UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Pablo Valério Polônia

PROPOSTA DE ARQUITETURA ORIENTADA A RECURSOS PARA SCADA NA WEB

Florianópolis

2011

Pablo Valério Polônia

PROPOSTA DE ARQUITETURA ORIENTADA A RECURSOS PARA SCADA NA WEB

Dissertação submetida à Universidade Federal de Santa Catarina para a obtenção do Grau de Mestre em Engenharia de Automação e Sistemas
Orientador: Max Hering de Queiroz, Dr.

Florianópolis

2011

Catalogação na fonte elaborada pela biblioteca da
Universidade Federal de Santa Catarina

P778p Polônia, Pablo Valério Proposta de arquitetura orientada a recursos para SCADA na Web [dissertação] / Pablo Valério Polônia ; orientador, Max Hering de Queiroz. - Florianópolis, SC, 2011. 145 p.: il., tabs.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Engenharia de Automação e Sistemas.

Inclui referências. 1. Engenharia de sistemas. 2. Sistemas operacionais distribuídos (Computadores). 3. Arquitetura de computador.

4. Serviços da Web. 5. Engenharia de software. I. Queiroz, Max Hering de. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia de Automação e Sistemas. III. Título.

CDU 621.3-231.2(021)

Pablo Valério Polônia

PROPOSTA DE ARQUITETURA ORIENTADA A RECURSOS PARA SCADA NA WEB

Esta Dissertação foi julgada adequada para a obtenção do Título de Mestre em Engenharia de Automação e Sistemas, Área de Concentração em *Controle Automação e Sistemas* e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina.

Florianópolis, 31 de Agosto 2011.

José Eduardo Ribeiro Cury, Dr.
Coordenador do Programa de Pós-Graduação
em Engenharia de Automação e Sistemas

Max Hering de Queiroz, Dr.
Orientador

Banca Examinadora:

Joni da Silva Fraga, Dr.

Ricardo José Rabelo, Dr.

Rosvelter Coelho da Costa, Dr.

Aos meus pais.

AGRADECIMENTOS

Primeiramente gostaria de agradecer aos meus pais, Divair e Emerson, por todo apoio que deram ao longo da minha formação. Agradeço todo o carinho e o esforço que fizeram para que eu pudesse ter acesso a uma boa educação em um país onde infelizmente este ainda é um privilégio de poucos.

Sou muito grato aos meus amigos do Edugraf, em especial ao Prof. Melgarejo pelas orientações e aos colegas Diego, Giovani, Lucas e Ricardo por todas as críticas, sugestões e também pela paciência que tiveram em assistir todas as minhas apresentações. Fico feliz em poder aprender com vocês todos os dias.

Agradeço ao Prof. Max pela orientação do trabalho e por todas as críticas realizadas. Também gostaria de agradecer a minha companheira Aline pelo amor e compreensão, e a todos os meus amigos que me ajudaram nesta empreitada. Por último, agradeço a colaboração dos demais professores, colegas e servidores da UFSC.

A simplicidade é uma grande virtude, mas requer trabalho duro para alcançá-la e educação para apreciá-la. E para piorar as coisas: a complexidade vende melhor.

Edsger Wybe Dijkstra

Resumo

Esta dissertação descreve uma proposta de arquitetura de software para aplicações típicas de Sistemas de Supervisão e Aquisição de Dados (SCADA), utilizando a World Wide Web como plataforma. O objetivo é mostrar como os requisitos característicos de aplicações SCADA podem ser incorporados em uma arquitetura condizente com os princípios arquiteturais que fundamentam a Web, dado que as arquiteturas comumente propostas para SCADA, baseadas em Chamadas Remota de Procedimento (RPC, na sigla em inglês), apresentam problemas de forte acoplamento, manutenção de estado e interfaces especializadas, que dificultam uma integração plena com a Web. Para isto é projetada uma Arquitetura Orientada a Recursos (ROA, na sigla em inglês), que utiliza as tecnologias da Web (HTTP, URI e tipos de mídia) de acordo com seus princípios de arquitetura. A arquitetura é projetada utilizando como cenário uma Célula Flexível de Manufatura (CFM), ambiente característico de um SCADA. As funcionalidades típicas de SCADA são projetadas como recursos e expostas para clientes que podem exibir sinóticos em uma IHM, esperar pelo disparo de alarmes, controlar o processo, configurar dispositivos e abastecer com dados sistemas de MES/ERP. Uma implementação é realizada, para demonstrar como se dá a interação entre aplicações na arquitetura. Nesta implementação um aplicativo SCADA (Mango M2M) teve sua arquitetura de software estudada e modificada para se adaptar as necessidades da arquitetura proposta. Como resultado, obtém-se uma arquitetura que cobre os requisitos típicos de aplicações SCADA, integrando-se à Web de forma condizente com seus princípios arquiteturais. Posteriormente a arquitetura projetada é comparada com uma arquitetura baseada em Web Services RPC e as diferenças em termos de integração com a Web e no cumprimento dos requisitos típicos de SCADA são analisadas.

Palavras-chave: SCADA. Arquitetura de Software. Web. Serviços Web. REST. ROA.

Abstract

This dissertation describes a proposed software architecture for typical Supervisory Control and Data Acquisition (SCADA) systems, using the World Wide Web as a platform. The goal is to show how the characteristic requirements of SCADA can be incorporated into an architecture consistent with the architectural principles that underlie the Web, since the commonly proposed architectures for SCADA, based on Remote Procedure Call (RPC) have problems with strong coupling, maintenance of state and interface specialization that hinder full integration with the Web. For accomplishng this goal is designed a Resource Oriented Architecture (ROA), which uses Web technologies (HTTP, URI, and media types) according to its architectural principles. The architecture is designed from a characteristic SCADA environment, a Flexible Manufacturing Cell (FMC). Typical SCADA features are designed as resources for clients that can display synoptics on HMI, wait for an alarm to fire, control processes, configure devices and supply data for MES/ERP systems. An implementation is carried out to demonstrate how the interaction takes place between applications in the proposed architecture. In this implementation a SCADA application (Mango M2M) had its software architecture studied and modified to suit the needs of the architecture. As a result, the proposed architecture covers the typical requirements for SCADA applications, integrating the Web in a manner consistent with their architectural principles. Later the designed architecture is compared with an Web Services architecture based on RPC and the differences in terms of integration with the Web and in compliance with the requirements of typical SCADA are analyzed.
Keywords: SCADA. Software Architecture. Web. Web Services. REST. ROA.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

1.1 Justificativa

Sistemas de Supervisão e Aquisição de Dados (SCADA) são sistemas computacionais utilizados na coleta de informações, monitoramento e controle (BAILEY; WRIGHT, 2003). Esses sistemas possibilitam que operadores supervisionem processos que muitas vezes estão localizados em regiões longínquas ou de difícil acesso, centralizando as informações coletadas em interfaces que refletem o que está acontecendo em campo. São amplamente utilizados na indústria e em serviços de utilidade pública, em áreas como manufatura, metalurgia, energia, água e esgoto, gás e petroquímica (DANEELS; SALTER, 1999) (MORAES, 2005).

A expansão da Internet e a evolução dos sistemas gerenciais e de planejamento, como os Sistemas de Execução da Manufatura (MES, na sigla em inglês) e os Sistemas Integrados de Gestão Empresarial (ERP, na sigla em inglês) têm trazido novas oportunidades para as indústrias. Esse cenário tem motivado uma integração entre os sistemas utilizados nos vários níveis de uma dada organização, na qual as informações operacionais colhidas por um SCADA tem um peso importante na tomada de decisões estratégicas (FAVARETTO, 2001) (HOWELLS, 2000). A abertura dos mercados, atuação de agências reguladoras e a desregulamentação de setores, tais como o da energia elétrica, têm propiciado um interação entre organizações. Esses fatores trazem novos desafios, pois a troca de informações passou a se dar em um nível que extrapola barreiras geográficas e organizacionais (QIU; GOOI, 2000)(KHATIB et al., 2000).

Em paralelo, a World Wide Web, um sistema distribuído de alcance mundial, tem revolucionado a forma como pessoas, organizações e sistemas geram, acessam, e compartilham informações (BERNERS-LEE, 2000). Esta rede tem se expandido rapidamente desde seu surgimento, agregando informações de sistemas heterogêneos sem a necessidade de qualquer tipo de controle centralizado e com o cumprimento de requisitos de confiabilidade e segurança (FIELDING; TAYLOR, 2002). Uma melhor compreensão dos conceitos que nortearam a Web tem fomentado o interesse da indústria e da academia nos últimos anos, levando profissionais e pesquisadores a explorarem as potencialidades da Web como uma plataforma para o desenvolvimento de aplicações distribuídas (RICHARDSON; RUBY, 2007).

Observa-se que os sistemas SCADA têm feito uma transição na direção de possibilitar o seu uso como aplicativos Web (FAN; CHEDED; TO-

KER, 2005) (LI; SERIZAWA; KIUCHI, 2002) (QIU; GOOI, 2000) (KHATIB et al., 2000). Atualmente existem sistemas SCADA através dos quais operadores podem configurar sensores, enviar ações de controle, gerenciar alarmes, visualizar sinóticos, etc. usando tão somente navegadores típicos da Web(SEROTONIN, 2011). No entanto ainda é muito difícil encontrar sistemas SCADA que participem de forma plena da Web, não tendo sido projetados de acordo com seus princípios de arquitetura de software. Um SCADA construído com base nestes princípios poderia aproveitar-se da interoperabilidade garantida pelas tecnologias abertas e padronizadas que sustentam a Web, favorecendo a interação com outras aplicações. Além disso poderia se beneficiar de agentes intermediários favorecendo aspectos relacionados a segurança, escalonabilidade e performance (FIELDING, 2000).

Pode-se citar alguns trabalhos relacionados na literatura. As necessidades de dispor dados coletados por um SCADA para diferentes setores de uma mesma organização são exploradas em (ZECEVIC, 1998). Como solução os autores propõem a conexão das redes de comunicação de SCADA com a rede interna (Intranet) da organização, combinada ao uso de navegadores Web como interface padrão para o acesso dessas informações. Também são apontados os benefícios do uso de tecnologias abertas e padronizadas na redução dos custos e da dependência de fabricantes de equipamentos.

Em (MEDIDA; SREEKUMAR; PRASAD, 1998), (QIU; GOOI, 2000) e (KHATIB et al., 2000) são exploradas a vantagens de expor um SCADA na Internet, no sentido de disponibilizar acesso remoto e global das informações coletadas pelos dispositivos de campo, que posteriormente podem ser apresentadas através de navegadores na Web.

Uma análise do uso combinado das tecnologias Java e XML (Extensible Markup Language, na sigla em inglês) para favorecer requisitos de portabilidade, performance e interoperabilidade de sistemas SCADA na Web é feita em (FAN; CHEDED; TOKER, 2005). O trabalho traz aspectos conceituais importantes, categorizando as aplicações de automação industrial na Web como “habilitadas para Web” e as “integradas à Web”. O termo “habilitadas” para Web” se refere as aplicações que possuem uma interface gráfica que pode ser acessada por um navegador, como as citadas nos trabalhos descritos anteriormente. Já o termo “integradas a Web” é utilizado para aplicações distribuídas, compostas por diferentes subsistemas integrados por meio de tecnologias Web. Outra contribuição importante é o projeto de uma arquitetura de software para uma aplicação SCADA dita integrada a Web, baseada em Web Services RPC.

1.2 Objetivos

Este trabalho tem como objetivo propor a integração de aplicações típicas de SCADA com a Web através de uma Arquitetura Orientada a Recursos (ROA, na sigla em inglês), buscando mostrar como requisitos característicos destes sistemas podem ser incorporadas por esta arquitetura.

Com base neste objetivo geral, pode-se citar os seguintes objetivos específicos:

- Analisar as arquiteturas de software comumente propostas para SCADA;
- Realizar uma implementação da arquitetura através da modificação de um aplicativo para SCADA;
- Desenvolver aplicações que se integrem com a arquitetura projetada, mostrando algumas de suas propriedades;
- Comparar a arquitetura projetada com uma arquitetura existente e representativa das arquiteturas de software comumente propostas para SCADA;

1.3 Metodologia

A metodologia utilizada durante a pesquisa compreendeu os seguintes passos:

- Revisão bibliográfica acerca de SCADA, focando nos requisitos de software típicos destas aplicações;
- Revisão bibliográfica sobre os princípios de arquitetura de software que fundamentam a Web e arquiteturas condizentes com estes princípios;
- Análise das arquiteturas de software comumente propostas para SCADA e as dificuldades que apresentam para realizar uma integração com a Web;
- Projeto e implementação de uma arquitetura para aplicações típicas de SCADA condizente com os princípios arquiteturais da Web;
- Comparação da arquitetura projetada com uma existente buscando mostrar as diferenças no cumprimento dos requisitos típicos de SCADA e em sua integração com a Web;
- Avaliação dos resultados;

1.4 Organização do Documento

Este documento está organizado da seguinte forma: no Capítulo 2 é apresentada uma visão geral de SCADA, abrangendo seus conceitos, tecnologias e requisitos típicos de software, levando em conta alguns aspectos históricos que demonstram a evolução pela qual esta área tem passado.

No Capítulo 3 são apresentados conceitos da área de arquitetura de software, em sequência é feita uma introdução sobre a Web e seus princípios de arquitetura, incorporados no estilo arquitetural REST. O capítulo é concluído com a descrição de uma metodologia para o projeto de Arquiteturas Orientadas a Recursos, que incorporam as tecnologias da Web de forma condizente com seus príncipios arquiteturais.

O Capítulo 4 expõe uma análise sobre arquiteturas de software baseadas em RPC, onde são explorados os pontos em que estas arquiteturas, predominantes em SCADA, divergem dos princípios de arquitetura da Web dificultando sua integração com a mesma.

No Capítulo 5 é apresentado um estudo de caso envolvendo um ambiente típico de SCADA, uma CFM. Com base nessa CFM é projetada uma ROA que abrange os requisitos típicos de uma aplicação SCADA. Em seguida são apresentadas algumas aplicações, descrita a implementação e analisados os resultados.

Em seguida no Capítulo 6 é realizado um estudo comparativo entre a arquitetura projetada e uma arquitetura baseada em Web Services RPC. As duas arquiteturas são utilizadas para modelar uma aplicação SCADA na Web e as diferenças em termos de integração com a Web e no cumprimento dos requisitos típicos de SCADA são analisadas. Finalmente, apresentam-se as conclusões da pesquisa no Capítulo 7, bem como sugestões para trabalhos futuros.

2 SCADA

2.1 Contextualização

Aplicativos SCADA são utilizados na supervisão e controle de processos (BAILEY; WRIGHT, 2003). São amplamente empregados na indústria e em serviços de utilidade pública, em áreas como manufatura, metalurgia, energia, água e esgoto, gás e petroquímica (DANEELS; SALTER, 1999) (MORAES, 2005). Podem variar em tamanho e complexidade, podendo ser utilizados no monitoramento de processos que vão desde as condições ambientais de uma sala, ou de todos os equipamentos de uma central de distribuição de energia elétrica (National Communications Systems, 2004).

Esses sistemas utilizam-se de telemetria, realizando a coleta de dados dos dispositivos de campo, muitas vezes geograficamente dispersos e as enviando para centrais de controle, onde são processadas e armazenadas. Nestas centrais os operadores podem acompanhar o processo, visualizando informações por meio de sinóticos, tabelas e gráficos que refletem o que está acontecendo em campo. Por meio desta interface os operadores também podem eventualmente realizar alguma ação de controle, como a abertura de uma válvula ou a configuração de um dispositivo de campo (BAILEY; WRIGHT, 2003).

No contexto de outras aplicações de automação industrial os sistemas SCADA possuem um papel importante, que é o de fornecer os dados de chão de fábrica para outras aplicações, além de receberem comandos de controle das mesmas. Neste sentido, um SCADA tem um papel operacional, enquanto que aplicações como MES ou ERP têm um escopo mais estratégico. Aplicações MES/EPS tem como finalidade o planejamento e o controle da produção, permitindo a gerência de lotes de produção, alocação de recursos, controle de qualidade, sequenciamento de operações, e permitindo a contabilização e o rastreamento de equipamentos e matéria prima (FAVARETTO, 2001). Os Sistemas ERP envolvem decisões estratégicas em um nível mais alto, pois integram todos os setores de organização, sejam eles operacionais, produtivos, administrativos ou comerciais (MARDEGAN; AZEVEDO; OLIVEIRA, 2002). Essa divisão hierárquica de tarefas entre os sistemas utilizados em uma organização é muito bem capturada pela pirâmide da automação, que pode ser vista na figura 1, adaptada de (HARJUNKOSKI; NYSTROM; HORCH, 2009). No nível mais baixo estão os sistemas que e relacionam com equipamentos de campo, como os sistemas de controle distribuído e SCADA, enquanto que nos níveis mais altos estão as aplicações

MES e ERP.

Para cumprir com suas funcionalidades um sistema SCADA precisa de um arranjo muitas vezes complexo de hardware, software e tecnologias de comunicação (National Communications Systems, 2004), descrito a seguir.

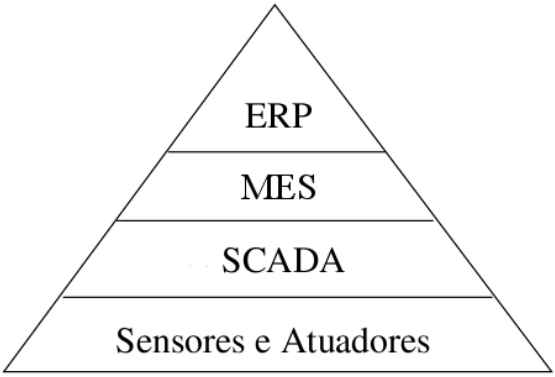

Figura 1: Pirâmide da automação

2.2 Componentes de Hardware

Na literatura são utilizados alguns termos recorrentes para designar os componentes de hardware que fazem parte desses sistemas, de acordo com o papel que eles desempenham no fluxo das informações coletadas do processo. Esses componentes podem ser categorizados em dispositivos de campo, estações remotas, estações mestre e estações de operação (KRUTZ, 2005).

2.2.1 Dispositivos de Campo

Os dispositivos de campo podem ser divididos em sensores e atuadores. Os sensores medem grandezas físicas associadas ao processo, como a temperatura de uma caldeira, a pressão da água em uma represa ou a tensão em uma linha de transmissão de energia. Existe uma vasta gama de dispositivos com esse propósito que abarcam sensores de pressão, calor, proximidade, movimento, entre outros (National Communications Systems, 2004). Já os

atuadores são responsáveis pelo controle do processo, por exemplo através da abertura ou fechamento de válvulas, acionamento de motores ou fechamento de um contato elétrico. O tipo de sinal que os dispositivos de campo trabalham pode variar de sinais analógicos (contínuos) para sinais digitais (discretos).

Pode se notar uma certa tendência da indústria em tornar esses instrumentos de campo mais inteligentes, dotando-os de microprocessadores que permitem o tratamento das informações, auto-calibragem e mesmo a interação com outros dispositivos através de protocolos de comunicação (POPA; POPA; PATITOIU, 2007).

2.2.2 Estações Remotas

Estações remotas, ou Remote Terminal Units (RTU, na sigla em inglês) são dispositivos eletrônicos programáveis cujo objetivo é realizar a comunicação entre dispositivos de campo e estações mestre, coletando dados dos sensores e repassando ações de controle para os atuadores (WARD et al., 2004). O nome estação remota está relacionado ao fato de que geralmente esses dispositivos estão em lugares remotos, distantes da central de controle onde se encontram os operadores (BAILEY; WRIGHT, 2003). RTUs são dotadas de microprocessadores e memória, podendo ser programados para executar localmente algoritmos de controle (um laço de controle realimentado por exemplo).

Muitas vezes Controladores Lógico Programáveis (CLPs) são utilizados com o mesmo propósito dos RTU. CLPs são utilizados para controlar diversos tipos de máquinas e processos, tendo sua programação derivada da lógica dos diagramas elétricos a relés, executando através de uma rotina cíclica de operações (SOUZA, 2005). De fato existe alguma confusão sobre a distinção entre CLPs e RTUs. Os primeiros surgiram em um contexto fabril da automação, possuindo capacidade de programação e um menor necessidade de alcance na comunicação, devido as distâncias menores encontradas entre os dispositivos no chão de fábrica. Os RTUs são oriundos dos primeiros sistemas de telemetria, portanto a comunicação a longas distâncias era um requisito crucial que superava a necessidade de esses dispositivos serem programáveis. Contudo, pode se observar que ambos foram evoluindo, suprindo suas deficiências, fato que torna esta distinção hoje um tanto nebulosa (National Communications Systems, 2004). Portanto daqui em diante quando for utilizado o termo estação remota, este também estará abrangendo os CLPs.

A interface das estações remotas com os dispositivos de campo geralmente é física, para tanto esses equipamentos são dotados de módulos de

entrada e saída analógicos e digitais. Já a interação com as estações mestre se dá através de pilhas de protocolos de comunicação comuns nos meios industrias, que serão detalhadas posteriormente.

2.2.3 Estações Mestre

As estações mestre são computadores pessoais ou industriais, estes últimos possuindo maior robustez para resistir as condições adversas encontradas no chão de fábrica. Suas responsabilidades são obter, processar e armazenar os dados oriundos das estações remotas assim como repassar comandos de controle para elas, o que geralmente é feito através de protocolos industriais. Por possuírem um maior poder computacional as estações mestre acabam também por concentrar grande parte das funcionalidades de software de um SCADA (National Communications Systems, 2004). Um SCADA pode ter apenas uma estação mestre ou então várias, em uma arquitetura distribuída de máquinas conectadas em rede (WARD et al., 2004).

