamento fundamental do sistema, fornecendo uma funcionalidade, ou parte dela, e os componentes colaboram entre si para oferecer o conjunto de funcionalidades que se espera dum sistema. A colaboração entre componentes assenta no "modelo Push". O componente publicador faz o *push* de um evento para o BUS; baseado na configuração do meta-modelo, o BUS decide qual o tópico de entrega do evento e encaminha-o, pela informação do tópico, para todos os componentes subscritores desse tópico. O encaminhamento de eventos é assíncrono e desacoplado, isto é, a partir do momento em que um componente entregou o evento ao BUS, está livre para continuar com o seu próprio processamento. O componente não tem qualquer conhecimento dos potenciais interessados no evento: os componentes que subscrevem o tópico. De igual forma não tem qualquer controlo sobre o momento de entrega dos eventos aos componentes subscritores.

Um evento é um objecto com estado, constituindo a quantidade mínima de informação que pode ser trocada entre componentes. É definido por um nome e constituído por pares atributo/valor, com significado para o utilizador humano. O evento tem uma representação serializada em texto, o "formato stringuificado" [Mar00], para poder ser passado de acordo com o "modelo Push". Os objectos que representam o evento devem ser capazes de o construir a partir desse formato, mas também devem saber gerar essa representação.

Uma variante do evento, o "Data Request", pode ser visto como um serviço; é um pedido específico de informação realizado por um componente, que recebe como resposta dados que satisfazem esse pedido, orientados para o emissor do pedido. Um "Data Request" contém um evento e mantém as características de objecto que pode ser enviado por

componentes. Possui, no entanto, duas características ligadas ao conceito de serviço: URN (*Universal Request Name*) que identifica o pedido de forma universal e RN (*RequestName*), o nome do pedido utilizado para diferenciar pedidos do mesmo tipo, caso necessário.

Sob o ponto de vista de implementação, o BUS é uma *framework* desenvolvida em C++, que se encontra construída no topo do ORB (*Object Request Broker*) Orbix. Esta *framework* estende as funções do ORB de forma a suportar um modelo de eventos de publicação/subscrição que permita uma comunicação baseada em ligações ponto-a-ponto.

3 Arquitectura Proposta

Este trabalho pretende desenhar, implementar e validar uma arquitectura que permita interagir, com independência de dispositivo e de localização, com o sistema SCADA/DMS GENESys, nomeadamente a partir de dispositivos de pequeno porte e afastados geograficamente.

Assim, um dos requisitos óbvios da arquitectura procurada é o de suportar o envio e recepção de eventos com o BUS do sistema GENESys. Uma vez que os eventos são representados num formato proprietário é necessário traduzi-los para um formato universal que permita a utilização de processadores gerais standard, que podem ser retirados gratuitamente da Web e usados na lógica de negócio. Será então necessário processar estas representações em formatos universais por forma a construir uma interface adequada ao dispositivo de interacção.

Por forma a garantir independência de localização é necessário que a arquitectura esteja acessível a partir da Web. O requisito de independência de dispositivo leva à necessidade de reconhecimento do tipo de dispositivo que faz o pedido e espera a respectiva resposta e uma separação entre as camadas de apresentação e de lógica de negócio.

Por último e não menos importante, é necessário garantir que os pedidos e respostas atravessem as *firewalls*, instaladas pelas empresas para impedir acessos não autorizados a partir do exterior.

A figura 1 ilustra a arquitectura proposta, que inclui uma divisão em duas partes: "Mediador Web" (Web Mediator) e "Zona Desmilitarizada" (DMZ). O Web Mediator é um *proxy* que recebe pedidos da camada da lógica de negócio, fala com o BUS e produz documentos XML para serem transformados e apresentados nos dispositivos de interacção. A DMZ inclui os componentes de software que suportam a apresentação multi-dispositivo e toda a restante lógica de negócio.