2.2.4 Estações de Operação

As interfaces homem-máquina (IHM), ou estações de operação são computadores pessoais ou industriais que tornam possível a interação de operadores humanos com os processos subjacentes. A estação de operação executa o software que apresenta uma interface gráfica pela qual o operador pode supervisionar e atuar sobre o processo. Em alguns casos a estação de operação e estação mestre são a mesma máquina, contudo é comum que tais funcionalidades estejam separadas em máquinas distintas (FAN; CHEDED; TOKER, 2004).

Como pode ser visto na descrição dos componentes de hardware de um SCADA, o fluxo das informações coletadas do processo segue um caminho hierárquico, fluindo dos sensores no nível mais baixo e próximo do processo supervisionadopara para as estações remota, mestre e de operação respectivamente. O sentido do fluxo é invertido quando se trata de alguma ação de controle efetuada pelo operador, a figura 2 ilustra esta organização hierárquica dos componentes de um SCADA.

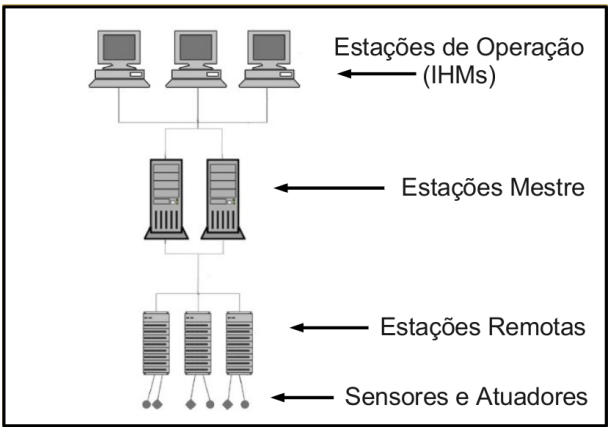

Figura 2: Hierárquia típica dos componentes de hardware em um SCADA

2.3 Tecnologias de Comunicação

As tecnologias de comunicação são as responsáveis pela troca de informações entre os componentes de um sistema SCADA. Essas tecnologias sempre tiveram uma atribuição importante, principalmente devido a natureza distribuída de um SCADA, com seus dispositivos dispersos em campo (National Communications Systems, 2004). Para a comunicação das estações remotas e mestre tipicamente são utilizados protocolos de redes industriais como o Modbus, DNP3, Fieldbus e Profibus ou então algum protocolo proprietário. Estes protocolos são empregados devido a necessidades intrínsecas das aplicações do meio industrial, tais como controle de tempo real. Em um nível mais alto, as estações mestre e de operações se conectam através de redes locais, utilizando Ethernet ou Token Ring. O meio utilizado para esta comunicação é algo que varia com o ambiente monitorado. Cabeamento é muito utilizado em fábricas onde as distâncias entre os equipamentos é geralmente pequena, entretanto quando existem pontos supervisionados que cobrem uma grande área a comunicação via rádio, telefonia ou mesmo satélite é preferida. Nos casos de comunicação via rádio frequência, é muito comum as estações remotas se comunicarem entre si, atuando como repetidoras de sinal (KHATHAIR, 2006).

2.3.1 Histórico

Os sistemas SCADA tiveram sua história influenciada pelos avanços da tecnologia da informação, principalmente no que diz respeito as tecnologias de comunicação. Nos últimos anos estas mudanças tem sido aceleradas em virtude da penetração das tecnologias de informação neste domínio (FAN; CHEDED; TOKER, 2004). Conforme (National Communications Systems, 2004), pode-se dividir a evolução de SCADA com relação a suas tecnologias de comunicação em três gerações, que são descritas a seguir.

2.3.1.1 Primeira Geração, Monolítica

Os primeiros SCADA surgiram na década de 60, quando a cena da computação era dominada pelos Mainframes (FAN; CHEDED; TOKER, 2004). As redes de computadores praticamente não existiam, portanto esses sistemas eram centralizados em grandes máquinas com limitada capacidade de processamento e praticamente nenhuma conectividade. A exceção era a comunicação física com os dispositivos de campo, e posteriormente com as estações remotas criadas com a evolução do hardware. Os protocolos utilizados eram proprietários e extremamente simples, tendo como objetivo exclusivo a coleta de informações em campo, muitas vezes envolvendo comunicação a longas distâncias via rádio. A figura 3 adaptada de (National Communications Systems, 2004) ilustra uma arquitetura de comunicação tipíca de um SCADA da primeira geração.

2.3.1.2 Segunda Geração, Distribuída

A próxima geração foi influenciada pelos avanços da miniaturização e das redes locais de computadores. Desta forma a arquitetura que antes era centralizada passou a ser distribuída, possibilitando a existência de estações mestre e de operação conectadas em rede. Diversas vantagens foram trazidas devido a esse avanço, permitindo maior confiabilidade devido a possibilidade de redundância e uma maior escalonabilidade devido a capacidade de distribuir funcionalidades e carga em máquinas distintas (ZECEVIC, 1998).

Contudo os protocolos de comunicação continuaram majoritariamente proprietários, isto acontecia porque a maioria dos fabricantes fornecia um pacote de soluções que englobava componentes de hardware, software e comunicações, pouco interessando para esses a interoperabilidade com outros fabricantes. A figura 4 adaptada de (National Communications Systems,

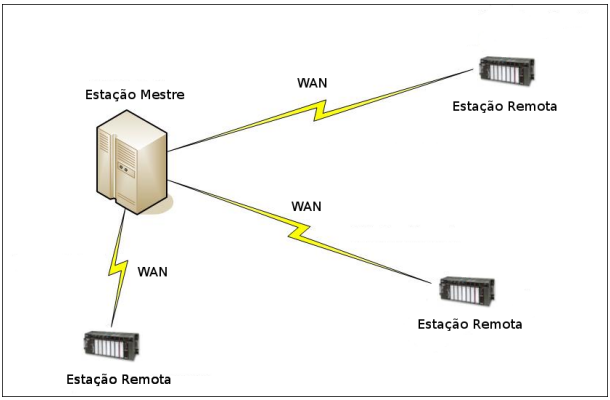

Figura 3: Arquitetura de comunicação típica de um SCADA da Primeira Geração

2004) mostra uma arquitetura de comunicação típica de um SCADA da segunda geração, exibindo a comunicação entre estações mestre e de operação em rede local e entre estações remotas e estação mestre via rede de longa distância.

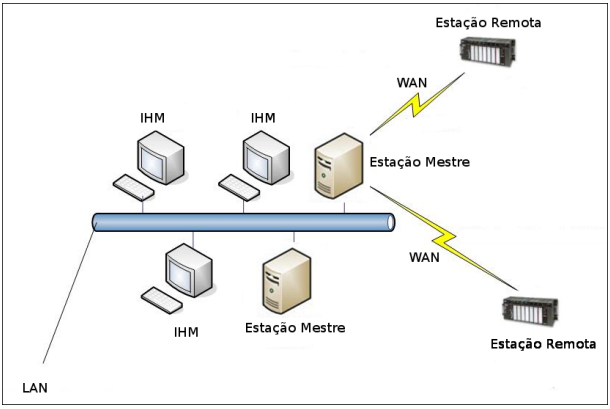

Figura 4: Arquitetura de comunicação típica de um SCADA da Segunda Geração

2.3.1.3 Terceira Geração, Em Rede

A grande diferença da geração atual para a anterior é que esses sistemas estão se tornando mais abertos. Os protocolos proprietários estão sendo gradualmente substituídos por protocolos abertos e padronizados, favorecendo a interoperabilidade entre equipamentos de fabricantes distintos (MEDIDA; SREEKUMAR; PRASAD, 1998). Tal fato tem criado uma separação maior entre os nichos de mercado, permitindo que as empresas que desenvolvem o software por exemplo possam se focar somente neste aspecto, sem se preocupar com questões de hardware ou sistema operacional (National Communications Systems, 2004).

Outro aspecto desta abertura está na transição das redes locais para redes de longa distância na comunicação entre as estações mestre, motivada pela necessidade de integração entre sistemas geograficamente dispersos (MEDIDA; SREEKUMAR; PRASAD, 1998) (QIU; GOOI, 2000) (KHATIB et al., 2000). Neste âmbito a pilha TCP/IP da Internet tem sido primordial para realizar esta integração. Tais mudanças também tem penetrado gradualmente nas estações remotas na medida em que as plataformas de hardware e software utilizadas por elas tem evoluído, possibilitando o uso dos protocolos da Internet nestes dispositivos (LI; SERIZAWA; KIUCHI, 2002). Na figura 5 adaptada de (National Communications Systems, 2004) é ilustrada uma arquitetura de comunicação típica de um SCADA em rede, cujo ponto de convergência de interação entre os equipamentos é a Internet.

2.3.1.4 SCADA na Web

Pode-se observar que os sistemas SCADA em rede têm feito uma transição na direção de possibilitar o seu uso como aplicativos na Web (FAN; CHEDED; TOKER, 2005) (LI; SERIZAWA; KIUCHI, 2002) (QIU; GOOI, 2000) (KHATIB et al., 2000). Esta transição tem sido motivada pela melhora de interoperabilidade que as tecnologias abertas e padronizadas da Web trazem para a área (FAN; CHEDED; TOKER, 2005).

Uma das vantagens de utilizar tecnologias da Web em SCADA esta na padronização das interfaces gráficas através de navegadores, substituindo tecnologias proprietárias e dependentes de plataforma. Neste aspecto um avanço recente é a possibilidade de os navegadores serem atualizados através de solicitações assíncronas de informação, sem que seja necessária uma interação do usuário para isto (CHAU; KHAI, 2007). Combinando atualizações assíncronas, gráficos vetoriais (FENG-PING; CHUN-HUA; JIAN, 2008) e código móvel em Java (FAN; CHEDED; TOKER, 2004) e Java Script (WALLACE,

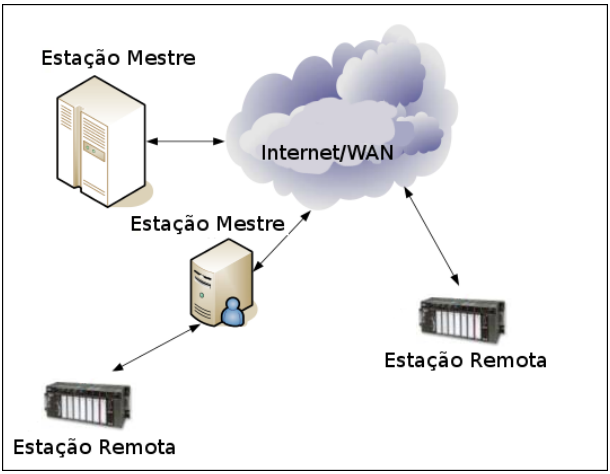

Figura 5: Arquitetura de comunicação típica de um SCADA da Terceira Geração

2004) é possível o desenvolvimento de interfaces gráficas dinâmicas o suficiente para acompanhar processos típicos da indústria.

Inicialmente o uso das tecnologias da Web em SCADA se limitava a apresentação de interfaces gráficas, estando estas aplicações apenas habilitadas para Web. Contudo, com o surgimento da linguagem de marcação XML e do conceito de serviços na Web novas oportunidades se apresentam, no sentido de realizar uma integração entre aplicações através destas tecnologias, possibilitando o surgimento de sistemas SCADA não só habilitados para a Web, mas também integrados a ela (FAN; CHEDED; TOKER, 2005).

2.4 Requisitos de Software

Existe uma diversidade de aplicativos SCADA no mercado, cada um deles possui particularidades em relação as funcionalidades que oferece. Contudo, apesar destas especificidades, é possível obter um conjunto em comum de requisitos e funcionalidades para esse tipo de sistema (CASIMIRO, 1995).

Na engenharia de software os requisitos de um sistema podem ser divididos em funcionais e não funcionais (GHEZZI; JAZAYERI; MANDRIOLI, 2002). Os requisitos funcionais ditam o que o sistema deve fazer, ou seja quais funcionalidades deve prover. Enquanto isto, os requisitos não fun-

cionais dizem respeito a propriedades de qualidade do sistema, tais como segurança e performance (BASS; CLEMENTS; KAZMAN, 2003). Esta divisão será adotada para descrever o escopo de cada requisito de SCADA. Primeiramente serão abordados os requisitos funcionais e em seguida os não funcionais.

2.4.1 Requisitos Funcionais

2.4.1.1 Aquisição de dados

A aquisição de dados é o procedimento de coleta de informações sobre o processo. Estas informações geralmente estão associadas a grandezas físicas, como a temperatura capturada por um sensor ou a tensão elétrica em uma linha de transmissão. Outro tipo de informação capturada é o estado de equipamentos de campo, por exemplo se uma válvula está aberta ou fechada ou se um instrumento está com defeito (GOMES, 2003).

O procedimento de aquisição se dá entre as estações mestre e as estações remotas, que capturam dados através dos sensores. A forma como acontece a comunicação entre esses componentes depende dos protocolos utilizados, sendo tipicamente empregados protocolos industriais que possuem uma dinâmica mestre/escravo, onde as estações mestre solicitam dados periodicamente das estações remotas. Sistemas SCADA encontrados no mercado geralmente oferecem um conjunto abrangente de protocolos de comunicação tais como Modbus, DNP3 e BACnet. Assim que são capturadas estas informações são mapeadas para entidades do sistema que representam as leituras, agregando atributos como estampilhas de tempo.

A aquisição de dados é uma funcionalidade essencial para qualquer sistema SCADA, pois todas as outras funcionalidades dependem dos dados coletados dos dispositivos. Portanto é importante que os dados sejam coletados em uma frequência que satisfaça as necessidades de monitoramento do processo. Outro requisito é a confiabilidade do canal de comunicação entre as estações mestre e remotas, dado o emprego destes sistemas em aplicações de missão crítica (GOMES, 2003).

2.4.1.2 Controle

As ações de controle permitem que o operador possa atuar sobre o processo. Esse controle é efetuado através do envio de comandos das estações mestre para as estações remotas, que ao receberem os comandos acionam seus

atuadores. Em sua natureza o controle de SCADA difere dos algoritmos de controle que executam nas estações remotas, sendo um controle em nível de supervisão (SOUZA, 2005). Ações típicas que podem ser efetuadas pelos operadores são o estabelecimento de *setpoints* para os algoritmos de controle de nível mais baixo e a atuação sobre o estado de equipamentos (o fechamento de uma válvula por exemplo).

2.4.1.3 Interface Homem Máquina

As interfaces homem-máquina, ou estações de operação tornam possível a interação de operadores humanos com os processos subjacentes (BAILEY; WRIGHT, 2003). Uma representação gráfica "rica" do processo é tradicionalmente uma preocupação nos sistemas SCADA, que devem representá-lo de uma forma intuitiva. A esta representação gráfica é dado o nome de sinótico, provendo ao operador uma visão geral dos processos através de animações.

Além de espelhar o comportamento de um processo estas interfaces também permitem que um operador interaja através do envio de comandos de controle (GOMES, 2003). Uma facilidade oferecida neste contexto é o gerenciamento de "receitas", que permitem ao operador definir um conjunto de valores para determinadas variáveis. Receitas são úteis pois permitem a reutilização de configurações para um processo especifico, por exemplo um operador pode definir uma receita para uma mistura química, definindo valores para variáveis de temperatura, pressão e os componentes utilizados.

Dada a diversidade de processos que um SCADA pode monitorar a maioria das interfaces permitem a edição de sinóticos, possibilitando a personalização das animações, imagens, sons, etc. Também fazem parte da interface gráfica áreas onde é possível visualizar e configurar os alarmes, gerenciar dispositivos, visualizar relatórios que permitem a apresentação de dados em tabelas, gráficos de diversas modalidades e ferramentas estatísticas como acompanhamento da tendência das variáveis monitoradas. (CASIMIRO, 1995).

2.4.1.4 Configuração

Um SCADA é composto de diversos componentes, que devem ser organizados e configurados de forma a cumprir com as necessidades da aplicação especifica onde serão utilizados. Esse arranjo inclui componentes de software, dispositivos de hardware e infraestrutura de comunicação.

Desta forma uma das funcionalidades destes sistemas é possibilitar a

configuração do software para o processo que será supervisionado. Entram nesta categoria o gerenciamento dos dispositivos que servirão como estações remotas, definição das variáveis que serão monitoradas, taxas de aquisição de dados, configuração de alarmes e sinóticos (CASIMIRO, 1995). Tais configurações devem ser persistentes, sendo que alguns sistemas permitem a exportação destas configurações para formatos que possam ser consumidos por outras aplicações (SEROTONIN, 2011).

2.4.1.5 Histórico

O histórico se refere ao armazenamento e possibilidade de consulta de séries histórias dos dados adquiridos do processo. Os dados coletados são importantes para a geração de relatórios, gráficos e tabelas e são essenciais para a realização de análises estatísticas como cálculo de médias, variâncias e previsões através das curvas de tendência (CASIMIRO, 1995).

2.4.1.6 Alarmes

Alarmes são notificações que indicam que um determinado evento ocorreu no processo que está sendo monitorado. A configuração destes eventos costuma seguir a mesma lógica, onde variáveis monitoradas são associadas a condições de disparo. Por exemplo, um operador pode associar a variável pressão de um tanque a um limite, quando este valor for atingido será gerado um alarme.

As condições mais comuns estão relacionadas com operações lógicas, embora alguns sistemas permitam a elaboração de condições mais elaboradas como baseadas em cálculos estatísticos e combinações de eventos (GOMES, 2003). A definição de prioridades para os alarmes também é uma funcionalidade tipicamente empregada para facilitar seu gerenciamento pelos operadores, dado que alguns alarmes podem ser de suma importância por estarem relacionados a ocorrência de situações críticas. Alguns sistemas também permitem a geração de alarmes a partir de eventos internos do próprio sistema (e não do processo), por exemplo a autenticação de um usuário no sistema (SEROTONIN, 2011).

2.4.2 Requisitos Não Funcionais

Existem propriedades que são geralmente citadas como requisitos para SCADA, abaixo serão descritas estas propriedades e suas motivações.

2.4.2.1 Interoperabilidade

Interoperabilidade é a capacidade de um sistema interagir com outros sistemas através da troca de informações (IEEE, 1991). Esta é uma propriedade que vem sendo enfatizada dada a crescente necessidade de integração entre processos na indústria, levando os aplicativos SCADA a interagirem com sistemas de apoio a gestão ou sistemas situados em outras organizações (FAVARETTO, 2001).

Para permitir esta interação com outras aplicações um SCADA deve disponibilizar uma interface programável acessível através da rede. Dependendo das necessidades da organização o acesso a esta interface pode estar confinado a aplicações da rede local ou então podem estar disponíveis através da Internet. Sistemas interoperáveis devem ter suporte para lidar com uma gama variada de plataformas de hardware e software. Em SCADA esta tem sido uma dificuldade devido a práticas como a adoção de tecnologias proprietárias e fechadas (URDANETA et al., 2007).

2.4.2.2 Portabilidade

A portabilidade é a capacidade de um sistema ser adaptado para executar em diferentes ambientes de hardware e software (GHEZZI; JAZAYERI; MANDRIOLI, 2002). Em SCADA esta propriedade permite reduzir a dependência do usuário do sistema de uma determinada plataforma de hardware ou software, agraciando-o com uma maior liberdade de escolha de fornecedores. Do ponto de vista dos desenvolvedores a portabilidade facilita a adaptação do sistema para diferentes plataformas, ocasionando em uma maior abrangência de mercado (GOMES, 2003). A escolha por determinadas tecnologias dependentes de plataformas específicas é um problema constante em SCADA, o padrão OPC baseado em DCOM é um exemplo, dada a sua dependência do sistema operacional Microsoft Windows.

2.4.2.3 Escalonabilidade

A escalonabilidade [1] em uma arquitetura de software está relacionada com capacidade que esta possui de suportar um número grande de componentes ou interações entre eles (FIELDING, 2000). Sistema escalonáveis permitem que um aumento da demanda possa ser tratado com um aumento linear de recursos físicos, sem que para isto seja necessário realizar alterações no software, com exceção de configurações para a utilização destes novos recursos físicos (BRATAAS; HUGHES, 2004).

Aplicativos SCADA escalonáveis são flexíveis o suficiente para serem utilizados em diferentes contextos, sejam estas exigentes ou não quanto a carga requerida. Outro aspecto é que eles estão preparados para a expansão dos sistemas monitorados, por exemplo uma fábrica que aumentou consideravelmente o número de equipamentos supervisionados (GOMES, 2003). Arquiteturas que gerenciam recursos de forma ineficiente, dificultando a distribuição de carga entre os componentes são apontadas como causadoras de problemas de escalonabilidade em aplicativos SCADA (URDANETA et al., 2007).

2.4.2.4 Confiabilidade

Confiabilidade é capacidade de um sistema continuar funcionando mesmo após a ocorrência de falhas em seus componentes (PERRY; WOLF, 1992). A confiabilidade é muitas vezes ressaltada em SCADA devido ao uso destes sistemas em aplicações de missão crítica que requerem alta disponibilidade, como o monitoramento de centrais de distribuição de energia elétrica e água (GOMES, 2003). Esta propriedade esta relacionada com os mecanismos de redundância e tolerância a falhas utilizados no sistema, portanto uma arquitetura de software deve suportar a introdução destes mecanismos, tornando possível melhorar a confiabilidade do sistema (URDANETA et al., 2007).