A DMZ inclui o servidor Web e o servidor de aplicações e encontra-se limitada por duas *firewalls*. A primeira estabelece políticas de segurança em relação à rede exterior, deixando passar os pedidos ao servidor Web e respectivas respostas. A segunda encontra-se entre a "Zona Desmilitarizada" e o sistema GENESys, que contém a informação crítica. Os utilizadores da rede exterior apenas têm acesso ao servidor Web que reside na DMZ, ficando assim garantida a segurança da informação crítica.

4 Interacção com o BUS

Por forma a que os pedidos da DMZ atravessem a *firewall* e assim possam aceder ao BUS, será desenvolvido um Serviço Web, em que os pedidos serão realizados em SOAP sobre HTTP. O cliente SOAP reside na camada da lógica de negócio, realizada recorrendo à plataforma Java, e o servidor SOAP estará do lado do sistema GENESys e, por isso, será implementado em C++.

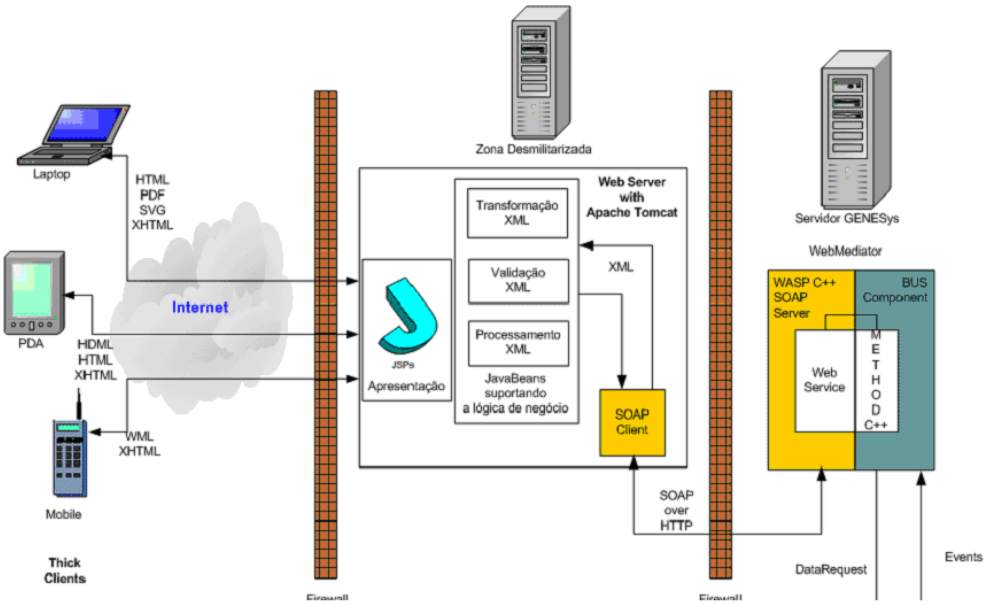

Figura 1: Arquitectura Proposta

Este Serviço Web interactua com o BUS através de um "Data Request". A sua interface (descrita por um ficheiro WSDL não incluído) inclui três parâmetros e uma string como resultado:

```
string xmlResponse DataRequest (string stringifiedEvent,
    string universalRequestName, string requestName)
```

sendo:

`stringifiedEvent` Evento "stringuificado", obedecendo às regras definidas em [Mar00], que permite definir a informação pedida (pelo nome do evento e atributos com valor associado);

`universalRequestName` URN (*Universal Request Name*) que permite identificar o tipo de pedido;

`requestName` RN (*Request Name*) que permite diferenciar pedidos do mesmo tipo;

`xmlResponse` string com um evento descrito em XML construído com base na informação recebida num evento do BUS.

Para implementação do servidor SOAP foi escolhido o WASP da Systinet [Sys02] dadas as suas vantagens de compatibilidade total com o standard SOAP 1.1, interoperabilidade comprovada com .NET, Apache AXIS e servidores J2EE, portabilidade para várias plataformas (Linux, Solaris, Windows, etc), suporte de segurança, bom desempenho e custo zero para instalação em máquinas de um só processador.

Para além da classe que implementa o Serviço Web, o Web Mediator tem ainda de ser um componente do BUS para enviar o "Data Request" e esperar pelo respectivo evento resposta. Assim, o Web Mediator desempenha as seguintes funções:

Componente do BUS Acesso à API BUS conseguindo subscrever e publicar tópicos, enviar e receber eventos de outros componentes do GENESys e processar essa informação em "callbacks";

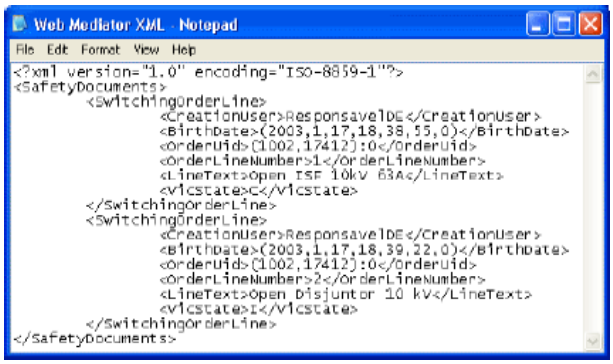

Figura 2: XML produzido pelo Web Mediator

Servidor SOAP Ouvir os pedidos SOAP e invocar a instância que os processa;

Transformar eventos em XML Utilizar a API do BUS para processamento de eventos, conseguindo fazer a sua transformação num documento XML (por exemplo, o XML da Figura 2).

5 Camada da Lógica do Negócio

A camada da lógica de negócio, para além de conter o cliente do Serviço Web disponibilizado pelo Web Mediator, contém ainda as classes Java que realizam a validação e a transformação XSLT do conteúdo XML vindo do BUS para WML ou HTML, dependendo do cliente Web em utilização.

O cliente SOAP foi conseguido utilizando WASP Java e, dado que é conhecida a localização do serviço a utilizar, não é necessário usar UDDI (*Universal Description, Discovery and Integration*). No Exemplo 1 apresenta-se o método `invokeDataRequestService` que invoca o Serviço Web.

```
public String invokeDataRequestService (String stringifiedEvent, String
                                        universalRequestName, String requestName) {
  String serviceURL= "http://seetde24.se.efacec.pt:8080/genesys/wsdl/DataRequest.wsdl";
  String returnVal;
  try {
    WebServiceLookup lookup= (WebServiceLookup)Context.getInstance(
          Context.WEBSERVICE_LOOKUP);
    DataRequestPortType client= (DataRequestPortType)lookup.lookup(serviceURL,
          DataRequestPortType.class);
    returnVal= client.dataRequest (stringifiedEvent, universalRequestName,
          requestName);
  } catch (WebServiceLookupException e){
    returnVal= e.toString();
  }
  return returnVal;
}
```

Exemplo 1: Cliente SOAP Proxy.java

Para ler o documento XML e analisar a sua estrutura foi usado o *parser* Xerces-J da Apache [Pro02b] e para processar o conteúdo do evento descrito em XML foi desenvolvida

a classe `DOMGenesysParse`. Esta classe utiliza a API DOM da especificação JAXP (*JAVA APIs for XML Processing*) e contém, por exemplo, um método que permite obter o valor de um atributo especificando o nome do evento e o nome do atributo:

```
public String getAttributeValue(String eventName, String attributeName)
```

A validação do documento XML é opcional e pode ser omitida por razões de eficiência, sempre que este tiver origem numa fonte de confiança. Para a validação dos documentos XML representando os eventos, são usados Esquemas XML (*XML Schema*) [Fal01] sendo a API DOM JAXP notificada dos erros. Para que isso aconteça é necessário ter configurada a classe `DocumentBuilderFactory`:

```
factory.setNamespaceAware(true); factory.setValidating(true)
```