[1] O termo é oriundo do inglês *scalability*, sendo comumente traduzido na literatura de computação através do neologismo escalabilidade. Neste trabalho o termo escalonável é preferido em detrimento de escalável, pois o primeiro significa 'algo passível de ser colocado em níveis, etapas, escalões, degraus' enquanto que o segundo denota 'algo escalável', tal como uma montanha.

2.4.2.5 Segurança

A segurança está relacionada com a capacidade do sistema em resistir ao uso não autorizado ao mesmo tempo que permite que usuários legítimos consigam utilizá-lo. Esta propriedade pode ser caracterizada segundo a habilidade que o sistema tem de fornecer mecanismos que forneçam disponibilidade, confidencialidade, integridade e não repúdio (BASS; CLEMENTS; KAZMAN, 2003).

Assim como a confiabilidade, esta é uma propriedade muito enfatizada devido ao uso de SCADA em aplicações de missão crítica, cujo impacto de um ataque pode causar danos severos, tanto no aspecto humano como material (GOMES, 2003). Em 2010 um ataque massivo foi realizado contra aplicações SCADA através do verme Stuxnet, colocando em evidência a vulnerabilidade destes sistemas (FALLIERE; MURCHU; CHIEN, 2010).

2.4.2.6 Tempo-real

Sistemas de tempo-real são aqueles onde os aspectos temporais do seu comportamento fazem parte de sua especificação. Um sistema com restrições de tempo-real possui limites de tempo para a realização de suas tarefas, denominados *deadlines* (BERNAT; BURNS; LIAMOSI, 2001).

Alguns processos físicos controlados por um sistema SCADA podem possuir necessidades de interação em tempo real. Contudo esta restrição não é constante, variando de processo para processo (DAWSON et al., 2006). Como as funcionalidades do SCADA estão associadas com a visualização da informação pelo operador e sua correspondente ação, os *deadlines* para estes processos costumam estar na casa dos segundos (GOMES, 2003). Neste sentido os *deadlines* também costumam ser *soft*, sendo que sua perda não causa efeitos críticos no sistema (BALASUBRAMANIAN et al., 2009). Devido a natureza não-determinística da Internet não está no escopo deste trabalho tratar de aplicações SCADA cujos processos tenham requisitos de tempo real.

2.5 Conclusão do Capítulo

Neste capítulo foi apresentada uma visão geral de SCADA, abrangendo seus principais conceitos, tecnologias e requisitos de software, levando em conta alguns aspectos históricos que demonstram a evolução pela qual esta área tem passado. No próximo capítulo será feita uma descrição da Web e uma introdução a conceitos da área de arquitetura de software, que servem

como base teórica da Arquitetura Orientada a Recursos.

3 Arquitetura de Software e a Web

3.1 A Web

A Web é um sistema distribuído de hipermídia de alcance mundial que revolucionou a forma como pessoas, organizações e sistemas geram, acessam, e compartilham informações (BERNERS-LEE, 2000). Seu surgimento se deu em 1989, quando Tim Berner Lee viu a necessidade de compartilhar informações de pesquisadores da Organização Europeia para a Pesquisa Nuclear (CERN, na sigla em inglês). Desde o princípio seus requisitos eram desafiadores, os pesquisadores utilizavam sistemas diferentes, estavam espalhados por diversos países da Europa e as informações compartilhadas não tinham um formato comum (BERNER-LEE, b). Os sistemas da época eram muito engessados para prover acesso a informações, que necessitavam seguir uma estrutura pré-definida. A solução foi utilizar o conceito de hipertexto (NELSON, 1965), possibilitando que as informações fossem relacionadas livremente e permitindo que o sistema se expandisse na medida em que novas informações e relacionamentos eram incluídos, sem que para isto fosse necessário qualquer tipo de controle centralizado (BERNER-LEE, a).

Entre 1990 e 1991 foram publicadas na Internet as primeiras versões das especificações que formam a base da Web: os Identificadores Universais de Documentos (UDIs), que dariam origem aos Identificadores Uniformes de Recursos (URI), o Protocolo de Transporte de HiperTexto (HTTP) e a Linguagem de Marcação de Hipertexto (HTML). Também foram divulgadas livremente ferramentas que permitiam a qualquer indivíduo consumir, produzir e hospedar informações na Web, através do primeiro navegador, editor de hipertexto e servidor Web respectivamente. Nestes primeiros anos a Web passou a ser utilizada por pesquisadores do mundo todo, e suas especificações passaram a ser discutidas e modificadas colaborativamente, sendo criado em 1994 o World Wide Web Consortium (W3C) (W3C, 2000).

Nos idos de 1993 ficou claro que a Web não estaria confinada ao meio acadêmico quando indivíduos começaram a utilizá-la para publicar seus sítios pessoais e o mercado pelo seu uso comercial. A comunidade de desenvolvedores e pesquisadores da Internet passou a temer que o crescimento exponencial da Web causasse um colapso em toda a rede. Apesar de a Web primordial ter sido baseada em boas práticas de engenharia de software, como separação de responsabilidades, simplicidade e generalidade, surgia a necessidade de melhoramentos em aspectos de escalonabilidade, performance e extensibilidade para que problemas futuros pudessem ser evitados (FIELDING, 2000).

Todavia a Web não possuía uma descrição clara de sua arquitetura de software, o que dificultava o estabelecimento de uma filosofia comum pela qual seria guiado o seu projeto. Em seguida é apresentada a fundamentação teórica sobre arquitetura de software, para que fiquem claros os princípios que ajudaram a Web a continuar crescendo e mantendo suas boas propriedades.

3.2 Arquitetura de Software

Na medida em que cresce o tamanho e a complexidade de um sistema, é preciso tomar decisões de projeto que vão além da escolha de linguagens de programação, algoritmos ou estruturas de dados (GARLAN; SHAW, 1993). Essas decisões de projeto estão em um nível mais alto de abstração, se preocupando com questões relativas a organização geral do sistema, ou seja, com a sua arquitetura. De uma forma geral, uma arquitetura de software define os elementos que fazem parte de um sistema, incluindo a forma como estes elementos se relacionam para cumprir com os requisitos para os quais o sistema foi projetado (BASS; CLEMENTS; KAZMAN, 1997). Apesar do interesse da comunidade de pesquisadores de engenharia de software nesta área, existe pouco consenso quanto a uma definição precisa do termo arquitetura de software. Neste trabalho será utilizada a definição de (FIELDING, 2000), que descreve uma terminologia auto-consistente para a área.

Segundo (FIELDING, 2000) arquitetura de software é uma abstração dos elementos em execução de um sistema durante uma de suas fases de operação. Abstração é um conceito chave, pois permite omitir os detalhes que não são necessários para compreender o cerne da dinâmica de um sistema. Neste sentido, aspectos de implementação dos elementos que compõem uma dada arquitetura, tais como a linguagem de programação em que foram desenvolvidos, podem ser ignorados. Uma abstração está restrita a um determinado nível, sendo que um sistema pode possuir diversos níveis de abstração, cada um com uma determinada arquitetura. A definição também vincula o conceito de arquitetura diretamente com o sistema durante sua execução, tornando-a independente da parte estática, presente em seu código fonte e em qualquer descrição ou documentação de seu projeto. Outro aspecto é que um sistema pode apresentar diferentes arquiteturas para cada fase de sua operação, por exemplo as fases de inicialização, processamento e finalização no ciclo de vida de uma aplicação.

Uma arquitetura de software é definida por uma configuração de seus elementos. Estes elementos incluem componentes, dados e conectores. Os componentes são unidades abstratas de software que possuem um estado interno e realizam a transformação de dados através de sua interface (PERRY;

WOLF, 1992). O comportamento de um componente é definido pela sua interface, não importando para outros componentes como se dá a sua implementação. Por exemplo, um componente que realiza uma soma pode ser utilizado por outros componentes, basta que estes saibam como solicitá-la, o que é expresso através de interface deste componente (por exemplo quais parâmetros são aceitos na soma). Neste sentido um componente atua como uma ”caixa preta”, com uma interface bem definida para que outros possam utilizar seus serviços. O nível de abstração de uma arquitetura é delimitado pela interface exposta pelos seus componentes. Componentes por sua vez podem ser implementados internamente a partir de outros componentes, gerando uma recursão que termina quando são encontrados os componentes básicos, aqueles que não podem ser decompostos em outros menos abstratos.

Os conectores são os mecanismos abstratos que mediam a comunicação entre os componentes (SHAW; CLEMENTS, 1997). Estes elementos atuam como uma “cola” que une os componentes de uma arquitetura (GARLAN; SHAW, 1993). Os componentes de uma arquitetura nem sempre estão localizados em uma mesma máquina ou processo, portanto um conector pode envolver a implementação de algum protocolo de comunicação, podendo utilizar mecanismos como troca de mensagens, fluxos de dados (*streams*) e RPC com este propósito.

Dados são os elementos de informação trocados entre os componentes a partir de seus conectores (FIELDING, 2000). Os dados contém a informação que é transformada pelos componentes, sejam estas informações passadas como parâmetros, sequências de bytes ou através de objetos serializados. Uma distinção importante entre componentes e conectores é que os primeiros realizam algum tipo de transformação sobre os dados, enquanto os últimos apenas os transferem entre os componentes.

As propriedades de uma arquitetura incluem tanto os aspectos funcionais, que ditam o que o sistema deve fazer, quanto os não funcionais, que definem características desejadas tais como, performance, segurança e flexibilidade. É possível favorecer determinadas propriedades através da introdução de restrições nos componentes, conectores e dados e na forma como estes elementos se relacionam (FIELDING, 2000). Essas restrições são embasadas em princípios conhecidos de engenharia de software, como separação de responsabilidades. Diferentes domínios de aplicação possuem diferentes necessidades, de modo que algumas propriedades arquiteturais podem ser mais interessantes para um domínio do que para outro, portanto o projeto de uma arquitetura deve ser guiado pela aplicação de restrições que favoreçam as propriedades desejadas pelo domínio em questão.

3.2.1 Estilos Arquiteturais

Através da aplicação de boas práticas de engenharia de software os desenvolvedores passaram a perceber a existência de determinados padrões no projeto de arquiteturas, que passaram a ser descritos e categorizados (SHAW; CLEMENTS, 2006). Esses padrões entre arquiteturas distintas abstraem o que é essencialmente comum entre elas, e em uma analogia com os estilos arquitetônicos das construções levam o nome de estilos arquiteturais. Um estilo arquitetural é um conjunto de restrições comuns aplicadas nas arquiteturas que o seguem (FIELDING, 2000). Desta forma uma arquitetura de software pode ser vista com uma instância de um dado estilo arquitetural.

Arquiteturas de software envolvem propriedades funcionais, portanto é complicado comparar arquiteturas de domínios de aplicação diferentes. Por outro lado os estilos abstraem estes aspectos funcionais, se concentrando nas propriedades que são induzidas pelas restrições utilizadas. Estilos podem ser usados em diferentes domínios de aplicação, e comparados de acordo com a relevância das propriedades que eles favorecem para cada domínio específico (FIELDING, 2000). Estilos também podem ser combinados, formando estilos híbridos.

3.3 REST

Nos seus primórdios a Web não possuía uma descrição clara e objetiva de sua arquitetura de software. Seu projeto era guiado por alguns textos e discussões publicadas na rede e a melhor descrição de sua arquitetura de software estava internalizada em implementações. Era necessário estabelecer um modelo para a arquitetura de software da Web. Um modelo abstrato que formalizasse os princípios existentes, incorporando as modificações necessários para os requisitos que se apresentavam e que pudesse ser utilizado para avaliar modificações futuras. Tais necessidades motivaram o desenvolvimento do estilo arquitetural REST.

O estilo REST é híbrido, composto de diversos estilos arquiteturais para aplicações em rede, que quando aplicados em conjunto favorecerem as propriedades desejadas para um sistema distribuído de hipermídia na escala da Internet, como a Web. A seguir será apresentado um resumo da tese de (FIELDING, 2000), onde são descritos os estilos arquiteturais que compõem REST.

3.3.1 Os Estilos de REST

3.3.1.1 Cliente-Servidor

O estilo cliente-servidor é muito popular para aplicações em rede, abrangendo uma vasta gama de aplicativos utilizados para correio eletrônico, transferência de arquivos, base de dados, terminais remotos, etc (DOLLIMORE; KINDBERG; COULOURIS, 2005). Nesse estilo um componente servidor oferece serviços e espera por requisições, que são enviadas por componentes clientes interessados nestes serviços. Ao receberem as requisições os servidores decidem se as rejeitam, ou se enviam uma resposta ao cliente que as enviou (FIELDING, 2000). Sua motivação está no princípio da separação de responsabilidades, que restringe o papel de cada componente. Separando a gerência da interface gráfica do armazenamento dos dados é possível melhorar a portabilidade dos clientes e simplificar a implementação dos componentes servidores, melhorando a escalonabilidade. Outra propriedade reforçada é que os clientes e servidores podem evoluir de forma independente, desta forma a implementação de qualquer componente pode ser totalmente alterada, desde que a interface de comunicação entre eles não o seja. A figura 6 mostra o diagrama de um arquitetura estilo cliente-servidor, onde o componente cliente é representado pela elipse e o componente servidor pelo retângulo.

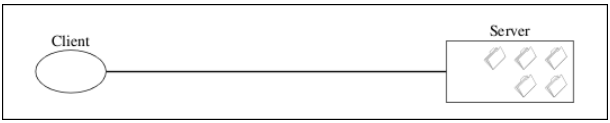

Figura 6: Cliente-Servidor

3.3.1.2 Sem-Estado

Ao estilo cliente-servidor adiciona-se também uma característica de comunicação sem estado. No estilo cliente-servidor-sem-estado cada requisição feita de um cliente deve conter toda a informação necessária para que o servidor possa compreendê-la, evitando que o servidor necessite guardar qualquer contexto de uma interação anterior (FIELDING, 2000). Essa restrição favorece a visibilidade, pois somente é necessário observar uma requisição

para entender seu resultado no sistema. A escalonabilidade é melhorada pois o servidor não necessita armazenar o estado compartilhado entre múltiplas requisições, facilitando a liberação de recursos. Uma desvantagem desse estilo é a redundância de dados, que precisam ser enviados novamente a cada requisição, ocupando mais a rede.

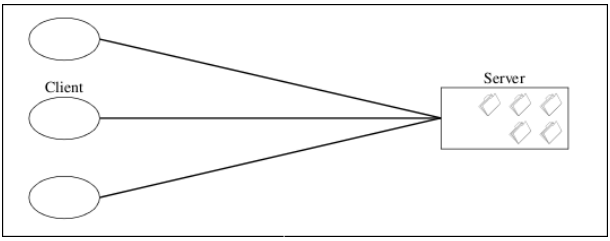

Figura 7: Cliente-Servidor-Sem-Estado

3.3.1.3 Cache

Com a combinação dos estilos cliente-servidor-sem-estado e cache chega-se ao estilo cliente-com-cache-servidor-sem-Estado. Esse estilo busca melhorar a eficiência no uso da rede através da adição de um conector de cache, com ele requisições marcadas como passíveis de cache podem ser reaproveitadas, eliminando assim a necessidade de realização de algumas interações (FIELDING, 2000). Com a cache as propriedades de eficiência, escalonabilidade e performance percebida pelo usuário são melhoradas. A desvantagem está no fato de que informações em cache podem estar desatualizadas, reduzindo a confiabilidade destas informações. O estilo Cliente-Cache-Servidor-Sem-Estado pode ser visualizado na figura 8 onde os conectores de cache (cifrão) são acrescentados aos componentes cliente.

3.3.1.4 Interface Uniforme

Aplicando o princípio de generalização à interface dos componentes chega-se ao estilo cliente-com-cache-servidor-sem-estado com interface uniforme. Através deste estilo a interface entre os componentes é uniformizada. Isto significa que existe uma forma padronizada de os componentes se comunicarem, que é a mesma para todos os componentes. A interface uniforme

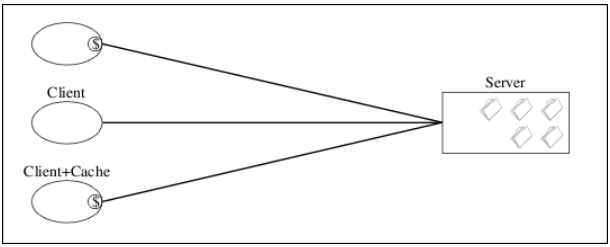

Figura 8: Cliente-Cache-Servidor-Sem-Estado

favorece a simplicidade, pois não é necessário lidar com as especificidades das interfaces arbitrárias. A visibilidade também é melhorada, pois existe uma semântica comum na comunicação entre os componentes, que pode ser verificada através da inspeção das mensagens trocadas entre eles. O problema desta uniformização é que toda a troca de informação é feita de uma forma genérica ao invés de ser específica para cada necessidade, piorando a eficiência em alguns casos através da transferência de mais informação do que é necessário. Outras restrições comportamentais entre os componentes são necessárias para que a interface seja uniforme: identificação de recursos, manipulação de recursos através de representações, mensagens autodescritivas e hipermídia(FIELDING, 2000). Estas restrições serão apresentadas nos elementos arquiteturais de REST, que serão descritos posteriormente neste capítulo. A uniformidade na comunicação entre os componentes deste estilo de arquitetura pode ser visualizada no diagrama da figura 9 através da existência dos conectores genéricos ligando os componentes.

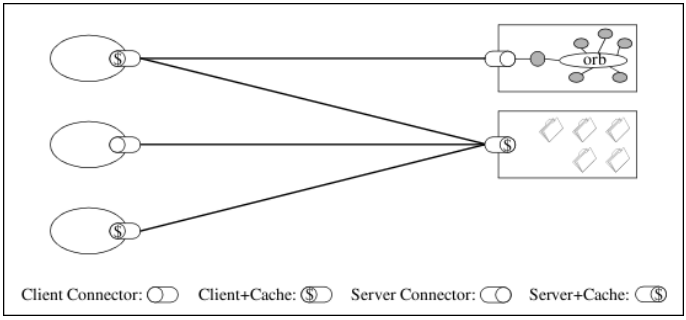

Figura 9: Cliente-Cache-Servidor-Sem-Estado-Uniformes

3.3.1.5 Sistema em Camadas

Neste estilo o sistema é dividido em camadas, sendo que cada camada só tem conhecimento da camada adjacente. Sua motivação está na separação de responsabilidades, desacoplando as camadas não adjacentes. Exemplos de sistemas em camadas podem ser encontrados em pilhas de protocolos de comunicação, como o modelo OSI/ISO e a pilha TCP/IP (PUDER; ROMER; PILHOFER, 2005). As camadas podem ser utilizadas para encapsular serviços legados, proteger novos serviços de clientes legados, melhorar a escalonabilidade através da distribuição de carga ou introduzirem políticas de segurança. A desvantagem deste estilo está na adição de latência extra na comunicação entre as camadas, podendo causar um impacto negativo na performance. Este problema é contrabalanceado através do uso de componentes de cache compartilhadas, que permitem que clientes distintos possam reaproveitar informações de uma cache comum, poupando novas requisições (FIELDING, 2000). A figura 10 ilustra o estilo em camadas, as camadas podem ser vistas da esquerda para a direita iniciando nos clientes que originam as requisições, passando por componentes intermediários como proxies e gateways e terminando nos servidores de origem que fornecem os recursos.

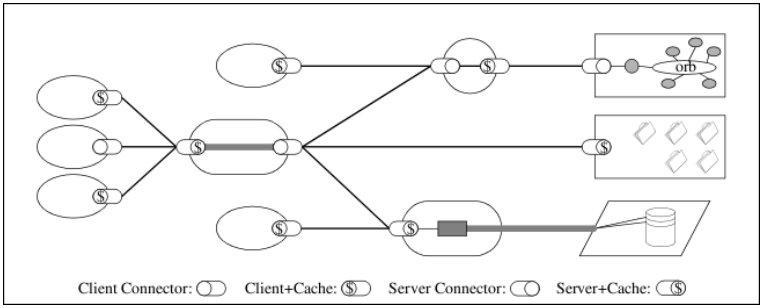

Figura 10: Cliente-Cache-Servidor-Sem-Estado-Uniformes-Em-Camadas

3.3.1.6 Código sob demanda

Com a inclusão do estilo código sob demanda é possível agregar funcionalidades ao cliente através de código executável fornecido pelo servidor. Esta capacidade permite uma simplificação na implementação dos clientes. A desvantagem está na redução da visibilidade, pois o código móvel precisa

ser interpretado, tornando mais complexa a tarefa de descobrir quais são seus resultados. O código sob demanda é a única restrição arquitetural opcional de REST (FIELDING, 2000). Na Web o código sob demanda está presente por meio de JavaScript e das Applets Java. Na figura 11 o código sob demanda é representado através das engrenagens e *scripts* circulando entre clientes e servidores.

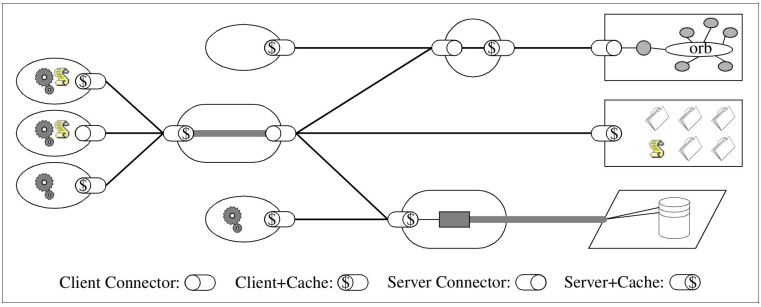

Figura 11: REST

3.3.2 Elementos Arquiteturais de REST

Descritos os estilos que compõem REST, dá-se prosseguimento com a apresentação de seus elementos arquiteturais, ou seja, seus dados, conectores e componentes.

3.3.2.1 Dados

REST possui três elementos de dados: recursos, identificadores de recursos e representações (FIELDING, 2000). Recurso é qualquer informação que possa ser nomeada. Qualquer conceito que possa ser referenciado através de hipermídia se encaixa nesta definição, seja ele um livro, uma fábrica, a imagem de uma cidade ou uma coleção de outros recursos. Recursos são mapeamentos que associam conceitos com um conjunto qualquer de entidades, estas entidades podem ser representações ou identificadores de recursos. É importante distinguir os recursos das entidades para as quais eles mapeiam. Por exemplo, um recurso que simboliza a última peça fabricada por uma máquina pode estar associado a diferentes peças ao longo do tempo, o im-

portante é que a semântica associada a ele permaneça constante, ou seja, que ele sempre esteja associado a última peça fabricada.

Identificadores de recursos são os elementos responsáveis por nomear e endereçar recursos. A identificação é unívoca, ou seja, dois recursos não podem possuir o mesmo identificador. São os identificadores que possibilitam que os recursos possam ser referenciados na interação entre os componentes da arquitetura, permitindo que os clientes possam expressar quais recursos desejam acessar, e aos servidores identificarem quais recursos estão sendo requisitados.

Os recursos se materializam através das representações. Uma representação é uma sequência qualquer de bytes e metadados que descrevem estes bytes. Os componentes manipulam os recursos através da transferência de representações, que descrevem o estado atual ou desejado do recurso. Os metadados descrevem a representação e o recurso, definindo por exemplo a ação desejada em uma interação com o recurso. Metadados também permitem a parametrização de requisições e respostas e aspectos de controle de cache.

3.3.2.2 Componentes

Os componentes de REST são categorizados pelo papel que exercem no âmbito geral da arquitetura. O componente agente de usuário utiliza o conector cliente para efetuar requisições e receber respostas. Um exemplo de agente de usuário é o navegador Web que requisita recursos e renderiza representações, exibindo-as para um usuário. Um componente servidor de origem utiliza um conector do tipo servidor e trata requisições de clientes. Este componente provê uma interface genérica para os recursos que administra, sendo uma fonte de representações de recursos e o último destino de requisições que busquem alterá-los (FIELDING, 2000).

Os outros dois componentes, gateway e proxy, atuam ao mesmo tempo como clientes e servidores, exercendo funcionalidades como encaminhamento de requisições e respostas, encapsulamento de outros serviços e implementação de políticas de segurança. Mais especificamente, o proxy é um componente intermediário explicitamente escolhido pelo cliente. Já o gateway é utilizado pelo servidor de origem, de forma que o cliente não precisa saber de sua existência. O gateway também é conhecido como proxy reverso.

3.3.2.3 Conectores

Os conectores provêm uma interface uniforme para a manipulação de recursos, encapsulando as atividades de acesso e transferência de representações. Os conectores fundamentais de REST são o cliente e o servidor. O cliente é quem inicia a comunicação, efetuando uma requisição, o servidor espera por conexões e responde a requisições que estejam acessando seus serviços. Um componente pode possuir ambos os conectores (FIELDING, 2000).

Conectores do tipo cache podem ser utilizados por clientes ou servidores para armazenarem mensagens, reutilizando-as com objetivo de reduzir o número de interações realizadas na rede. O conector resolvedor traduz identificadores em endereços de rede para estabelecer a interação entre componentes. Já o túnel possui a função de criar um caminho virtual para travessia de informações de forma a transpor limites, como um firewall ou um roteador.

3.4 Arquitetura Orientada a Recursos

A Arquitetura Orientada a Recursos (ROA) é uma arquitetura guiada pelos princípios de REST e baseada em tecnologias consolidadas da Web como HTTP, URI e tipos e mídia (RICHARDSON; RUBY, 2007). Esta arquitetura surgiu das dificuldades que os desenvolvedores enfrentavam ao projetar aplicações Web com base em REST. As dificuldades estão relacionadas ao alto nível de abstração dos princípios de REST, que abrem um amplo horizonte de possibilidades de projeto. Entre as dificuldades está o fato de que REST não está vinculado com a Web e suas tecnologias, pois é um estilo arquitetural para sistemas de hipermídia distribuídos, dos quais a Web é uma instância especifica. Deste fato se deriva diretamente outra dificuldade, que é como utilizar as tecnologias da Web de acordo com as restrições do estilo REST. Das necessidades de uma abordagem mais concreta e pragmática para os problemas que se colocavam surgiu a ROA. Esta arquitetura segue os princípios de REST, mas está vinculada com as tecnologias da Web como o protocolo HTTP, URI e formatos de dados como XML, XHTML e JSON. A seguir serão apresentados seus conceitos, tecnologias e propriedades.

3.4.1 Conceitos

3.4.1.1 Recursos

Os recursos são o cerne da ROA, como seu seu próprio nome indica. Os recursos são definidos a partir de conceitos existentes no domínio da aplicação em questão. Por exemplo, em uma aplicação sobre SCADA existem conceitos específicos, como equipamentos de campo, coleções de dados adquiridos e alarmes.

Coleções de outros recursos são recursos típicos, por exemplo a coleção de todos os equipamentos em uma planta, que mapearia para um conjunto de enlaces, cada qual apontando para o recurso de um equipamento. Resultados de algoritmos também podem ser expostos como recursos, por exemplo uma busca por um determinado equipamento, ou uma filtragem do histórico de aquisição de dados de um CLP. Outro recurso comum neste tipo de arquitetura é o ponto de entrada da aplicação, que expõe uma coleção dos recursos da aplicação, pelos quais um cliente pode navegar. O projetista também pode definir diferentes graus de granularidade para um mesmo sistema, criando novos recursos ou fundindo recursos existentes conforme as suas necessidades.

3.4.1.2 URI

Cada recurso precisa ser identificado para que os diferentes componentes interessados nele possam referenciá-lo e interagirem com ele. O que determina que determinado conceito do sistema é um recurso é o fato de este ser identificado a partir de uma URI. Cada URI identifica unicamente um recurso na Web e simultaneamente o torna endereçável, possibilitando que ele seja manipulado. A relação entre recurso e URI é uma relação um para muitos, ou seja, um recurso pode ser referenciado por muitas URIs, mas uma URI só pode ser associada com um recurso.

Nesta arquitetura é aconselhado que sejam escolhidas URIs cujos nomes possam ser associados ao significado dos recursos para o qual apontam. Por exemplo, a URI para o recurso da coleção de equipamentos de uma fábrica poderia ser: ”/fabrica/equipamentos”. A razão deste conselho de utilizar URIs descritivas está no uso desses endereços por pessoas. Além das máquinas que consomem informações, estão usuários que podem memorizar facilmente uma URI que lembre algum conceito. Desta forma um dos objetivos da ROA é construir recursos que possam ser bem aproveitados por pessoas e por máquinas.

3.4.1.3 HTTP

É necessário que exista um protocolo para que os componentes da arquitetura possam se comunicar, possibilitando que os componentes sirvam, acessem e modifiquem recursos. Para isto a ROA utiliza o protocolo HTTP, um protocolo de nível de aplicação para sistemas de hipermídia distribuídos que forma a base de comunicação de toda a Web.

O protocolo HTTP possui um estilo cliente-servidor, onde a interação entre os componentes se dá através de um modelo requisição/resposta. Os clientes realizam requisições para servidores, indicando o recurso nos qual estão interessados através de uma URI. Estas requisições também contêm informações a respeito da ação que o cliente deseja realizar sobre o recurso. Devido a sua interface uniforme o HTTP define um conjunto finito de operações, com semântica determinada, que podem ser realizadas em qualquer recurso da Web (FIELDING et al., 1999). Os métodos HTTP utilizados na ROA, e suas semânticas são descritos abaixo. Esta descrição será mais detalhada posteriormente, quando for apresentada a metodologia de projeto da ROA.

* GET: Obtém a representação de um recurso;
* PUT: Cria ou atualiza um recurso na URI especificada;
* POST: Cria recursos subordinados a URI especificada. Anexo de informações a um recurso existente ou submissão a um processamento de dados;
* DELETE: Remove um recurso;

Além de indicarem os recursos e operações sobre eles, estas interações realizam a troca de informações sobre os recursos. Isto é feito através de representações de recursos. Cada resposta possui uma representação, por exemplo a resposta de uma requisição GET contém a representação do estado atual do recurso requerido. Requisições que criam recursos, ou atualizam suas informações, tais como PUT e POST também carregam representações, que descrevem o recurso criado ou os dados que serão atualizados.

As representações possuem formatos de dados, que podem variar entre diversos tipos de linguagens de marcação (HTML, XML, XHTML), imagens (PNG, SVG), textos (TXT, CSV), entre muitos outros. Para distinguir entre os formatos o HTTP utiliza o conceito de tipo de mídia (o inglês Media-type), herdeiro da especificação MIME utilizada em correio eletrônico. Através de metadados colocados nos cabeçalhos das requisições e respostas, o HTTP

explicita o tipo de mídia das representações. Isto possibilita que clientes possam decidir o que fazer com a representação de acordo com esta informação. Outro aspecto é que clientes e servidores podem negociar conteúdo, com os clientes expondo quais tipos de mídia preferem, e os servidores expondo quais tipos de mídia oferecem. Outros metadados também são utilizados nas requisições e respostas para fornecer informações sobre os componentes e também controle de cache.

O HTTP é intrinsecamente sem-estado, portanto quando o cliente envia uma requisição HTTP, ele inclui toda a informação necessária para que o servidor satisfaça a requisição. O servidor por sua vez não necessita armazenar o contexto da comunicação com cada um de seus clientes, podendo tratar cada requisição como um evento isolado.

Outro aspecto do protocolo é o seu suporte a componentes intermediários na comunicação entre clientes e servidores. Esta característica permite a existência de *proxies* e *gateways*, possibilitando caches compartilhadas entre clientes distintos e também o encapsulamento de sistemas legados.

3.4.1.4 Aspectos de Segurança

O protocolo HTTP possui mecanismos de autenticação incorporados a sua especificação: os métodos Basic e Digest. Ambos métodos permitem a utilização de credenciais com identificação de usuário e senha, embutidas nas requisições do protocolo em uma interação com o servidor no estilo Desafio-Resposta (do inglês Challenge-Response). O que difere essencialmente nos mecanismos é a forma com que são representadas as credenciais. O servidor, ao receber estas requisições pode confirmar ou negar o acesso de clientes aos recursos solicitados. As credenciais são sempre enviadas nas requisições, não existindo qualquer mecanismo nativo para manutenção de sessões nestes métodos, pois acrescentaria estado na comunicação.

No método Basic a dinâmica se dá da seguinte forma. Quando o cliente não utiliza as credenciais na requisição para um recurso protegido, ele recebe uma resposta do servidor indicando que não está autorizado a acessá-lo (código HTTP 401) e informando-o através do cabeçalho "WWW-Authenticate" que o método de autenticação é o Basic. O parametro "Realm" também é informado na resposta, e permite ao servidor descrever grupos de recursos guardados por autenticação, permitindo por exemplo que os clientes exibam aos usuários a descrição do grupo e armazenem em cache as informações para se autenticar nos mesmos (FRANKS et al., 1999). Quando recebe esta negativa o cliente pode fazer uma nova requisição, desta vez contendo as suas credenciais concatenadas e codificadas em Base64 no cabeçalho

"Authorization". O servidor por sua vez passa a aceitar o acesso ao recurso protegido.

O método Digest surgiu porque a codificação em Base64 empregada no HTTP Basic pode ser decodificada facilmente. De fato a intenção da codificação Base64 não é tornar segura a transmissão das credenciais, mas sim possibilitar o uso de caracteres não compatíveis com a codificação permitida pelo HTTP. No Digest as credenciais são protegidas em um esquema mais complexo que envolve a aplicação de criptografia utilizando a função de hash MD5 (FRANKS et al., 1999). É importante frisar que somente as credenciais são protegidas por criptografia nesse esquema. Como veremos a seguir o uso de HTTPS suplanta a necessidade do Digest. Desta forma sua vantagem fica restrita a contextos onde o overhead do HTTPS não é desejado, mas formas mais seguras de esconder as credenciais são necessárias.

Conhece-se por HTTPS o uso combinado do HTTP com os protocolos SSL/TLS. Estes protocolos, utilizados na camada de transporte (TCP), possibilitam que toda comunicação HTTP seja criptografada e realizada por meio de um canal seguro, garantindo a confidencialidade e integridade das informações (DIERKS; ALLEN, 1999). Estes protocolos utilizam mecanismos de criptografia assimétrica que permitem a assinatura digital das mensagens. Através da assinatura digital a autenticidade das partes comunicantes pode ser comprovada, e a integridade das mensagens garantida. Os protocolos permitem duas configurações, a mútua e a simples. Na configuração mútua clientes e servidores necessitam possuir certificados, utilizados na assinatura digital das mensagens. Na simples somente o servidor necessita possuir o certificado, sendo amplamente utilizada na Web. Neste caso, como forma de garantir a autenticidade dos clientes, o HTTPS pode ser combinado ao esquema de autenticação Basic.

Não existe um padrão universal na Web para lidar com a questão da autorização, embora existam propostas como o OAuth sendo utilizadas pela comunidade (HAMMER-LAHAV, 2010). Desta forma, geralmente cada aplicação acaba por definir seus próprios mecanimos para cumprir com este requisito.

3.4.1.5 Representações

O principal critério na escolha de representações é o tipo de representação que os clientes desejam consumir. A ROA propõe que sejam utilizadas tipos de mídia comuns da Web, tais como XML, XHTML, e JSON. Representações que permitam a inclusão de hipermídia também são aconselhadas, pois permitem conectividade entre os recursos, conceito que será

explorado posteriormente.

3.4.2 Propriedades

A ROA também define algumas propriedades que uma aplicação que segue esta arquitetura deve possuir: Endereçabilidade, Sem-estado, Conectividade e Interface Uniforme.

3.4.2.1 Endereçabilidade

Aplicações endereçáveis expõem as informações que são interessantes para seus clientes através de recursos na Web. Pelo fato de possuírem URIs, as informações tem um endereço certo onde podem ser acessadas. Pessoas podem encontrá-las navegando na Web, através de enlaces nas representações de outros recursos. Encontrado um endereço, este pode ser referenciadas em novos recursos, salvo nos favoritos do navegador, ou mesmo anotado em um papel como um lembrete. Outras aplicações também podem se beneficiar desta endereçabilidade, guardando endereços para verificar informações periodicamente, e seguindo enlaces automaticamente como fazem os chamados ”crawlers”, robôs que são muito usados em motores de busca.

3.4.2.2 Sem-estado

A propriedade sem-estado é diretamente derivada do estilo arquitetural REST. A enfase da ROA está no uso correto do protocolo HTTP, sem a utilização de mecanismos que implementem sessões. É importante esclarecer que o conceito de estado se refere ao estado do cliente em relação com a aplicação. Por exemplo, ao navegar na Web um cliente segue uma série de enlaces, cada acesso representa um acesso a um recurso. Informações como quais recursos o cliente já visitou e a ordem em que foram acessados são ignoradas pelo servidor em uma aplicação sem-estado. Sendo assim, se existe algum interesse neste tipo de informação ela deve ser guardada pelo cliente. Isto evita a complexidade adicional que o servidor teria, dado que seria necessário analisar o histórico de requisições para tomar qualquer decisão. Além de trazer complicações, manter estado traz problemas de performance, pois é preciso guardar o contexto de cada cliente no servidor.

3.4.2.3 Conectividade

Os recursos da aplicação devem estar conectados através de enlaces em suas representações. Desta forma os clientes podem escolher qual será seu próximo passo, ou seja o próximo estado da aplicação. Esta propriedade confere a uma aplicação orientada a recursos o aspecto de uma teia, que pode estar ligada inclusive a outras aplicações, ou recursos quaisquer da Web.

3.4.2.4 Interface Uniforme

Operações sobre os recursos são realizadas através dos métodos HTTP. A interface para o acesso e manipulação dos recursos é uniforme, permitindo que exista um "vocabulário" comum, com o qual é possível interagir com qualquer recurso na Web. A seguir são descritos os métodos HTTP utilizados na ROA e suas semânticas.

Qualquer recurso para o qual seja possível a um cliente obter uma representação deve expor o método GET. Os outros métodos servem para modificar os recursos. O método PUT recebe uma representação e é responsável por criar um recurso na URI especificada, ou então atualizar a sua representação caso já exista um recurso nesta URI. O método POST é responsável por criar recursos subordinados ao recurso informado na URI, a inclusão de um item em um recurso de coleção é um exemplo comum de uso do método POST. Outros usos existem, como por exemplo anexar mais informação a um recurso existente, ou submeter um bloco de dados a um processamento qualquer.

A diferença sutil entre os métodos PUT e POST para criação e atualização de recursos está no significado da URI indicada pelo cliente na requisição. No caso do método POST esta URI representa um ponto de processamento da requisição, mas não necessariamente ela indica a URI onde o recurso será criado ou atualizado, cuja responsabilidade é inteiramente do servidor que gerencia este recurso. No método PUT, a URI especificada pelo cliente na requisição obrigatoriamente define o endereço onde este cliente espera que esteja o recurso após o processamento da requisição pelo servidor. O método DELETE por sua vez tem como propósito remover um dado recurso.

3.4.3 Projeto de uma Arquitetura Orientada a Recursos

Um metodologia para o projeto de ROAs é apresentada em (RICHARDSON; RUBY, 2007). Este método é baseado em uma série de observações

pragmáticas de como transformar os requisitos de uma aplicação em recursos da Web. A metodologia define uma sequência de etapas, que tem como objetivo o projeto de uma ROA a partir de um conjunto de funcionalidades de um dado domínio de aplicação. A seguir serão descritas estas etapas, explicando as decisões de projeto envolvidas em cada uma delas.

3.4.3.1 Levantamento do conjunto de dados

O primeiro passo no projeto de uma ROA está no levantamento de seu conjunto de dados. Nesta fase do projeto deve-se analisar o domínio de aplicação, identificando qualquer informação que possa vir a ser útil para seus clientes. Por exemplo, uma aplicação sobre SCADA poderia fornecer informações a respeito dos equipamentos de campo, alarmes, últimos dados adquiridos por um dado equipamento e histórico de dados. O importante nesta etapa é definir quais são as informações disponibilizadas, como estas informações serão disponibilizadas é um assunto para os próximos passos.

3.4.3.2 Definição dos Recursos

Com o conjunto de dados definido o próximo passo é expor estas informações como recursos. Uma ideia similar a da modelagem orientada a objetos pode ser aplicada. A ideia envolve a identificação de sujeitos presentes no vocabulário comum do domínio de aplicação, por exemplo: equipamento, alarme, histórico, última leitura. Esses sujeitos são um bom indicativo de quais são os recursos envolvidos em uma aplicação. Outros recursos típicos são os relacionados com coleções e resultados de algoritmos, por exemplo uma lista de todos os equipamentos ou o histórico dos dados coletados por um sensor na última semana.

3.4.3.3 Escolha das URIs

Existem algumas decisões na escolha de URIs que são consideradas boas práticas, uma delas é a utilização do simbolo “/” em recursos que possuem alguma relação hierárquica. Por exemplo o recurso de todos os equipamentos de uma fábrica poderia ser identificado por ”/equipamentos”, enquanto que “/equipamento/grua” poderia identificar um equipamento específico, no caso uma grua. Outra boa prática é o uso de URIs que possam ser facilmente compreendidas por pessoas, ajudando usuários finais e proje-

tistas da arquitetura a entenderem de forma intuitiva os conceitos envolvidos em cada recurso.

3.4.3.4 Exposição de um subconjunto da interface uniforme

Após definir como um recurso será identificado é necessário especificar quais são as operações possíveis que um cliente pode efetuar neste recurso. Este passo se resume a definir o subconjunto de métodos da interface uniforme do protocolo HTTP pelas quais será possível manipular o recurso. Qualquer recurso para o qual seja possível a um cliente obter uma representação deve expor o método GET. A Alteração de Recursos é realizada através do método PUT. A criação pode ser feita por POST ou PUT, dependendo de quem (cliente ou servidor) é o responsável por definir a URI onde o recurso é criado. Já o método DELETE deve ser exposto por recursos que permitam sua remoção.

3.4.3.5 Projeto das representações

Neste passo devem ser definidas quais informações representam cada recurso, assim como o formato utilizado para representá-las. No âmbito do formato, a lógica está na seleção de tipos de mídia padronizados da web (como o HTML e XML), que satisfaçam as necessidades dos clientes e servidores envolvidos. Diferentes representações podem ser definidas para um mesmo recurso, de forma que clientes podem aproveitar deste aspecto através da técnica de negociação de conteúdo do protocolo HTTP.

3.4.3.6 Integração com outros recursos

Definidas as representações o próximo passo é interligar os recursos com outros. Esta associação pode ser feita com qualquer recurso julgado interessante, seja ele um recurso do sistema que está sendo projetado ou externo a ele. Recursos são relacionados entre si através de enlaces presentes nas suas representações, portando a escolha de representações que permitam hipermídia(como XML, HTML e XHTML) no passo anterior é crucial para uma boa conectividade.