No GENESys já existe o meta-modelo de eventos contendo, guardada numa base de dados, a modelização da estrutura de tópicos, eventos e "Data Requests". Para obter os correspondentes Esquemas XML foram desenvolvidos procedimentos em PL/SQL que os produzem por consulta à base de dados. Existem vários ficheiros Esquema XML: um contendo a informação de todos os tópicos existentes; outro contendo as definições dos tipos de dados existentes (os tipos de dados que são utilizados pelo BUS); finalmente, um ficheiro por cada tópico do sistema contendo a definição de eventos para esse tópico e os atributos que pertencem a cada evento.

Para produzir uma representação adequada a cada dispositivo de interacção, o evento representado em XML é transformado usando uma transformação XSLT [Cla99]. Na arquitectura proposta, o suporte de novos dispositivos reduz-se, praticamente, ao desenvolvimento de novos ficheiros XSL, para transformação dos conteúdos XML no formato de saída que esses dispositivos suportam. Numa primeira implementação, são suportados HTML e WML como formatos de saída.

Para a realização das transformações XSLT é usado o processador Xalan [Pro02a] da Apache. A API JAXP fornece um camada de adaptação aos processadores XSLT, à semelhança do que acontecia para os *parsers*. Esta abordagem, e tal como acontece para o processamento, permite a substituição de um motor XSLT por outro, caso seja necessário. A versão JAXP 1.1 inclui a API standard para motores XSLT, chamada TRaX (*Transformation API for XML*), que foi usada na implementação.

A implementação da transformação dividiu-se em duas tarefas: a implementação de uma classe Java para realizar a transformação e o desenvolvimento de várias folhas de estilo XSLT (*.xsl) com as especificações das transformações a aplicar. A classe Java implementada (`TransformerBean`) permite encapsular todo o comportamento da transformação. Esta classe disponibiliza o método `doTransformation`, que tem como argumentos a folha de estilo XSLT (`xslFile`) e uma string XML (`xmlStream`):

```
public String doTransformation(String xmlStream, String xslFile)
```

As folhas de estilo realizam a transformação de XML para um outro formato, mais adequado às características do dispositivo cliente. No Exemplo 2 apresenta-se parte do código da folha de estilo XSLT que efectua a transformação para WML.

6 Camada de Apresentação

A camada de apresentação realiza a interface com o utilizador. Recebe os pedidos dos utilizadores Web, analisa o tipo de pedido e os parâmetros definidos pelo utilizador e, utilizando