3.4.3.7 Definição das respostas

Nesta etapa é definida a interação entre cliente e servidor, ou seja, como se dão os diálogos de requisição/resposta entre eles. Deve ser respeitado o protocolo HTTP, que define metadados como os códigos de resposta de acordo com a requisição que foi feita, além de estabelecidas quais podem ser as representações enviadas no conteúdo de cada resposta. Neste ponto devem se pensar nos casos onde as mensagens são processados com sucesso pelos componentes, assim como nos casos onde ocorre algum erro. Nos caso onde houve alguma falha deve ser informado qual o componente (cliente ou servidor) foi o responsável pelo erro. Um resumo do procedimento completo presente na metodologia para projetar uma arquitetura orientada a recursos a partir de um dado domínio de aplicação é apresentado abaixo:

1. Levantamento do conjunto de dados
2. Definição dos recursos, e para cada recurso:
3. Nomear o Recurso com URIs;
4. Expor um Subconjunto da Interface Uniforme;
5. Projetar as Representações;
6. Integrar com Outros Recursos;
7. Definir Respostas;

3.5 Conclusão do Capítulo

Neste capítulo foi apresentada uma descrição da Web e dos conceitos da área de arquitetura de software que fundamentam seus príncipios de arquitetura. Na descrição da Web foi abordado seu histórico e as motivações que levaram a elaboração de um estilo arquitetural que favorecesse seus requisitos. Em seguida foram apresentados os principais conceitos de arquitetura de software, incluindo a terminologia presente no referencial teórico da área. Com base nestes conceitos chega-se ao estilo arquitetural REST, o estilo arquitetural híbrido que fundamenta os príncipios arquiteturais da Web. Por fim conclui-se o capítulo com a descrição da ROA, proposta como uma abordagem pragmática que incorpora as tecnologias da Web como HTTP, URI e tipos de mídia em conformidade com o estilo REST. O capítulo seguinte expõe uma análise sobre arquiteturas de software baseadas em RPC onde são

explorados os pontos em que estas arquiteturas, predominantes em SCADA, divergem dos princípios arquitetônicos da Web dificultando sua integração com a mesma.

4 Arquiteturas de Software para SCADA na Web

Pode-se observar que os sistemas SCADA têm feito uma transição na direção de possibilitar o seu uso como aplicativos Web. No entanto ainda é muito difícil encontrar sistemas SCADA que participem de forma plena da Web, não tendo sido projetados de acordo com os princípios de arquitetura de software descritos no capítulo anterior. Na indústria predominam abordagens baseadas em RPC para o desenvolvimento destes sistemas, principalmente devido a influência do OPC, que se tornou um padrão de facto na indústria (HONG; JIANHUA, 2006)(KAPSALIS; KOUBIAS; PAPADOPOULOS, 2002)(TORRISI; OLIVEIRA, 2008). Neste capítulo, serão apresentadas arquiteturas para SCADA explorados os pontos em que estas arquiteturas divergem dos princípios da Web.

4.1 Arquiteturas Baseadas em RPC

O RPC é um mecanismo para comunicação entre aplicações distribuídas (DOLLIMORE; KINDBERG; COULOURIS, 2005). As ideias em torno deste paradigma são discutidas desde os anos 70 (WHITE, 1976), sendo a formalização de seus príncipios creditada a (BIRRELL; NELSON, 1984). Os conceitos do RPC são centrados na abstração das chamadas de procedimento, presentes em linguagens de programação como C e Pascal. Procedimentos são trechos de código que cumprem com determinada tarefa, podendo ser chamados em diferentes pontos de um programa. Quando um procedimento é chamado, aquele que o chamou passa para ele um conjunto de parâmetros, o fluxo de execução é então transferido para o procedimento chamado, que irá efetuar uma computação utilizando estes parâmetros. Efetuada a computação, o fluxo de execução é devolvido para o componente que o chamou, juntamente com o resultado da operação.

O mecanismo para chamadas remotas é similar, a diferença é que os componentes se encontram distribuídos em nós de uma rede. Chamadas remotas tem uma dinâmica de requisição/resposta, sendo efetuadas por um componente solicitante através do envio de uma mensagem de requisição, que contêm o procedimento solicitado juntamente com os dados dos parâmetros. O receptor da mensagem extrai os parâmetros da requisição, executa o procedimento solicitado, e envia uma mensagem de resposta contendo os dados do resultado da computação. Geralmente esta comunicação é feita de forma síncrona, onde o componente que chamou o procedimento fica bloqueado en-

quanto não receber uma resposta (BIRRELL; NELSON, 1984).

A similaridade entre chamadas locais e remotas levou a integração de RPC com as linguagens de programação. Esta integração permite que componentes se comuniquem da mesma forma, independentemente se estão localizados em um mesmo nó ou em nós distintos da rede, propriedade conhecida como transparência de localização (DOLLIMORE; KINDBERG; COULOURIS, 2005). Com a transparência de localização, o programador pode desenvolver código de forma usual através de procedimentos, enquanto uma camada de software, conhecida como *middleware*, cuida de todas as responsabilidades relacionadas com a comunicação remota.

Nos termos de arquitetura de software, o RPC se faz presente nos conectores, fornecendo o mecanismo de comunicação entre os componentes. Ele é tipicamente empregado em arquiteturas de software em rede que seguem o estilo cliente-servidor. Neste estilo um servidor fornece seus serviços através de seu conector RPC, expondo um conjunto de procedimentos que poderão ser requisitados por seus clientes. A implementação dos conectores através de componentes locais denominados *stubs* permite o tratamento das chamadas remotas como se fossem locais, garantindo a transparência de localização. No conector do cliente um *stub* recebe as chamadas locais e deve conhecer a interface do servidor para que possa empacotar os parâmetros e realizar as chamadas remotas. No conector do servidor um *stub* tem como responsabilidade desempacotar estes parâmetros e enviar respostas com dados empacotados para seus clientes. Muitas vezes são empregadas ferramentas de geração de código, para poupar o programador do trabalho relacionado ao desenvolvimento dos stubs. Estas ferramentas tipicamente utilizam linguagens que descrevem a interface dos componentes, denominadas linguagens de descrição de interface (PRYCE, 1998).

Com o passar dos anos o paradigma de programação procedural foi sendo substituído pela orientação a objetos, e o desenvolvimento de sistemas orientados a objetos distribuídos passou tomar a cena nos anos 90 com o surgimento de tecnologias como Common Object Request Broker Architecture (CORBA, da sigla em inglês) e o Distributed Component Object Model (DCOM, da sigla em inglês). Apesar de serem baseados em um novo paradigma, estes sistemas utilizavam praticamente os mesmos conceitos oriundos do RPC tais como os *stubs* e as linguagens de descrição de interface, com a principal diferença que as chamadas de procedimentos remotos foram substituídas por invocações remotas de métodos. Fruto desta geração, a tecnologia DCOM causou um grande impacto no mundo de SCADA, pois foi com base nela que foi elaborada a especificação do OPC. O OPC é considerado um padrão de facto para a área, por este motivo uma visão geral dele será apresentada a seguir.

4.2 OPC

O OPC é um conjunto de especificações para automação industrial, que define a troca de informações entre equipamentos. Tais equipamentos podem ser dispositivos de campo, estações remotas, estações mestre ou interfaces homem máquina. Os dados trocados entre eles são típicos de aplicações SCADA, como dados adquiridos de campo, séries históricas, alarmes e comandos de controle (HONG; JIANHUA, 2006).

Seu surgimento se deu nos anos 90, motivado pela diversidade de equipamentos e protocolos de comunicação existentes na automação industrial, combinada com a ausência de qualquer forma de padronização para esta comunicação. Este cenário, onde cada equipamento era acessado de uma forma distinta, gerava problemas. Cada fabricante oferecia soluções completas de hardware e software, baseadas em protocolos proprietários que não permitiam interoperabilidade, limitando as opções de escolha das indústrias. Outro aspecto estava nas dificuldades encontradas pelo mercado de software, que precisava desenvolver diferentes drivers para cada equipamento ou protocolo encontrado nas plantas, gerando custo e complexidade (TU et al., 2010). Este cenário está exposto na figura 12 (HONG; JIANHUA, 2006), onde três aplicações se comunicam com equipamentos distintos, e para isto necessitam de três drivers diferentes.

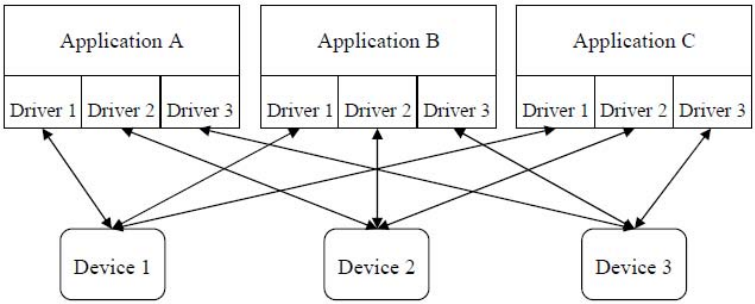

Figura 12: Complexidade antes do surgimento do OPC

O OPC propõem a adoção de interfaces padronizadas para a comunicação entre os componentes de software. A comunicação ocorre no estilo cliente-servidor, onde os clientes são as aplicações interessadas nos dados fornecidos pelos dispositivos (eg: uma interface homem máquina). Um servidor OPC expõe uma interface padronizada para o acesso aos dados dos dispositivos, desta forma os clientes podem ter acesso as informações pelas quais

estão interessados sem precisarem conhecer a especificidades de cada equipamento. Para implementar o acesso a estas informações em cada equipamento, o servidor se comunica com drivers OPC fornecidos pelos fabricante. A figura 13 (HONG; JIANHUA, 2006) ilustra como o OPC soluciona os problemas do caso exposto anteriormente.

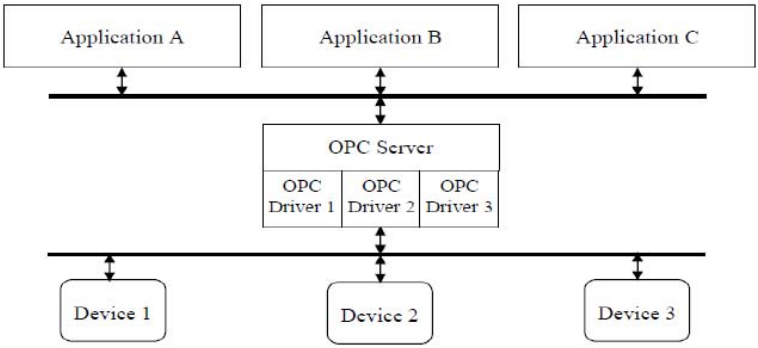

Figura 13: Arquitetura baseada em OPC

As especificações do OPC são baseadas na tecnologia DCOM, da Microsoft. O DCOM é um padrão para comunicação entre objetos distribuídos, baseado no seu predecessor, o Component Object Model (COM), surgido da necessidade de comunicação entre aplicações na plataforma Windows (PRYCE, 1998). Arquiteturas de software baseadas no estilo de objetos distribuídos decompõem as aplicações em um conjunto de objetos, que colaboram entre si. Cada objeto é uma entidade que possui um estado interno e dados, que só podem ser acessados ou alterados através de operações (também conhecidas como métodos). As operações definem a interface dos objetos, e são a forma com que os objetos se comunicam. Quando as interações são remotas, as chamadas seguem um modelo requisição/resposta e devem transportar os parâmetros da solicitação e da resposta. A principal diferença entre objetos distribuídos e o RPC procedural está na gerência de referências e coleta de lixo (FIELDING, 2000).

A tecnologia DCOM estabelece uma linguagem, a Microsoft Interface Definition Language (MIDL), que serve para descrever as interfaces dos componentes. Além de servir como documentação, a MIDL é utilizada na geração de código fonte nas linguagens C/C++, automatizando a criação dos stubs utilizados na comunicação. A comunicação entre os objetos por sua vez, se dá através do protocolo ORPC (Object RPC).

O OPC é formado por um conjunto de especificações, cada uma delas

abrange determinadas funcionalidades da uma aplicação de automação industrial. Cada especificação determina classes de objetos, e suas interfaces. O OPC-DA define interfaces para acesso a dados adquiridos por equipamentos dotados de sensores, tais como CLPs e RTUs. A especificação OPC-HDA define o acesso a séries históricas. Já o gerenciamento de alarmes e eventos é responsabilidade da especificação OPC-AE. Existem também outras especificações, que abarcam funcionalidades, como receitas e comandos de controle(TU et al., 2010).

Apesar de sua predominância em SCADA, existem limitações no OPC que o impede de suprir as novas necessidades que vem se apresentando para área. Entre os problemas está a questão da interoperabilidade e independência de plataforma. A tecnologia DCOM é fortemente vinculada ao sistema operacional Windows, limitando seu uso em outras plataformas. Outra questão é a comunicação na Internet, dado que o protocolo utilizado pelo OPC é frequentemente bloqueado por firewalls. Por ser totalmente desvinculado da Web, também existe um fator que acaba por isolar as aplicações OPC do restante do mundo, limitando a integração com sistemas de MES, ERP entre outros(TU et al., 2010).

4.3 OPC-UA

As limitações do padrão OPC motivaram a criação de um novo padrão, o OPC Unified Architecture (OPC-UA). O nome "unificado" do OPC-UA se refere a unificação de todas as especificações de interfaces (OPC-DA, OPC-HDA, OPC-AE...), simplificando a arquitetura (HADLICH, 2006). A especificação do OPC-UA é recente, tendo sido publicada em 2006, e atualmente está em fase de pesquisas e adoção pela indústria (HANNELIUS; SALMENPERA; KUIKKA, 2008).

O OPC-UA eliminou os problemas de dependência de plataforma e trouxe os conceitos do OPC para Web, através de uma abordagem baseada na pilha de Web Services. Em termos de arquitetura de software o estilo baseado em objetos distribuídos foi mantido. As maiores mudanças estão nas tecnologias utilizadas, apoiadas em padrões da Web. As mensagens passaram a ser descritas na linguagem XML, através do protocolo SOAP. Os protocolos proprietários do DCOM foram abandonados, e o HTTP passou a ser adotado para o transporte das mensagens. As interfaces passaram a ser definidas através de Web Service Description Language (WSDL), uma linguagem de descrição para Web Services. A especificação inclui uma forma de representar modelos semânticos de dados através de uma hierarquia extensível de tipos, permitindo que nichos específicos da industria especifiquem seus próprios padrões

de dados. O uso destas tecnologias trouxe um avanço, melhorando aspectos de interoperabilidade, e trazendo os sistemas desenvolvidos mais próximos de uma integração com a Web.

4.4 DPWS

Devices Profile for Web Services (DPWS da sigla em inglês) é uma especificação baseada em um subconjunto da pilha de Web Services direcionada para comunicação em dispositivos com restrições de recursos computacionais (OASIS, 2006). Seu foco é direcionado ao nível dos dispositivos, lidando com questões como descoberta dinâmica e propagação de eventos. Nesta linha o DPWS se apresenta como um sucessor do padrão UPnP (Universal Plug and Play) na comunicação descentralizada entre dispositos, frequentemente empregada em áreas como domótica e computação ubíqua (SLEMAN; MOELLER, 2008).

Ao contrário do OPC-UA, o DPWS se origina de uma área mais próxima da TI convencional do que da informática industrial. O OPC-UA é orientado para servir como uma interface para que aplicações de mais alto nível como IHM, MES e ERP possam obter dados oriundos do nível onde se encontram os dispositivos de campo. Desta forma o OPC-UA possui um foco mais direcionado para as funcionalidades das aplicações SCADA do que o DPWS (CANDIDO et al., 2010). Como o DPWS e o OPC-UA compartilham a pilha de especificações de Web Services existem propostas de uma integração destas abordagens, buscando cobrir os requisitos dos diferentes níveis da automação industrial (BONY; HARNISCHFEGER; JAMMES, 2011) (IZAGUIRRE; LOBOV; LASTRA, 2011).

4.5 Arquiteturas Baseadas em RPC e sua Integração com a Web

Padrões como o OPC-UA foram influenciados pela leva de tecnologias de Web Services, surgidas no final dos anos 90. Os Web Services tem como objetivo criar padrões para que aplicações desenvolvidas em diferentes plataformas pudessem interoperar (W3C, 2004). Estes padrões se apoiam em tecnologias da Web, especificamente o XML e o HTTP, como soluções para os problemas de interoperabilidade enfrentados com os sistemas de objetos distribuídos CORBA e DCOM. A primeira especificação publicada com este propósito foi o XML-RPC, lançado em 1998. O XML-RPC é uma especificação para realização de RPC utilizando XML para a codificação das mensagens, e o protocolo HTTP para os transporte destas mensagens através

da Web (WINER, 1999). O XML-RPC trouxe um novo folêgo para o RPC, levando seus conceitos para a Web. Pouco depois do lançamento do XML-RPC foi publicada a primeira versão do SOAP, que assim como XML-RPC, definia padrões para realizar RPC na Web utilizando XML e HTTP (BOX KAKIVAYA, 1999).

Ao longo dos anos o SOAP e seus derivados da pilha de especificações de Web Services ganharam espaço, e fizeram um esforço para generalizar as especificações, permitindo outros estilos de comunicação que não fossem os do RPC. O problema das arquiteturas baseadas em RPC para a Web é que elas utilizam tecnologias como o XML e HTTP de uma forma não condizente com os princípios de arquitetura da Web. Desta forma, requisitos importantes da Web, como evolução independente, visibilidade das interações e escalonabilidade podem ser prejudicados.

Um dos problemas é a questão do acoplamento. Um dos princípios fundamentais da engenharia de software está no desenvolvimento de sistemas modulares que apresentem alta coesão e fraco acoplamento (MYERS, 1978). O acoplamento é uma propriedade que representa a interdependência entre módulos, sendo que um acoplamento fraco é considerado benéfico por facilitar a compreensão, testabilidade e evolução de um sistema (GHEZZI; JAZAYERI; MANDRIOLI, 2002). De uma forma geral, a literatura de Web Services caracteriza os sistemas com acoplamento fraco como aqueles onde diferentes serviços interoperam através de um pequeno conjunto de pressupostos e, portanto, o impacto de mudanças é limitado e os serviços podem evoluir independentemente (KAYE, 2003). O acoplamento forte nas aplicações distribuídas tipicamente as torna engessadas para se adaptarem a mudanças, prejudicando a evolução independente dos sistemas e implicando em um forma de cooperação mais burocrática com outras aplicações (PAUTASSO; WILDE, 2009).

Arquiteturas baseadas em RPC compartilham um modelo comum de dados entre as aplicações. Isto implica que as mensagens trocadas entre os componentes contêm as entidades internas destes sistemas codificadas de alguma forma (objetos de domínio serializados por exemplo). A existência deste modelo de dados compartilhado introduz um forte acoplamento, pois qualquer mudança na definição do modelo deve ser replicada entre todas as partes envolvidas. Quando existem poucos serviços sobre o controle de uma mesma organização este não parece um problema tão grande, mas ele cresce de magnitude na medida em que mais organizações passam a cooperar. No HTTP não existe a dependência de um modelo de dados compartilhado. As mensagens trocadas entre os componentes são direcionadas a recursos, que possuem representações com formatos de dados padronizados através de tipos de mídia. Os clientes podem escolher entre diferentes representações,

não dependendo de qualquer formato específico. Estas representações podem ser processados de forma padronizada, sendo que existem muitas bibliotecas de software disponíveis para seu processamento, não criando dependência de ferramentas especificas (VINOSKI, 2008a).

O acoplamento também se manifesta através das interfaces especializadas fornecidas pelos componentes de uma arquitetura baseada RPC. Nestas arquiteturas, os componentes se comunicam pela rede através do uso de uma Application Programming Interface (do inglês API) local, formada pelos stubs. Esta API é responsável pela transparência de localização, e sua implementação realiza a comunicação remota. O problema está na dependência de uma infraestrutura de software comum e dependente da aplicação entre os componentes. Em uma arquitetura orientada a recursos não existe esta dependência, dado que toda interação se dá diretamente através do protocolo HTTP, independente de especificidades da aplicação (PAUTASSO; WILDE, 2009).

Outro aspecto prejudicado nas arquiteturas baseadas em RPC é a visibilidade das interações entre os componentes. A visibilidade se refere a habilidade de um componente monitorar ou mediar uma interação entre dois outros componentes (FIELDING, 2000). A visibilidade pode ajudar na performance possibilitando a existência de caches compartilhados, escalonabilidade por meio de sistemas em camadas e a segurança através de firewalls. Nas arquiteturas baseadas em RPC as interfaces dos componentes não possuem uma interface uniforme, desta forma cada aplicação acaba por definir um conjunto de operações e um modelo de dados especifico para si própria. Isto prejudica a visibilidade, pois os componentes intermediários precisam conhecer a semântica especifica de cada aplicação com qual se relacionam, o que se torna inviável em escalas como a da Web (VINOSKI, 2008b).

A falta de visibilidade nas interações dificulta o processamento das mensagens por componentes intermediários. As caches são um exemplo destes componentes (VINOSKI, 2008b). Porém sem saber a semântica da operação e sem informações de controle presentes nas mensagens (como a data de vencimento da cache) uma cache não pode atuar. O uso de cache é crítico para sistemas distribuídos em larga escala na Web, e sua não utilização impacta em propriedades importantes como a performance percebida pelo usuário e a escalonabilidade.

Outra decisão arquitetural que pode prejudicar a visibilidade nas interações é que aplicações baseadas em RPC podem manter estado entre as interações. Desta forma um componente que deseje descobrir o propósito de uma mensagem pode ter que inspecionar todo o histórico de mensagens anteriores a ela. Manter estado na comunicação também causa impactos na escalonabilidade, pois o servidor necessita armazenar este estado para cada

cliente (FIELDING, 2000).

Serendipidade é um neologismo derivado da palavra inglesa *"serendipity"* definida como a habilidade de atrair ou descobrir coisas úteis por acaso (NEWMAN et al., 2002). No contexto dos sistemas distribuídos este é um termo que vem sendo explorado nos últimos anos para designar o uso de um serviço de formas que não foram planejadas durante o seu projeto. O uso recente dos *"mashups"* na Web, aplicações que combinam dados e serviços de outras aplicações para seus propósitos são um exemplo prático desta propriedade (RAZA; HUSSAIN; CHANG, 2008). Arquiteturas baseadas em RPC dificultam o reuso dos serviços, pois nada se pode assumir a priori sobre suas interfaces. Outra questão é que a ausência da hipermídia acaba por criar serviços desconectados uns dos outros, evitando que um cliente descubra novos serviços por acaso através da navegação pelos enlaces (VINOSKI, 2008b).

4.6 Conclusão do Capítulo

Este capítulo realizou uma análise sobre arquiteturas de software para SCADA na Web. Primeiramente foram descritos os conceitos das arquiteturas baseadas em RPC, predominantes em SCADA. Em seguida foram apresentadas as tecnologias OPC e OPC-UA, baseadas em RPC e populares em aplicações SCADA. Posteriormente foi feita uma análise das arquiteturas baseadas em RPC com relação a sua integração com a Web. Nesta análise foram levantados os problemas de escalonabilidade, visibilidade e evolução independente que as arquiteturas baseadas em RPC enfrentam quando utilizadas na Web, causados pelo forte acoplamento, interfaces especializadas e manutenção de estado. No capítulo seguinte será projetada uma arquitetura para aplicações típicas de SCADA, condizente com princípios de arquitetura que fundamentam a Web.

5 Proposta de Arquitetura Orientada a Recursos para SCADA na Web

Neste capítulo mostra-se como uma aplicação SCADA típica pode ser organizada de forma que seus requisitos sejam atendidos dentro dos princípios arquiteturais que fundamentam a Web. Conforme foi visto no capítulo 2, sistemas SCADA possuem um conjunto de funcionalidades em comum, tais como aquisição de dados, configurações, históricos e alarmes. A ideia geral da proposta é explorar como estas funcionalidades podem ser atendidas por uma ROA através de um estudo de caso, tomando como cenário um ambiente característico de SCADA.

Primeiramente é feita uma descrição do objeto de estudo: uma CFM utilizada na simulação de processos fabris. São levantadas as informações relativas a CFM e ao processo em si, e alguns cenários que exemplificam situações típicas de uma aplicação SCADA são ilustrados.

Para o projeto da arquitetura é aplicada a metodologia orientada a recursos descrita no capítulo 3. Inicia-se com uma visão geral da arquitetura, em seguida as informações da CFM e do processo em si são utilizadas no projeto dos recursos. A descrição do projeto é realizada de forma incremental, onde novos recursos são criados na medida em que as funcionalidades típicas de SCADA são incorporadas na arquitetura.

Os requisitos não funcionais geralmente enfatizados em SCADA também são levados em conta no projeto da arquitetura. Com o objetivo de mostrar aspectos de interoperabilidade, simplicidade e portabilidade foi realizada uma implementação, que possibilitou que fossem desenvolvidas aplicações que interagem com os recursos projetados. Esta implementação foi realizada através da modificação de um software livre para SCADA: o Mango M2M, que teve sua arquitetura de software adaptada para as necessidades da arquitetura proposta. Segue-se agora com a descrição do estudo de caso.

5.1 Arquitetura Orientada a Recursos para Aplicações Típicas de SCADA

As CFMs são parte integrante de um Sistema Flexível de Manufatura (FMS, na sigla em inglês). FMSs são sistemas de produção altamente automatizados, capazes de produzir uma grande variedade de diferentes peças e produtos usando o mesmo equipamento e sistema de controle (CHAN; CHAN, 2004). A CFM utilizada no estudo de caso foi construída pelo Instituto Federal de Santa Catarina (IFSC) para o DAS/UFSC com objetivos

didáticos, sendo uma plataforma para simulação de processos de manufatura automatizados (BENTO et al., 2007). A célula é composta por uma garra, uma mesa rotativa, dois alimentadores, três depósitos e quatro estações de trabalho que efetuam as operações de furação, solda, teste e retrabalho de peças. Um desenho esquemático pode ser visto na figura 14 (GARCIA; QUEIROZ, 2009), enquanto que uma foto da célula e sua bancada são exibidas nas figuras 15.

Comportamentos distintos podem ser apresentados pela célula, conforme sua programação. Neste estudo de caso utiliza-se do trabalho (GARCIA; QUEIROZ, 2009), onde um comportamento é proposto para a célula, modelado a partir da teoria de controle supervisório modular local (QUEIROZ; CURY, 2000). De acordo com o comportamento proposto, a célula deve furar e soldar peças depositadas na mesa pela garra, que pode retirar estas peças dos alimentadores de peças novas e retrabalhadas. Ao final do processo a peça é testada em busca de imperfeições e depositada pela garra em um dos armazéns disponíveis. Peças aprovadas são depositadas no armazém de aprovadas, peças que reprovam no teste e que foram retiradas do alimentador de peças novas são depositadas no armazém de retrabalho enquanto que as provenientes do alimentador de peças retrabalhadas são armazenadas no armazém reprovadas.

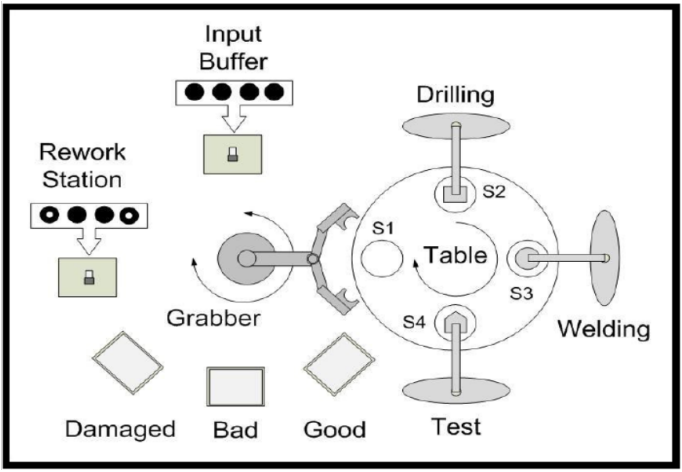

Figura 14: Célula de manufatura flexível

A organização dos equipamentos na CFM é típica de um SCADA, pois estão presentes os dispositivos de campo, representados por sensores

Figura 15: Foto da CFM

e atuadores, uma estação remota, constituída por um CLP, e estações mestre e de operação identificadas por computadores. A figura 16 ilustra esta organização.

Os sensores identificam a chegada de uma peça nos alimentadores e armazéns, e reconhecem se os equipamentos estão operando (estações de furação, solda, teste e motores da garra e da mesa). Atuadores estão presentes para acionar os motores e estações. Somente os sensores de chegada de peça influenciam no controle supervisório da planta, os demais são utilizados para fins de monitoramento

O CLP visto na figura 17 foi utilizado para efetuar a aquisição de dados e o controle dos equipamentos, através da ligação de suas entradas e saídas nos sensores e atuadores da CFM. Este equipamento possui suporte ao protocolo Modbus (Modbus Organization, 2011) que permite a leitura e escrita de dados em sua memória (Altus, 2011), possibilitando o controle e supervisão pelo SCADA. O algoritmo de controle supervisório que coordena o comportamento da CFM foi programado em sua memória, e não deve ser confundido com o controle exercido pelo SCADA. Para se comunicar com o CLP o computador foi ligado a porta serial do equipamento.

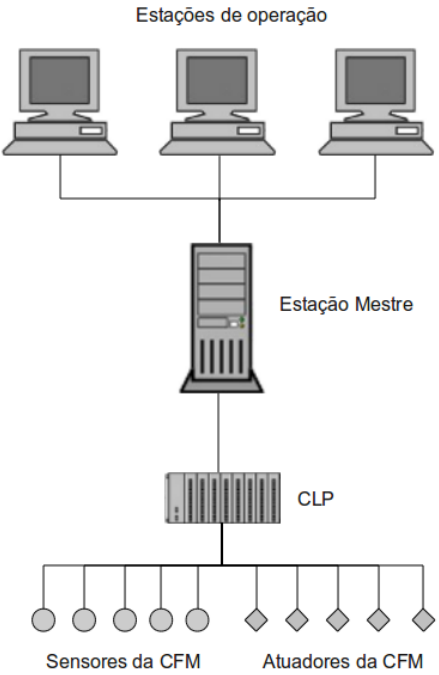

Figura 16: Organização dos componentes de hardware do SCADA na CFM

Figura 17: CLP Altus PO 3147 usado na CFM

5.1.1 Visão Geral da Arquitetura

Com base nos componentes encontrados tipicamente em SCADA é possível delinear uma visão geral de uma arquitetura que os integre na Web. Nesta arquitetura ilustrada pela figura 18 estão presentes os servidores SCADA, responsáveis por expor na Web os recursos para as aplicações clientes. Os recursos são a interface das aplicações clientes para com as funcionalidades típicas de SCADA, como histórico de dados e alarmes.

No contexto dos componentes de hardware encontrados em SCADA estes servidores estarão ambientados típicamente nas estações mestre, que possuem recursos computacionais suficientes para executar um servidor HTTP, como é o caso dos computadores da CFM. Entretanto, a arquitetura também está preparada para suportar estações remotas que atuem como servidores SCADA. Tais estações remotas trazem oportunidades interessantes, fornecendo acesso via Web as suas configurações e dados coletados.

Outro aspecto essencial é incluir na arquitetura aqueles dispositivos que não estão habilitados para a Web. Entram neste grupo todas as estações remotas que por necessidades do ambiente industrial (tempo-real por exemplo), limitações de plataforma, ou falta de recursos computacionais não são capazes de se comunicar via HTTP. Um exemplo deste caso é o CLP da CFM, que se comunica através do protocolo Modbus RTU e não tem capacidade de executar um servidor HTTP. Para lidar com esta questão podem ser utilizados gateways, responsáveis por fazer a ponte entre o protocolo nativo do dispositivo e a Web. O gateway fornece uma interface Web para aplicações cliente e encapsula a comunicação com o dispositivo incompatível com a Web.

As aplicações clientes podem ser classificadas de acordo com seus objetivos quanto aos recursos fornecidos pelos servidores. A aplicação utilizada pelo operador da CFM pode acessar os recursos para montar uma IHM com o sinótico da célula, configurar o CLP e atuar sobre o processo. Para uma interface gráfica mais dinâmica, a aplicação do operador pode aproveitar-se de código móvel residente no servidor (como JavaScript ou Applets Java) para enriquecer as capacidades do navegador. O cliente também pode se beneficiar de um conector de cache para evitar a realização de algumas requisições, melhorando a performance percebida pelo usuário.

Aplicações MES podem se interessar por dados históricos, como contagens de peças para controle de qualidade, enquanto que uma aplicação ERP pode utilizar as mesmas informações para realizar controle de estoque e agendar novas compras. Também podem existir aplicações interessadas em eventos específicos, como a quebra de um equipamento, o que pode ser realizado através da espera pelo disparo de alarmes.

Como a arquitetura proposta está fundamentada nos príncipios de ar-

quitetura da Web, podem ser incluídos componentes intermediários entre clientes e servidores. Uma cache compartilhada pode ser instalada entre clientes de uma mesma rede, de forma que os recursos acessados são salvos na cache melhorando a perfomance e a escalonabilidade. Um proxy de autenticação também pode ser incluído com o objetivo de adicionar credenciais de autenticação nas requisições não autenticadas, desacoplando funcionalidades.

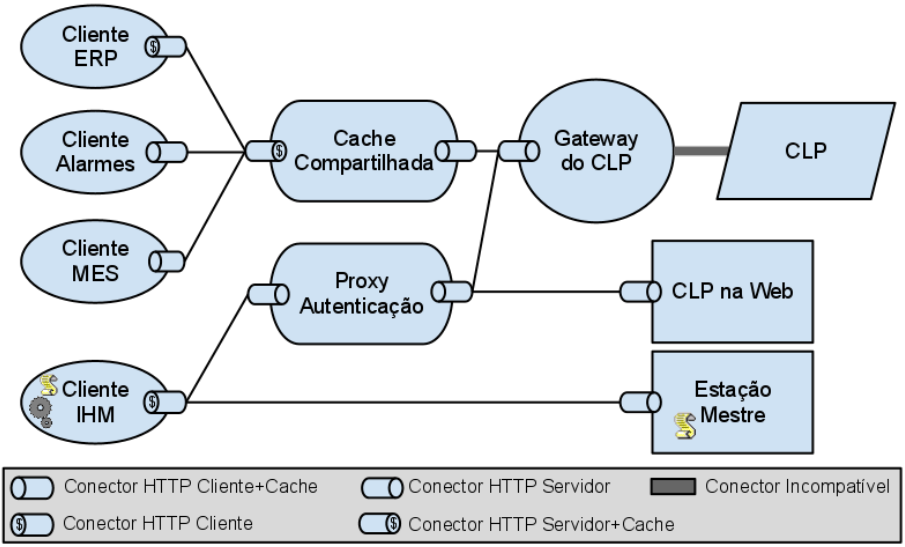

Figura 18: Arquitetura ROA para aplicações típicas SCADA

5.1.2 Projeto dos Recursos

Na visão geral da arquitetura estão presentes os componentes e conectores, segue-se com o projeto dos recursos e suas representações, que constituem os dados trocados entre os componentes. Para projetar os recursos são utilizados dados do domínio em questão, desta forma são recapituladas as funcionalidades típicas de SCADA.

Um dos requisitos típicos de SCADA é a possibilidade de configurar os dispositivos coletores de dados de acordo com as necessidades do processo onde serão utilizados. Na CFM, o CLP possui diversos parâmetros de configuração para realizar a aquisição de dados. Em outros casos, como

Recurso	URI
Coleção de coletores de dados	/coletores
Um coletor de dados	/coletor/{id}

Tabela 1: Recursos de coleta de dados

uma fábrica, poderiam existir diversos dispositivos de aquisição de dados (CLPs ou sensores inteligentes), cada qual com seu próprio protocolo de comunicação e parâmetros configuração específicos.

Cada dispositivo de coleta de dados (ou simplesmente coletor) é modelado como um recurso, tendo como informações o protocolo de comunicação utilizado, parâmetros de configuração e estado de operação (ativo ou inativo). Um recurso de coleção também é definido, contendo enlaces para todos os coletores existentes. Na tabela 1 os recursos relacionados aos coletores e suas URIs.

Para estes recursos é natural que se permita obter as informações de uma coleção de coletores ou de um determinado coletor específico. Estas informações podem ser apresentadas em uma interface gráfica qualquer, onde um técnico poderia saber quais são, e como estão configurados os coletores de dados do processo supervisionado. Desta forma é natural que estes recursos implementem o método GET, permitindo que clientes possam obter suas informações.

As necessidades do processo podem se alterar e novos dispositivos podem ser adicionados ou removidos do sistema. Portanto deve ser possível que clientes criem coletores de dados, atualizem as configurações de um coletor existente ou removam um coletor que não seja mais necessário.

A criação de um coletor é realizada através da inclusão do coletor na coleção. Isto é realizado através de um requisição POST no recurso da coleção de coletores, tendo como representação as informações do coletor que o cliente deseja criar. A escolha do método POST ao invés do PUT no caso da criação se dá pois é o servidor o responsável por definir a URI onde o recurso será criado. Por exemplo, se um cliente enviar uma requisição POST contendo uma representação válida de um coletor para a URI do recurso da coleção de coletores, o servidor poderá responder que conseguiu criar o recurso (código HTTP 201) e na representação da resposta incluir a URI onde este recurso foi criado, como: ”/coletores/1”(onde o número 1 é um identificador único definido pelo servidor).

Um coletor pode ter as suas configurações atualizadas através de uma requisição PUT, contendo uma representação com as informações de configuração desejadas. Por exemplo, o técnico da célula pode julgar que a

Recurso	GET	POST	PUT	DELETE
Coleção de coletores de dados	X	X		
Um coletor de dados	X		X	X

Tabela 2: Interface dos recursos de coleta de dados

frequência de aquisição de dados do coletor Modbus está muito baixa, e deseja atualizá-la, para isto ele utiliza uma aplicação cliente que realiza uma requisição PUT no recurso do coletor, informando na sua representação a frequência desejada. Da mesma forma o coletor pode ter a sua aquisição de dados ativada ou desativada, o que pode ser necessário para alguma manutenção. No caso do coletor não ser mais necessário para o sistema, ele pode ser removido através de uma requisição DELETE. Os métodos da interface uniforme definidos para os recursos de coleta de dados podem ser vistos na na tabela 2.

No projeto das representações fornecidas pelos recursos, a primeira decisão é escolher os tipos de mídia que serão utilizados. O formato JSON foi escolhido devido a sua simplicidade, por ser compacto e facilmente lido por humanos e interpretado por máquinas. Por ser um subconjunto da linguagem JavaScript este formato é interpretado nativamente por todos os navegadores que a suportam, sem a necessidade de bibliotecas adicionais para realização do *parsing*.O formato JSON possui duas estruturas, uma coleção de pares chave/valor (equivalente a um objeto ou tabela associativa em outras linguagens) ou uma lista ordenada de valores. Cada valor pode por sua vez assumir diferentes tipos: textual, numérico, booleano, objeto ou lista(JSON, 2010).

Tipos de mídia adicionais também podem ser definidos para cada recurso, fornecendo opção para que cada cliente especifique qual o formato mais apropriado para ele. Este mecanismo é conhecido como negociação de conteúdo, e é parte integrante da especificação do protocolo HTTP. Nela o cliente pode especificar o cabeçalho “Accept” nas suas requisições, fornecendo os tipos de mídia preferidos para que o servidor possa decidir o que fazer.

Definido o tipo de mídia, resta especificar quais dados serão embutidos em cada representação. A representação da coleção de coletores não necessita mais informação do que uma lista com os enlaces para cada coletor. Já a representação de um coletor deve fornecer todas as informações a respeito do mesmo, tais como: o protocolo utilizado, configurações e se o mesmo está ativo ou inativo. Exemplos das representações da coleção de coletores e de um coletor Modbus configurado para adquirir dados do CLP da CFM por ser vistos nas figuras 19 e 20. Na figura 21 um exemplo de como ficaria uma

representação alternativa do recurso de coletor, no formato XML.

```
"coletores": [
        {"nome" : CLP, "uri": "/coletor/1"}
]
```

Figura 19: Representação da coleção de coletores

```
{
 "nome":CLP,
 "protocolo": "ModbusRTU",
 "taxaDeAtualização":500,
 "taxaDeBaud":9600,
 "timeout":500,
 "portaSerial":COM1,
 "ativo": "sim",
}
```

Figura 20: Representação do coletor Modbus RTU

```
<coletor>
        <nome>CLP</nome>
        <protocolo>ModbusRTU</protocolo>
        <taxaDeAtualização>500</taxaDeAtualização>
        <taxaDeBaud>9600</taxaDeBaud>
        <timeout>500</timeout>
        <portaSerial>COM1</portaSerial>
        <ativo>sim</ativo>
</coletor>
```

Figura 21: Representação XML alternativa do coletor Modbus RTU

Outra funcionalidade típica de SCADA diz respeito a configuração de quais dados serão coletados. Na célula existem diversas variáveis que podem ser monitoradas, também conhecidas por outros termos, como entidades telemetradas (GOMES, 2003), pontos (SEROTONIN, 2011) (National Communications Systems, 2004) ou etiquetas (tags)(ADAMO et al., 2007). Exemplos de variáveis que podem ser colhidas da CFM são: o estado atual dos equipamentos (garra, mesa e as estações de furação, solda e teste de qualidade) e a contagem de peças que foram aprovadas, reprovadas ou enviadas para retrabalho.

Recurso	URI
Coleção de variáveis	/variaveis
Uma variável	/variavel/{idDaVariável}

Tabela 3: Recursos de variáveis do processo

Cada variável é associada a um tipo de dado, que classifica os valores possíveis que ela pode assumir, tais como dados booleanos, numéricos, enumerações e imagens. No caso da CFM, a contagem de peças pode ser classificada como um valor numérico, enquanto que a informação de que determinado equipamento está ligado ou desligado é um valor booleano.

Ações de controle permitem que o operador possa atuar sobre o processo. Estas ações geralmente são atribuídas as variáveis. Por exemplo um operador da célula poderia ligar ou desligar a furadeira enviando um comando de controle para a variável que a representa.

Como os dados são adquiridos pelo dispositivo coletor também se torna importante especificar como a variável pode ser localizada nos dados colhidos por ele. Na CFM o protocolo Modbus utilizado pelo CLP possui uma forma particular de endereçamento, onde trechos da memória do dispositivo servidor são expostos para os clientes. Permitindo que um cliente que está interessado em uma informação defina parâmetros como faixa de registradores, tipo do dado (por exemplo binário ou um inteiro sem sinal) e deslocamento de endereço que resultam na informação que será lida ou escrita.