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="wml" indent="yes"/>
<xsl:template match="/">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="SafetyDocuments">
  <wml>
    <card id="GeneralSO" title="SOrder">
      <p><b>Order UID: </b>
      <xsl:value-of select="SwitchingOrderLine/OrderUid"/></p>
      <p><b>Created by: </b>
      <xsl:value-of select="SwitchingOrderLine/CreationUser"/></p>
      <do type="accept" label="Return">
        <go href="view_soid.wml"/>
      </do>
      <do type="accept" label="More">
        <go href="#SOLines"/>
      </do>
    </card>
    <card id="SOLines" title="SOLines"> <xsl:attribute name="title">SO:
      <xsl:value-of
...
  </wml>
</xsl:template>
</xsl:stylesheet
```

Exemplo 2: wml.xsl

a camada da lógica de negócio, constrói uma página com as características adaptadas ao tipo de dispositivo, para que o utilizador que fez o pedido consiga visualizar e interagir com a apresentação.

A adaptação de conteúdos pode ser efectuada no servidor, num intermediário (*proxy*) ou no navegador. A adaptação consiste na transformação do conteúdo que existe num formato, possivelmente mais propício à sua transmissão, num outro, adaptado às características do dispositivo que o vai consumir. Na adaptação no servidor ou por intermediação, as entidades que a realizam necessitam de saber as características do dispositivo. As características podem ser obtidas directamente, pela sua especificação, ou através dum identificador único do dispositivo cliente, utilizado para pesquisar a especificação num repositório.

Os servidores e *proxys* podem determinar a especificação de características particulares dum dispositivo utilizando o cabeçalho do pedido (*header request*) do protocolo HTTP. Existem outros mecanismos de especificação de características em desenvolvimento, que poderão ser utilizados alternativamente: o standard W3C *Composite Capability/Preferences Profile* (CC/PP), o standard *WAP User Agent Profile* (UAPROF), o standard *SyncML Device Information* (DevInf) e o standard *Universal Plug and Play* (UPnP).

A identificação das características do dispositivo não é realizada na implementação realizada devido à dificuldade em obter os parâmetros do cabeçalho HTTP pretendidos, quando o pedido é originado por certos navegadores (Internet Explorer 5, por exemplo). Por outro lado, as outras abordagens para identificação de características de dispositivos ainda estão em fase de desenvolvimento e também não foram consideradas. É utilizado um parâmetro do pedido que permite identificar, não só o tipo de cliente, mas também o contexto da aplicação que gerou o pedido (por exemplo, `wml-login` ou `html-login` para as interfaces de login em WML e HTML, respectivamente).

A implementação segue o padrão *Front Controller*, uma aplicação do padrão MVC (*Model View Controller*) [ABD01] usado frequentemente para a implementação de aplicações

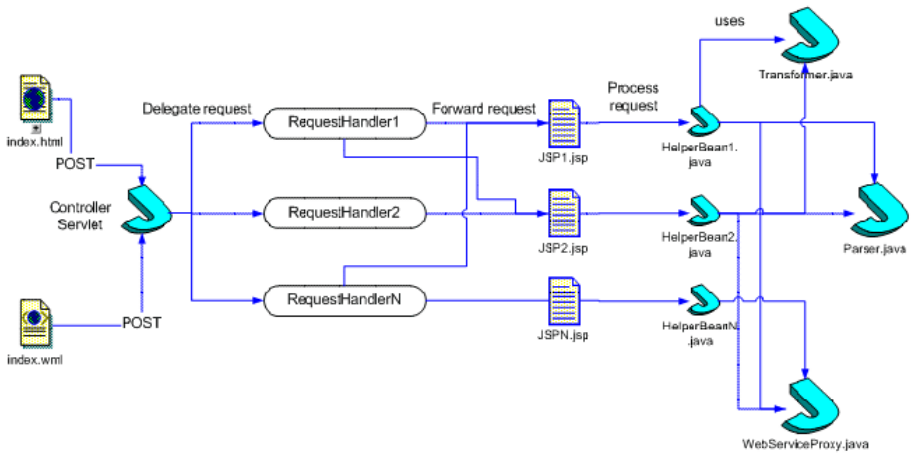

Figura 3: Camada de Apresentação