Com esta análise, podem ser definidos alguns novos recursos. Cada variável do processo por si só é um recurso com informações como nome, tipo de dado e especificações de como podem ser lidas pelo coletor. Uma coleção de variáveis também é um recurso necessário, por listar todas as variáveis existentes no processo. Na tabela 3 os recursos relacionados as variáveis do processo e suas URIs.

Informações associadas a esses recursos podem ser obtidas pelos clientes, pois permitem que saibam quais são as variáveis supervisionadas e qual a semântica de cada uma delas, portanto o método GET deve ser exposto para esses recursos. Também interessa a técnicos e operadores possuirem a capacidade de criar, modificar e excluir variáveis. Desta forma são expostas as operações POST, PUT e DELETE. Novas variáveis são criadas através do método POST na coleção de variáveis, enquanto que uma variável pode ser modificada ou removida através das operações PUT e DELETE, respectivamente. Ações de controle também podem ser efetuadas através da operação PUT, especificando na representação da variável os parâmetros da ação de

Recurso	GET	POST	PUT	DELETE
Coleção de variáveis	X			
Uma variável	X		X	X

Tabela 4: Interface dos recursos de variáveis do processo

controle. Os métodos da interface uniforme definidos para os recursos podem ser vistos na tabela 4.

O próximo passo é escolher como serão as representações para os recursos. Na definição dos recursos relacionados aos coletores foi escolhido o tipo de mídia JSON, que será mantido, portanto basta escolher quais informações estarão presentes em cada representação. As representações de coleções de variáveis não necessitam de mais informações do que uma lista com os enlaces para cada variável. A representação de uma variável fornece todas as informações da mesma, tais como: nome, descrição, tipo de dados, endereço no coletor e enlaces para seus recursos de última leitura e histórico. Exemplos de representações da coleção de variáveis e da variável da contagem de peças aprovadas da célula podem ser vistos nas figuras 22 e 23.

```
"variáveis": [
        {"nome" : peçasAprovadas, "uri": "/variavel/1"}
        {"nome" : peçasReprovadas, "uri": "/variavel/2"}
        {"nome" : peçasDeRetrabalho, "uri": "/variavel/3"}
        {"nome" : peçasNovas, "uri": "/variavel/4"}
        {"nome" : furadeira, "uri": "/variavel/5"}
        {"nome" : solda, "uri": "/variavel/6"}
        {"nome" : teste, "uri": "/variavel/7"}
        {"nome" : mesaGirando, "uri": "/variavel/8"}
        {"nome" : garraPegandoPeça, "uri": "/variavel/9"}
        {"nome" : garraSeMovendo, "uri": "/variavel/10"}
]
```

Figura 22: Representação da coleção de variáveis

Modelados os recursos dos coletores e variáveis estão cobertos os aspectos de configuração da aquisição de dados. Contudo os dados coletados propriamente ditos também devem ser expostos para os clientes. Os dados coletados estão diretamente associados as variáveis configuradas para aquisição, sendo portanto recursos subordinados a estas. A última leitura de uma variável pode ser modelada com um recurso, pois permite o acompanha-

```
{
        "nome":contagemDePeçasAprovadas,
        "coletor":"/coletor/1",
        "tipoDeDado": "booleano",
        "offset":22,
        "habilitada":true
        "últimaLeitura": "/variavel/1/ultimaLeitura"
        "histórico": "/variavel/1/historico"
 }
```

Figura 23: Representação de uma variável

mento do estado atual do processo supervisionado. Na tabela 5 está exposto o recurso de última leitura e sua URI.

Um cliente pode obter a última leitura da variável através do método GET. Outras operações (POST, PUT e DELETE) não são expostas, pois trata-se de um recurso somente para leitura, não fazendo sentido qualquer outra operação. Os clientes monitoram as variáveis através de sucessivas requisições GET, em uma técnica conhecida como varredura cíclica (”Polling”). Na figura 24 a representação da última leitura da variável contagem de peças aprovadas, que possui como informações um valor associado a uma estampilha de tempo.

Outro requisito típico é o histórico dos dados coletados. Este histórico pode possibilitar que clientes possam utilizar estes dados na montagem de um gráfico ou na plotagem de uma curva de tendência. Assim como a última leitura, o histórico está subordinado a variável monitorada. Alguns clientes possivelmente vão se interessar somente por um subconjunto do histórico, portanto torna-se importante definir um recurso que represente o histórico dentro de um determinado intervalo de tempo, especificado pelo cliente através de uma URI.

Da mesma forma que no recurso de última leitura, somente o método GET é exposto pelos recursos de histórico. A representação do recurso de histórico é uma coleção de leituras, cada uma contendo um valor, acompanhado do instante de tempo de sua coleta. Na figura 25 exemplos de uma representação do histórico da variável contagem de peças aprovadas em um determinado intervalo de tempo.

Muitas vezes pode ocorrer de não haver uma nova leitura para o cliente. Para evitar transferência desnecessária de informações, amenizando os problemas da varredura cíclica, pode-se utilizar o mecanismo de GET condicional. No GET condicional o cliente informa através do cabeçalho HTTP “If-Modified-Since” uma estampilha com o instante de tempo da última lei-

Recurso	URI
Última leitura de variável	/variavel/{id}/ultimaLeitura
Histórico de variável	/variavel/{id}/historico
Histórico de variável por intervalo	/variaveis/{id}/historico/{inicio},{fim}

Tabela 5: Recursos de dados coletados

```
{
        "instante": "31-07-2011 23:04:02",
        "valor": 30
}
```

Figura 24: Representação da última leitura de uma variável

tura que ele possui. O servidor, ao receber a requisição, analisa se houve uma nova leitura depois do instante informado pelo cliente, caso exista, uma representação da nova leitura é enviada, com o cabeçalho "Last-Modified" ajustado com o instante da última leitura. Caso não exista uma nova leitura, uma representação vazia é enviada como resposta, com o código HTTP 304, informando que o recurso não foi modificado. Ao receber essa resposta de recurso não modificado, o cliente pode utilizar as informações presentes em seu cache local. Este mecanismo de GET condicional também pode ser utilizado em outros recursos caso seja necessário.

Alarmes são conceitos comuns em SCADA, pois indicam a ocorrência de eventos inesperados no processo que está sendo monitorado. As condições de disparo dos alarmes estão associadas a variáveis do processo supervisionado, sendo expressas através de expressões lógico-matemáticas como igualdade e desigualdade de valores e mudança de estado. A existência de níveis de prioridade para os alarmes é importante, pois permite ao operador saber quais alarmes demandam uma maior atenção, por exemplo quando ocorre alguma situação crítica que necessita de um tratamento prioritário.

```
"leituras": [
        {"instante": "31-07-2011 22:12:02", "valor": 30}
        {"instante": "31-07-2011 22:08:02", "valor": 29}
        {"instante": "31-07-2011 22:04:02", "valor": 28}
]
```

Figura 25: Representação do histórico de uma variável

Assim como na aquisição de dados, um dos aspectos fundamentais quanto aos alarmes é a possibilidade de configurá-los para o processo que está sendo monitorado. Neste âmbito, deve ser possível visualizar e determinar quais são as condições alarmantes, ou seja, aquelas que disparam alarmes. Um recurso de coleção de todos os alarmes configurados para o processo pode ser criado com este propósito.

Também deve ser possível criar novos alarmes, por exemplo, o gerente de produção da fábrica deseja saber quando a contagem de peças retrabalhadas chegar a três, para que possa reclamar da qualidade das peças brutas. A criação de novos alarmes é realizada através de uma requisição POST na coleção de alarmes, onde o cliente especifica na representação enviada as condições de disparo, um texto descrevendo o alarme e a prioridade dos alarmes que serão disparados. Na figura 26 o exemplo de como seria a representação enviada pelo aplicativo cliente utilizado pelo gerente de produção.

```
{
        "nome": "númeroDePeçasEmRetrabalhoIgualATrês",
        "condição": "númeroDePeçasEmRetrabalho=3",
        "descrição": "três peças em retrabalho",
        "prioridade": "URGENTE",
}
```

Figura 26: Representação de um alarme enviado pelo cliente

Ao efetuar uma requisição GET no recurso da coleção dos alarmes, o cliente terá como resultado uma listagem das URIs de cada recurso de alarme que foi criado, podendo exibir esta listagem na interface gráfica por exemplo. A representação de uma coleção de alarmes contendo três alarmes pode ser vista na figura 27.

```
alarmes: [
        {"nome": "trêsEmRetrabalho", "uri": "/alarme/1"},
        {"nome": "mesaParouDeGirar", "uri": "/alarme/2"},
        {"nome": "peçaReprovada", "uri":"/alarme/3"},
]
```

Figura 27: Representação da coleção de alarmes

Cada recurso de alarme é identificado unicamente em sua URI por um número qualquer determinado pelo servidor. A representação do recurso de alarme reflete a representação enviada pelo cliente em sua criação, com

a adição da informação de quando o alarme foi criado, e de um enlace para o recurso que representa o disparo do alarme. Na figura 28 pode ser vista a representação do alarme criado no exemplo.

```
{
"nome": "númeroDePeçasEmRetrabalhoIgualATrês",
        "condição": "númeroDePeçasEmRetrabalho=3",
        "descrição": "peças em retrabalho chegaram a 3",
        "prioridade": "URGENTE",
        "criadoEm": "20-07-2011 22:00:00",
        "disparo": "/alarme/1/disparo",
}
```

Figura 28: Representação de um alarme recebido do servidor

Contudo, é preciso que exista um mecanismo para avisar os clientes do disparo dos alarmes. Tal comportamento pode ser modelado através de um recurso para o disparo do alarme (presente na representação do alarme). Ao efetuar uma requisição GET no recurso de disparo, o cliente fica aguardando o recebimento de uma resposta, assim que a condição estabelecida para o disparo do alarme se cumpre, o servidor envia como resposta para o cliente uma representação com dados como: o instante em que o alarme disparou, sua prioridade e a descrição do disparo. Esta técnica onde o servidor aguarda até que tenha alguma informação disponível para responder ao cliente é uma variante da varredura cíclica. Ela é conhecida como "Long Polling" e faz parte um conjunto de técnicas conhecidas como Comet (BOZDAG; MESBAH; DEURSEN, 2007), utilizadas para implementar esta dinâmica onde clientes HTTP esperam por notificações do servidor, sem que para isso precisem de um conector de servidor.

```
{
        "instante": "12-07-2011 12:12:01",
        "prioridade"": "URGENTE",
        "descrição": "três peças em retrabalho"
}
```

Figura 29: Representação do disparo de um alarme

Suponha que gerente de produção queira ser alarmado todas as vezes que uma peça é depositada no armazém de reprovadas. Peças podem ter sido depositadas no armazém de reprovadas no intervalo de tempo que inicia no instante em que o cliente recebeu a última resposta e termina no instante

em que o servidor recebe a próxima requisição. Este caso pode ser coberto através do uso dos cabeçalhos HTTP, o cliente indica o instante de tempo em que recebeu o último alarme através do cabeçalho “If-modified-since”, o servidor por sua vez, ao receber a requisição, verifica se disparou o alarme em que o cliente está interessado entre o instante informado pelo cliente o instante atual. Caso algum alarme tenha disparado neste intervalo, o instante do disparo mais recente é enviado na representação e no cabeçalho “Last-modified” da resposta. Ao receber a resposta o cliente utiliza o cabeçalho “Last-modified” nela presente para atualizar o cabeçalho “If-modified-since” para uma próxima requisição. Para informar ao cliente de todos os disparos ocorridos entre as requisições também é incluído um enlace para o recurso de histórico de disparos, presente na representação da figura de exemplo 30 através do atributo de enlace ”alarmesNãoObtidos”.

```
{
        "instante": "12-07-2011 12:12:01",
        "prioridade"": "URGENTE",
        "descriçãoDeDisparo": "peça reprovada",
        "alarmesNãoObtidos" :
        "/alarme/1/historico/22-07-2011 11:40:19,
        22-07-2011:11:42:06"
}
```

Figura 30: Representação do disparo de um alarme (quando ocorreram outros disparos)

O histórico dos disparos de alarme tem por finalidade registrar todos os disparos de um alarme em um dado intervalo de tempo. Este recurso pode interessar tanto a um cliente que queira saber quais disparos ocorreram entre duas requisições, como a um técnico que precisa fazer uma auditoria ou avaliar o desempenho da célula. Na figura 31 a representação do histórico de um alarme. Com este histórico dos disparos de alarmes estão cobertas as funcionalidades típicas de SCADA. A tabela 6 resume todos os recursos que foram projetados, mostrando suas URIs e quais métodos da interface uniforme são expostos para cada um deles.

5.1.3 Aspectos de Segurança e Confiabilidade

A segurança é uma propriedade indispensável para SCADA, dado o caráter estratégico que estes sistema possuem. A arquitetura orientada a recursos possui mecanismos para prover segurança para seus usuários, descritos

Recurso	URI	GET	POST	PUT	DELETE
Coleção de coletores	/coletores	X	X		
Coletor	/coletor/{id}	X		X	X
Coleção de variáveis	/variaveis	X	X		
Variável	/variaveis/{id}	X		X	X
Última leitura de uma variável	/variaveis/{id}/ultimaLeitura	X			
Histórico de uma varíavel	/variaveis/{id}/historico	X			
Histórico de uma varíavel em um intervalo	/variaveis/{id}/historico/{intervalo}	X			
Alarmes	/alarmes	X	X		
Alarme	/alarmes/{id}	X		X	X
Disparo de Alarme	/alarmes/{id}/disparo	X			
Histórico de disparos de um alarme	/alarmes/{id}/historico	X			
Histórico de disparos de um alarme em um intervalo	/alarmes/{id}/historico/{intervalo}	X			

Tabela 6: Resumo do recursos projetados

```
disparos: [
        { "instante": "22-07-2011 11:42:06,
        "prioridade": "INFORMATIVA"},
        { "instante": "22-07-2011 11:42:56,
        "prioridade": "INFORMATIVA"}
]
```

Figura 31: Representação do histórico de disparos de um alarme

em detalhes no capítulo 3.

Na arquitetura proposta optou-se por utilizar HTTPS (com assinatura digital) e HTTP Basic para prover autenticação, integridade e confidencialidade para os recursos projetados. Para a autorização foi desenvolvido um sistema de papéis para a aplicação da CFM, baseado no modelo RBAC (Role-based access control) (FERRAIOLO; CUGINI; KUHN, 1995).

No modelo RBAC são designados diferentes papéis para os usuários do sistema de uma organização, sendo que cada papel possui permissões distintas para realizar operações sobre os objetos do sistema. Em organizações industriais novos usuários podem ser contratados, demitidos e trocarem de função, desta forma o modelo RBAC traz a flexibilidade necessária para que as aplicações SCADA se adaptem as este ambiente dinâmico (GOLONKA; GONZALEZ-BERGES, 2009).

Os papéis utilizados na arquitetura baseiam-se em um modelo RBAC proposto para SCADA em (MAJDALAWIEH; PARISI-PRESICCE; SANDHU, 2007), adaptado para a granularidade dos recursos. Nesta adaptação os recursos são mapeados para os objetos do modelo RBAC, desta forma os papéis de usuários possuem permissões distintas em relação as operações possíveis nos recursos.

Foram definidos cinco papéis com diferentes permissões. O papel de operador está associado a visualização da IHM, portanto possui permissão de leitura a todos os recursos. O supervisor representa o papel de um operador com permissão de realizar ações de controle e configurar alarmes. Ao papel de técnico são atribuídas as operações de configuração. O usuário externo representa o papel de um sistema externo interagindo com os recursos de um SCADA, possuindo permissão de leitura a todos os recursos. O gerente possui autorização total para realizar qualquer operação em qualquer recurso. Esta hierarquia de papeis pode ser visualizada na figura 32.

Para que modelo acima funcione é necessário que o gerente do SCADA possa criar novos usuários, remover usuários existentes ou atualizar o papel de um usuário que trocou de função. Para isto foram projetados novos recursos. O recurso da coleção de usuários permite que o gerente crie novos

Recurso	GET	POST	PUT	DELETE
Usuários	X	X		
Um usuário	X		X	X

Tabela 7: Interface dos recursos de usuários

usuários através de uma requisição POST. O recurso de usuário permite que o administrador atualize o papel deste usuário ou o remova do sistema. Estes recursos estão descritos na tabela 7, juntamente com as operações que neles podem ser realizadas.

A tabela 8 sintetiza as permissões que cada papel possui em relação aos recursos projetados. Nela cada recurso é listado em uma linha, sendo que nas colunas são especificadas as operações que podem ser realizadas no recurso. Os papéis que podem realizar as operações são especificados nas células, através da inicial do nome do papel. Por exemplo em todas as células onde está presente a letra ”O” significa que a operação da coluna pode ser realizada no recurso especificado na linha pelo papel de Operador.

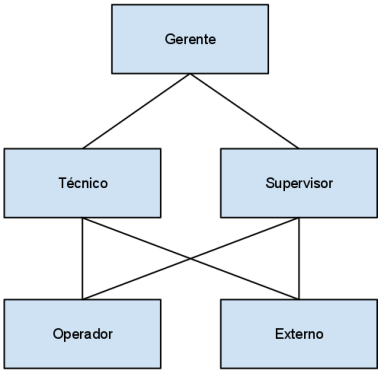

Figura 32: Hierárquia de papéis para aplicações típicas SCADA

A confiabilidade é muitas vezes ressaltada em SCADA devido ao uso destes sistemas em aplicações de missão crítica que requerem alta disponibilidade. Embora não tenha sido tratado na arquitetura projetada, não existe

Recurso	GET	POST	PUT	DELETE
Coleção de coletores	G,T,S,O,E	G,T		
Coletor	G,T,S,O,E		G,T	G,T
Variáveis do processo	G,T,S,O,E	G,T,S		
Uma variável	G,T,S,O,E		G,T,S	G,T,S
Última leitura de uma variável	G,T,S,O,E			
Histórico de uma variável	G,T,S,O,E			
Histórico de variável em um intervalo	G,T,S,O,E			
Coleção de alarmes	G,T,S,O,E	G,T,S,O		
Alarme	G,T,S,O,E		G,T,S,O	G,T,S,O
Disparo de alarme	G,T,S,O,E			
Histórico de disparos	G,T,S,O,E			
Histórico de disparos em um intervalo	G,T,S,O,E			
Usuários	G	G		
Usuário	G		G	G

Tabela 8: Operações autorizadas sobre os recursos

impedimento dentro dos princípios de arquitetura da Web de se construir um sistema que cumpra com este requisito de forma satisfatória. O Harc (do inglês, Highly-Available Robust Co-scheduler) por exemplo, permite a alocação de recursos em Grids através de um abordagem orientada a recursos(MACLAREN; KEOWN, 2006). Na Web, sistemas como o Google Analytics possuem APIs REST que utilizam o serviço Chubby, que através de Paxos fornece a confiabilidade necessária (CHANG et al., 2008).

5.2 Implementação

5.2.1 Aplicações

De forma a exemplificar as vantagens desta arquitetura, foram desenvolvidas duas aplicações clientes: uma aplicação que configura os recursos para a aquisição de dados da CFM e um sinótico para mostrar o estado atual da mesma. Tais aplicações mostram as capacidades de interação com outras aplicações que a arquitetura proposta possui, assim como seus aspectos de simplicidade e portabilidade.

A aplicação responsável pela configuração dos recursos da célula foi escrita na linguagem Java. Essa aplicação atua como um cliente, possuindo um conector HTTP que realiza requisições POST para o servidor que hos-

peda os recursos. As representações utilizadas nas requisições possuem as informações necessárias para configurar a aquisição de dados na célula.

O primeiro recurso criado é um recurso de coletor, contendo as configurações para que o CLP colete os dados através do protocolo Modbus RTU. Depois de criado o coletor são criados os recursos das variáveis do processo, que fornecem informações sobre a contagem de peças aprovadas, reprovadas ou enviadas para retrabalho e também sobre os equipamentos presentes na célula (garra, mesa, furadeira, solda e teste de qualidade). Ao final da execução do programa, todos os recursos foram criados, e o SCADA está configurado para coletar dados da célula, podendo iniciar seu funcionamento. Esta aplicação evidencia os aspectos de configuração da célula, que podem ser realizados por qualquer aplicação que possua um conector HTTP cliente. Outras aplicações poderiam ser utilizadas com este propósito, como qualquer navegador Web ou então uma ferramenta de linha de comando como o cURL. A figura 33 mostra como uma aplicação qualquer na Web pode configurar a CFM, onde cada requisição POST para criação dos recursos é atrelada a configuração de um dos dispositivos da célula.

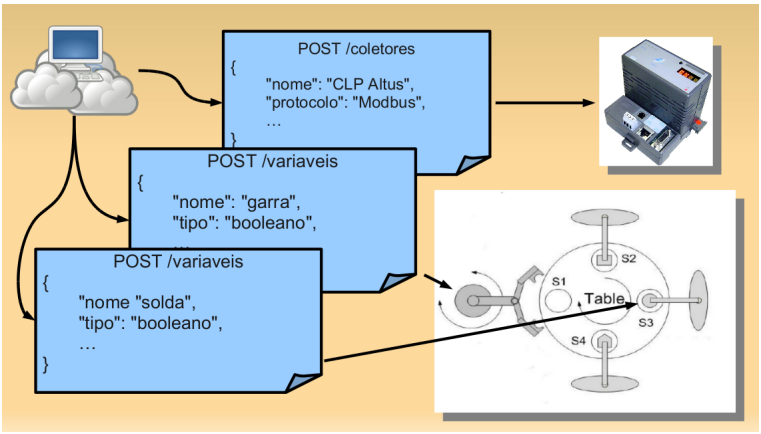

Figura 33: Configuração dos recursos para a CFM

A outra aplicação desenvolvida é um sinótico que exibe em uma interface gráfica o estado atual dos equipamentos da célula, mostrando se estão ligados ou desligados, permitindo acompanhar o estado atual do processo. A aplicação foi escrita na linguagem de programação Telis, desenvolvida pelo Edugraf. Telis é uma linguagem e um ambiente de programação de propósito educacional, direcionados para o aprendizado da programação, sendo utili-

zada na disciplina introdutória de programação do curso de Automação e Sistemas da UFSC (PIERI et al., 2009). Programas escritos em Telis, denominados apliques, podem ser executados em qualquer máquina que disponha de um navegador Web com a possibilidade de rodar ”applets”Java.