Web, dadas as suas vantagens em termos de escalabilidade e facilidade de manutenção. Tal como é ilustrado na Figura 3, é usado um *servlet de controlo*, um *request handler* por cada tipo de pedido, *Java Beans* que disponibilizam a informação do modelo para as vistas, e um conjunto de páginas JSP (*Java Server Pages*) que disponibilizam as vistas. O *servlet de controlo* trata todos os pedidos e delega o processamento nas classes que tratam cada tipo de pedido; cada *request handler* examina os parâmetros do pedido, actualiza o estado da aplicação, obtém a informação necessária, escolhe a vista JSP para a qual o controlador deve encaminhar a resposta e disponibiliza-lhe a informação necessária. As páginas JSP usam um `helperBean` para produzir uma representação adequada ao tipo de dispositivo; usam os componentes da lógica do negócio `WebServiceProxy` (para obter a informação necessária), `Parser` (para a processar) e `Transformer` (para a transformar).

7 Exploração e Validação da Arquitectura

Nesta secção é apresentada uma possível aplicação que utiliza a arquitectura apresentada nas secções anteriores. Esta aplicação foi pensada com o objectivo de validar e avaliar essa arquitectura.

Uma ordem de manobras (OM) é uma lista de operações a efectuar sobre a rede eléctrica, seja planeada ou para gerir um acidente. O operador que está no Centro de Comando é responsável pela evolução de uma ordem de manobra, gerindo-a com o auxílio de uma aplicação informática. Algumas instruções (marcadas com “I” — em “em instrução” — na Figura 4) são executadas no local, pelo piquete; este leva uma cópia da ordem de manobra, comunicando posteriormente ao operador a sua realização; o operador regista o facto (marcando a ordem com “C” — “confirmada”). Na Figura 2 apresentou-se um evento em XML, capturado numa execução desta aplicação, contendo informação sobre ordens de manobra.

A aplicação desenvolvida permite que um utilizador que faça parte de um piquete, possa ter acesso a informação sobre ordens de manobra usando um PDA com HTML ou um telemóvel com WAP. O utilizador pode actualizar a ordem de manobra no local, imediatamente após a execução da manobra. A aplicação possui as actividades ilustradas na Figura 5: identificação do utilizador, identificação da OM, visualização da OM, selecção de uma linha da OM e confirmação de uma linha de OM em instrução.

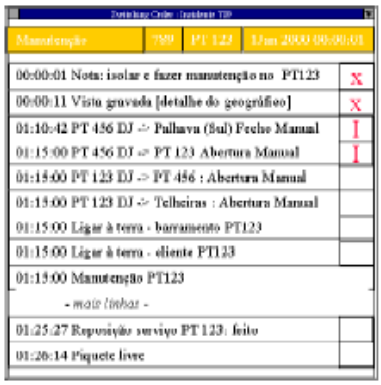

Figura 4: Ordem de manobras

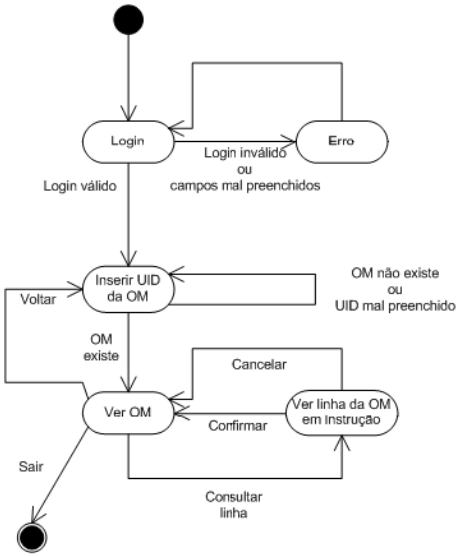

Figura 5: Actividades da Aplicação de OM

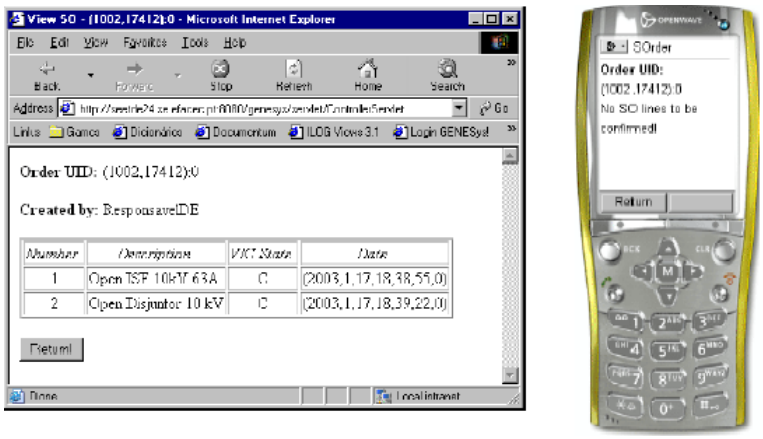

Figura 6: Exemplo de Interface HTML e WML

Na Figura 6 apresentam-se as interfaces HTML e WML para o caso em que não existem ordens de manobra em instrução. Nessa situação, para a interface HTML todas as linhas serão mostradas com o respectivo de estado de confirmado (“C”), não existindo possibilidade de selecção. Na interface WML é apresentada uma mensagem que indica a inexistência de linhas em estado de instrução (com possibilidade de serem confirmadas).

A aplicação desenvolvida prova que a interacção com o sistema GENESys foi conseguida, utilizando um navegador Web e um emulador WAP, respectivamente. A análise do desempenho não foi realizada, por este não ser um requisito identificado no âmbito deste trabalho.

8 Conclusões e Trabalho Futuro

O objectivo deste trabalho era propor e validar uma arquitectura que possibilitasse a interacção, através de dispositivos variados, com um sistema SCADA/DMS, cuja implementação se baseia numa arquitectura de componentes. A arquitectura deveria suportar independência da localização.

Analisando a aplicação desenvolvida e apresentada na Secção 7, podemos concluir que a interacção com o sistema GENESys foi demonstrada para dois tipos de dispositivos: clientes que utilizam navegadores Web e clientes que possuem telefones móveis que suportam WAP. A utilização da Internet é comum a ambos, garantindo a independência de localização. Utilizando a arquitectura proposta, a interacção com o sistema foi conseguida numa aplicação que envolve identificação de utilizadores, verificação de privilégios, visualização e interacção com o BUS envolvendo comunicação de eventos nos dois sentidos.

A utilização de Serviços Web permitiu a interoperabilidade entre sistemas heterogéneos, suportados em plataformas e linguagens de implementação diferentes, utilizando protocolos standard. A sua utilização permitiu a troca de informação entre o Web Mediator desenvolvido em C++ e as aplicações da “Zona Desmilitarizada”, desenvolvidas em Java.

A utilização da família de tecnologias XML garante um formato estruturado que permite que a informação seja auto-descrita. A utilização do XML na arquitectura facilitou a interoperabilidade e a utilização da tecnologia Java para a sua manipulação.

O suporte em Java da camada de apresentação e da camada de lógica de negócio, tecnologia amplamente utilizada na implementação de sistemas de apresentação sofisticados e na implementação de lógica de negócio em vários sistemas Web, facilitou a implementação pela compatibilidade evidente com XML; a portabilidade é outra grande vantagem da sua utilização na arquitectura desenvolvida.

A utilização de uma solução baseada no padrão MVC permite uma separação clara entre a camada de apresentação e a camada de lógica de negócio, permitindo que o desenvolvimento de aplicações seja dividido em diferentes fases e por diferentes técnicos (desenho de páginas Web, programação da lógica de negócio, integração de sistemas); a utilização desta solução trouxe vantagens, tais como a elevada reutilização e versatilidade, tempo de desenvolvimento curto e esforço de manutenção reduzido.

As possibilidades de trabalhos futuros são diversas, começando pela melhoria de desempenho e funcionalidades da implementação desenvolvida, uma vez que esta envolve uma panóplia de tecnologias que foi necessário colocar no lugar: Tomcat, Xerces-J, Xalan, AXIS, JAXP, SOAP, WASP, servlets, JSP, HTML e WML. Como neste trabalho apenas foram realizadas interfaces para HTML e WML, é possível a extensão a outros tipos de formatos de saída, como por exemplo o XHTML ou o GML.

Uma maior preocupação com a segurança, já que têm aparecido propostas na área dos Serviços Web neste sentido e a ligação desta arquitectura a outros sistemas legados para a além do GENESys, são outros desafios interessantes para a extensão deste trabalho.

Com a previsível evolução das capacidades dos dispositivos portáteis de uso pessoal, tornar-se-á possível o desenvolvimento de "clientes gordos". Será possível desenvolver aplicações utilizando maior capacidade de processamento do lado dos clientes e já existem pacotes de software que permitem desenvolver um cliente SOAP a ser utilizado em dispositivos móveis de pequeno porte. O suporte de SOAP pelos computadores de bordo de automóveis seria o ideal para o desenvolvimento de novas aplicações, por exemplo para serem utilizadas por equipas de piquete.

De salientar que este trabalho tem suporte em aplicações industriais e o seu resultado beneficia essas aplicações, para além do interesse de investigação envolvendo a aplicação deste tipo de novas tecnologias.

Agradecimentos: Este trabalho foi suportado pela "EFACEC Sistemas de Electrónica S.A.".

Referências

[ABD01] Subrahmanyam Allamaraju, Cedric Buest, and John *et al.* Davies. *Professional Java Server Programming J2EE 1.3 Edition*. Wrox Press Ltd., Setembro 2001.

[Boy99] Stuart A. Boyer. *SCADA: Supervisory Control and Data Acquisition*. ISA — The Instrumentation, Systems and Automation Society, 2nd edition, Janeiro 1999.

[BPM00] T. Bray, C.M. Paoli, J. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition). Technical report, W3C, Outubro 2000. `http://www.w3.org/TR/2000/REC-xml-20001006`.

[Cla99] James Clark. XSL Transformations (XSLT) Version 1.0, W3C Recommendation. Technical report, W3C, Novembro 1999. `http://www.w3.org/TR/1999/REC-xslt-19991116`.

[Fal01] David C. Fallside. XML Schema Part 0: Primer, W3C Recommendation. Technical report, W3C, Maio 2001. `http://www.w3.org/TR/xmlschema-0/`.

[Fer03] Ricardo Jorge Fernandes. Interfaces Web para Aplicações SCADA. Master's thesis, Departamento de Ciência de Computadores, Faculdade de Ciências da Universidade do Porto, Março 2003.

[Gom03] Miguel Pereira Gomes. Canal SCADA na Web. Master's thesis, Departamento de Ciência de Computadores, Faculdade de Ciências da Universidade do Porto, Fevereiro 2003.

[HPN00] Richard Han, Veronique Perret, and Mahmoud Naghshineh. Webslitter: A unified XML framework for multi-device collaborative web browsing mobility. In *ACM CSCW'00 Conference on Computer-Supported Cooperative Work*, pages 221–230, 2000.

[Mar00] Dave Marsh. O formato stringuificado. Technical report, Documento interno da EFACEC S.E., Março 2000.

[MV01] Dave Marsh and Paulo Viegas. The BUS architecture. In *2001 IEEE Porto PowerTech Proceedings, Porto, Portugal*, Setembro 2001.

[Per03] Carlos Ângelo Paiva Pereira. Interacção com uma arquitectura de componentes com independência de dispositivo e localização. Master's thesis, Departamento de Engenharia Electrotécnica e de Computadores, Faculdade de Engenharia da Universidade do Porto, Abril 2003.

[Pro02a] Apache Xalan Project. Xalan, Dezembro 2002. `http://xml.apache.org/xalan-j/`.

[Pro02b] Apache XML Project. Xerces-J, Dezembro 2002. `http://xml.apache.org/xerces-j/`.

[STK00] James Snell, Doug Tidwell, and Pavel Kulchenko. *Programming Web Services with SOAP*. O'Reilley & Associates, Inc., 2000.

[Sys02] Systinet. WASP Server for C^{++}, Dezembro 2002. `http://www.systinet.com/products/wasp_cserver/overview`.

[TT02] Lukas Tan and Ken Taylor. Mobile SCADA with thin clients — a web demonstration. In *International Conference on Information Technology & Applications (ICITA 2002), Bathurst, Australia (25-28 Novembro de 2002)*, 2002.