O sinótico é um aplique Telis que exibe uma foto da célula como imagem de fundo e figuras de pequenas lâmpadas sobre cada equipamento, conforme pode ser verificado na figura 34. Quando o equipamento está em atividade a sua lâmpada fica ligada, e desligada caso contrário. Para implementar este comportamento cada equipamento foi modelado como um ator, entidade computacional autônoma com memória e linha de execução própria. Cada ator verifica periodicamente se houve uma mudança de estado no recurso associado a variável que corresponde ao equipamento, fornecendo a resposta gráfica necessária. Os atores se comunicam com a célula através da primitiva ”obter”, realizando requisições GET na URI recursos de última leitura das variáveis. Do ponto de vista da performance, o sinótico desenvolvido pode acompanhar o estado da CFM de forma satisfatória.

O aspecto mais interessante do desenvolvimento desta aplicação foi a forma simples com o qual foi possível interagir com os recursos. Cada leitura de variável pode ser obtida através de uma requisição GET em sua URI. Esta abordagem permite que qualquer aplicação Web possa interagir com os recursos, dado que a operação GET é a mais básica operação do HTTP. Além disso, qualquer outro recurso na Web pode referenciar as informações da CFM através de enlaces, que podem ser seguidos por aplicações clientes, guardados em cache ou armazenados para interações futuras.

5.2.2 Implementação no Mango M2M

Foi realizada uma implementação parcial da arquitetura apresentada anteriormente, com o objetivo de possibilitar que as aplicações clientes interajam com os recursos. Para isto foi modificada a arquitetura de um software livre para SCADA.

A escolha de modificar um SCADA existente ao invés de desenvolver um novo teve uma motivação pragmática, que foi o aproveitamento de toda uma infraestrutura de software fornecida para a aquisição e armazenamento de dados. O aproveitamento desta infraestrutura possibilitou que o foco do trabalho fosse direcionado ao projeto dos recursos, evitando que questões de nível mais baixo de abstração (como a implementação da aquisição de dados via Modbus) se colocassem como problemas. Outra motivação foi estudar a arquitetura de software de um aplicativo SCADA existente e analisar as modificações necessárias para que os recursos pudessem ser implementados.

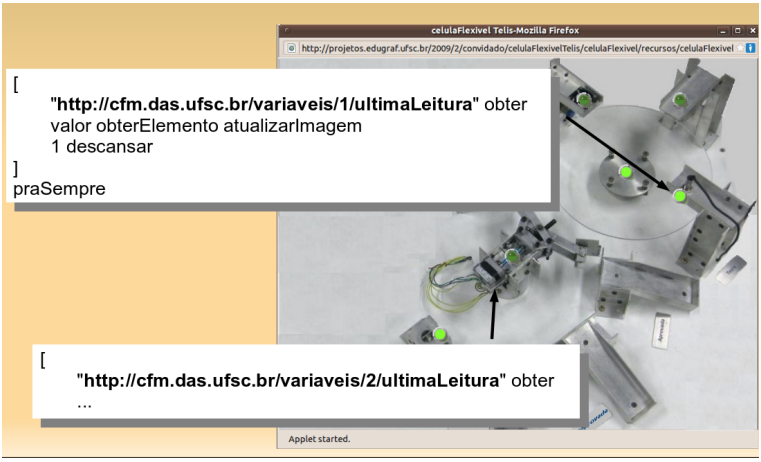

Figura 34: Sinótico da CFM desenvolvido em Telis

O software objeto desta implementação foi o Mango M2M. O Mango é definido como um software para "Machine-to-Machine (M2M), controle industrial, SCADA e domótica (automação residencial)" (SEROTONIN, 2011). No contexto de aplicações SCADA ele fornece fornece as funcionalidades típicas destes sistemas, tais como aquisição de dados de dispositivos através de protocolos de comunicação utilizados na automação industrial (Modbus, DPN3, OPC), históricos, alarmes (chamados de eventos), e uma interface gráfica dinâmica com sinóticos e gráficos.

O Mango é ambientado na Web e multiplataforma, sendo que todas as suas funcionalidades podem ser acessadas a partir de um navegador. Este fator influenciou na sua escolha como objeto de estudo, pois se trata de um SCADA moderno, que utiliza tecnologias da Web. Outro fator decisivo na sua escolha é o fato de ser software livre (licença GNU GPL versão 3.0), tendo seu código fonte aberto para modificações, tendo recebido diversas contribuições de indivíduos e empresas ao longo dos anos de atividade do projeto. O Mango possui todos os direitos reservados pela Serotonin Software, que durante o presente trabalho acabou por encerrar o projeto do software.

Para estudar a arquitetura de software do Mango foi necessário buscar documentação de referência. Embora não exista uma documentação oficial de sua arquitetura de software, foi possível encontrar em (JUHASZ, 2009) um documento onde são descritos alguns aspectos de sua arquitetura. Outras fontes de referência foram o sítio oficial do projeto (SEROTONIN, 2011)

e principalmente o próprio código fonte do software que é disponibilizado publicamente.

O Mango M2M possui uma arquitetura de software distribuída que segue o estilo cliente-servidor e utiliza o protocolo HTTP para comunicação. O cliente executa no navegador e tem suas funcionalidades extendidas através de código móvel JavaScript. O código do lado do servidor é implementado em Java através da tecnologia de Servlets, que tratam requisições HTTP. As Servlets podem executar em diferentes servidores Web, denominados Servlet Containers. No caso o Mango é ambientado no servidor Apache Tomcat. Devido aos fatores mencionados o Mango pode ser executado em sistemas que possuam uma máquina virtual Java compatível e os requisitos mínimos de hardware exigidos.

A interação entre cliente e servidor se dá através de RPC sobre HTTP. Os serviços definidos no servidor são nitidamente orientados para somente suprir as necessidades da interface gráfica utilizada pelo cliente. O processo se inicia com o cliente requisitando as páginas HTML da interface gráfica, sendo que cada funcionalidade (histórico, sinótico e alarmes) é apresentada em uma página diferente. As páginas possuem nela embutidas o código Java Script responsável por tratar ações do usuário e atualizar periodicamente os seus componentes gráficos em decorrência de mudanças de estado no servidor (a chegada de um novo alarme por exemplo). A atualização periódica se dá através de requisições HTTP assíncronas, utilizando a técnica do ”Long Polling”.

Para realizar a interação entre cliente e servidor o Mango utiliza a biblioteca DWR (Direct Web Remoting) (DWR, 2010). O DWR é usado para gerar código Java Script para realizar RPC no cliente (navegador) a partir de classes escritas em Java, no lado do servidor. Desta forma o DWR atua como uma ferramenta típica de aplicações RPC, gerando os stubs do lado do cliente e integrando-o aos serviços expostos por uma API ambientada no servidor. As chamadas de procedimento com informações como o nome do método invocado e seus parâmetros, assim como suas respostas, são codificadas através de Java Script e transportadas embutidas em requisições HTTP POST.

Em relação a segurança é mencionado ser possível configurá-lo para funcionar com SSL, criptografando a comunicação (SEROTONIN, 2011), embora nenhum exemplo deste uso tenha sido encontrado. O acesso ao sistema é protegido através de um esquema de autenticação e autorização próprios, implementado através sessões HTTP. Cada usuário que autentica no sistema inicia uma sessão, que é persistida no servidor. No servidor também são armazenadas informações relativas ao último estado do cliente, para que somente sejam enviadas as informações que ele ainda não possui no caso das atualizações periódicas. Desta forma fica evidente a manutenção do estado

da aplicação no servidor.

Através desta análise da arquitetura distribuída do Mango foi possível encontrar divergências em relação ao estilo da Web. Sua arquitetura viola espectos importantes, apesar de seguir um estilo cliente-servidor com HTTP. A interação através de RPC viola o príncipio da interface uniforme e abusa do método POST, que deve ser utilizado somente com propósitos especificos. Portanto, assim como foi analisado em detalhes no capítulo 4, o protocolo HTTP não é utilizado como um protocolo de aplicação, e sim como um protocolo de transporte. Outro aspecto está na manutenção de estado da aplicação no servidor, prejudicando a escalonabilidade e visibilidade das interações.

Descrita a arquitetura distribuída do Mango, e expostas as suas limitações, passa-se agora a uma análise de como essa arquitetura é implementada no componente do servidor. Essa análise tem como objetivo a implementação da arquitetura orientada a recursos através da modificação da arquitetura existente.

A arquitetura interna do Mango foi implementada a partir do paradigma orientado a objetos e está estruturada segundo o padrão de projetos MVC (do inglês, Model View Controller) (JUHASZ, 2009). O objetivo do MVC é separar as responsabilidades referentes a lógica de domínio das responsabilidades de apresentação das informações. No padrão MVC clássico existem três tipos de componentes cada qual com suas responsabilidades. O modelo é composto por objetos que encapsulam as informações sobre o domínio de aplicação. O papel da visão está na apresentação destas informações na interface com o usuário. Já o controle é responsável por tratar as ações efetuadas pelo usuário, manipulando o modelo e atualizando a visão (FOWLER, 2002). A figura 35 (FOWLER, 2002) ilustra os componentes presentes no MVC e como eles se relacionam.

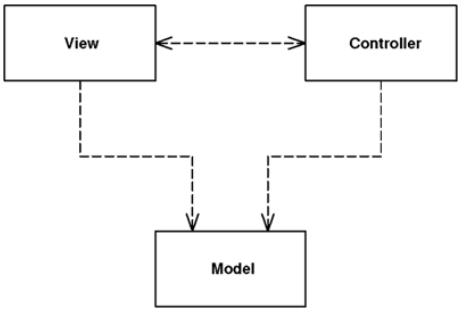

Figura 35: Padrão MVC

Um aspecto da separação de responsabilidades presentes no MVC é relação de dependência entre seus componentes. A visão depende do modelo mas não o inverso, o que implica que o modelo pode evoluir de forma independente. Uma das vantagens deste desacoplamento é que a visão pode ser alterada sem que o modelo o seja, outro aspecto é a possibilidade de utilização de múltiplas visões para um mesmo modelo. O MVC está presente no Mango através de um módulo do arcabouço de código aberto Spring, utilizado em aplicações Web (SPRING, 2010). Fazendo uma correspondência com o padrão MVC temos a visão do Mango como suas páginas web, que representam a interface com o usuário. Já os controladores são Servlets responsáveis por tratar as requisições HTTP efetuadas pelos clientes, escolhendo a visão que deve ser apresentada na resposta e preenchendo-a com dados oriundos do modelo. A figura 36 (SPRING, 2010) mostra como se dá esta interação entre os componentes MVC no arcabouço Spring.

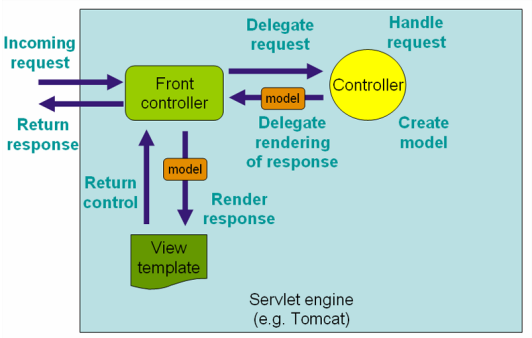

Figura 36: MVC no arcabouço Spring utilizado pelo Mango M2M

Quando uma requisição é efetuada pelo navegador ela é tratada pelo Apache Tomcat, que a direciona para a Servlet do Mango que atua como um Front Controller (FOWLER, 2002), implementando o padrão de mesmo nome. O Front Controller centraliza o processamento das requisições e as direciona para outros controladores (outras Servlets) e retornando a resposta para o cliente. Este redirecionamento é realizado de acordo com o padrão presente na URI. Por exemplo, uma requisição que contenha em sua URI o caminho ”/dwr”é delegada para o controlador responsável pelo processamento das chamadas RPC via DWR, enquanto que uma requisição cuja URI contêm o sufixo ”htm”é direcionada para o controlador do Spring, responsável pela obtenção das páginas HTML.

O controlador do Spring por sua vez também é um Front Controller,

delegando as requisições para outros controladores de acordo com a página solicitada. Cada um destes controladores instancía os objetos de domínio, obtendo as informações necessárias do modelo. Estas informações são encapsuladas em objetos e repassadas novamente para o Front Controller, que por sua vez o envia para o componente View Template, que irá escolher a visão requisitada e montar a página de acordo com as informações recebidas.

O modelo do Mango é estruturado em camadas. A camada do modelo do domínio é composta dos objetos que representam as entidades presentes no sistema. Abaixo da camada dos objetos do domínio está uma camada de acesso a dados que permite obter e persistir informações da base de dados relacional (Apache Derby) que o Mango utiliza. Camadas de acesso a dados em geral oferecem serviços de acesso a fontes de dados para seus clientes, abstraindo destes a forma como estes dados são acessados (FOWLER, 2002). Este desacoplamento permite que a forma de acesso aos dados possa ser alterada sem que a lógica do domínio da aplicação o seja.

A camada de acesso a dados do Mango é composta por objetos que implementam o padrão de projeto DAO (do inglês, Data Access Object). Um DAO é um objeto que possui uma interface de acesso a dados, tipicamente oferecendo serviços de criação, consulta, atualização e remoção (CRUD) dos objetos do domínio (ALUR et al., 2003). No Mango os DAOs são utilizados em conjunto com o padrão DTO (Data Transfer Objects, na sigla em inglês). DTOs são objetos utilizados para transferir dados entre sistemas ou subsistemas. Os DTOs do Mango são objetos serializáveis que encapsulam informações referentes a objetos de domínio, e são utilizados em todas as camadas do sistema. Na figura 37 é exposta a arquitetura de software interna ao componente servidor do Mango.

Encerrada a análise da arquitetura interna do componente servidor do Mango passa-se a descrição das modificações realizadas para implementar os recursos. O que interessa nesta implementação é a infraestrutura de software que o Mango provê para aquisição e armazenamento de dados. Desta forma as partes de sua arquitetura que estão relacionadas a interface gráfica não interessam a esta implementação. Outra parte que pode ser descartada é o módulo para comunicação via RPC (DWR) por se tratar de uma abordagem de arquitetura distribuída que difere da arquitetura orientada a recursos projetada.

Isolando as partes desnecessárias resta um núcleo que contém toda a infraestrutura de software para aquisição e armazenamento dos dados. Este núcleo é composto pela parte do modelo, no contexto do padrão MVC e inclui as camadas do modelo do domínio e acesso aos dados. Por cima destas camadas foi implementada uma camada de recursos, especificada pelo projeto descrito anteriormente. A figura 38 ilustra a arquitetura modificada, com

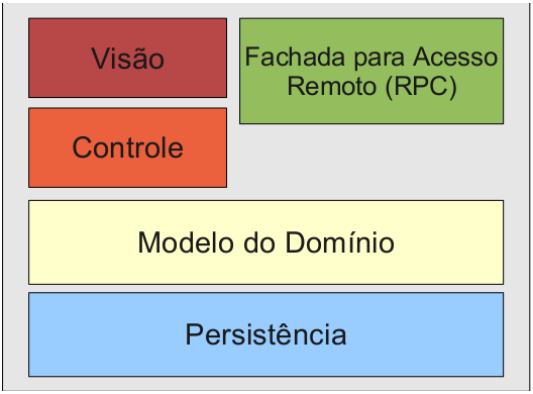

Figura 37: Arquitetura de Software do Mango M2M

a camada de recursos sobre as outras.

Figura 38: Arquitetura de software do Mango M2M modificada

No modelo do domínio estão todas as classes que representam conceitos importantes do sistema. No caso alguns conceitos são fundamentais para a aquisição de dados: os Data Sources e Data Points. Os Data Sources representam as fontes de informação, sendo responsáveis por adquirir dados do processo. Existem diferentes tipos de Data Sources, cada um deles associado a um protocolo específico de comunicação, possuindo diferentes parâmetros de configuração. Os Data Points por sua vez estão relacionados a variáveis do processo que deseja-se monitorar. Cada Data Point está associado a configurações específicas do protocolo de seu Data Source, que indicam como a informação pode ser encontrada nos dados adquiridos. Entre os tipos

de Data Sources e Data Points estão os para comunicação Modbus Serial, que são utilizados na comunicação com o CLP da célula.

Deste modo, pode-se aproveitar as classes do modelo do domínio associadas aos Data Sources e Data Points para coletar dados do processo e fornece-los aos recursos. Em um processo inverso, dados também são aceitos pelos recursos e repassados para a camada de acesso aos dados, para persistência. Portanto uma transformação dos dados tornou-se necessária, seja do modelo do Mango para as representações dos recursos ou então vice versa.

As requisições para os recursos são tratadas por uma nova Servlet. Quando uma requisição possui o prefixo "/recursos"em sua URI ela é delegada pelo Front Controller do Mango para esta Servlet. A Servlet também atua como um Front Controller, escolhendo a classe responsável por tratar a requisição de acordo com o recurso solicitado. Através desta abordagem foi possível "ativar" os recursos em uma instalação existente do Mango sem grandes esforços, incluindo a nova Servlet e seu mapeamento de URI na configuração do mesmo. Outro resultado alcançado foi o de ter a implementação da ROA funcionando em paralelo com o Mango de forma harmoniosa.

5.3 Resultados

Os recursos projetados cobriram os requisitos funcionais típicos de aplicações SCADA. A aquisição de dados foi proporcionada pelos recursos de coletores e variáveis do processo. As funções de histórico foram cobertas pelos recursos relacionados ao histórico de variáveis coletadas e alarmes. A configuração de dispositivos, variáveis e alarmes se deu através da possibilidade de criar, modificar e remover estes recursos. O controle em nível de supervisão foi associado a modificação do estado dos recursos de variáveis. Já a IHM foi delegada aos clientes, dentro do princípio da separação de responsabilidades.

O recurso do disparo do alarme possibilitou que clientes possam ser notificados da ocorrência de situações incomuns no processo. É necessário observar que a técnica do "Long Polling" empregada necessita que uma conexão HTTP persistente seja mantida enquanto o alarme não for disparado. Os navegadores possuem um limite configurável no número de conexões que podem manter com cada domínio, portanto pode ser necessário que este número de conexões seja aumentado dependendo do número de alarmes que o cliente deseja esperar que disparem. Dependendo da forma como são gerenciadas estas conexões persistentes pelo servidor este pode se tornar um problema de escala. Esta abordagem para os alarmes possui a limitação característica do "polling", que é a defesagem entre a ocorrência do evento, e

seu conhecimento por parte dos interessados (BOZDAG; MESBAH; DEURSEN, 2007). Estilos arquiteturais baseados em eventos, como o Publish-Subscribe (EUGSTER et al., 2003) possivelmente se adaptem melhor a natureza dos alarmes, ficando seu estudo como uma ideia para trabalhos futuros.

A abordagem para possibilitar a aquisição de dados da estação remota (CLP) foi encapsular a comunicação com o mesma na estação mestre. A estação mestre atuou como um *gateway* para se comunicar com a estação remota. Espera-se que seja possível aplicar a mesma abordagem em outros aplicações de SCADA, dado que as estações remotas geralmente possuem recursos de hardware limitados e utilizam protocolos industriais incompatíveis com a Web.

Os requisitos não funcionais típicos de SCADA são contemplados pela arquitetura. A interoperabilidade é favorecida através do uso de tecnologias abertas da Web, como o protocolo HTTP, URI e tipos de mídia. O princípio da interface uniforme também favorece esta propriedade no contexto da Web, pois possibilita que aplicações interajam com a arquitetura de uma forma padronizada e condizente com a especificação do HTTP. A possibilidade de existência de diversos tipos de mídia nas representações dos recursos também traz flexibilidade, no sentido de permitir que as necessidades de diferentes clientes possam ser satisfeitas.

A arquitetura é portável, pois pode ser executada em diferentes ambientes de hardware e software, sob a condição de que suportem as tecnologias da Web. Este requisito foi evidenciado na implementação que utiliza um ambiente de execução multiplataforma.

A escalonabilidade é favorecida pelo fato de a arquitetura estar embasada nos estilos arquiteturais que fundamentam a Web (cliente com cache, servidor sem-estado). Caches compartilhadas, previstas pelo estilo de sistemas em camadas também podem ser facilmente incorporados a arquitetura, fator que também contribui para a escalonabilidade.

Através da implementação foram desenvolvidas aplicações clientes para interagir com os recursos. Estas aplicações possibilitaram a configuração do CLP utilizado para aquisição de dados e a exibição de um sinótico em um navegador. Os experimentos evidenciaram a interoperabilidade, portabilidade e simplicidade da arquitetura projetada.

A segurança, mencionada frequentemente na literatura sobre SCADA, foi coberta através do HTTPS, combinado com autenticação HTTP Basic e do modelo RBAC para autorização. Uma limitação da abordagem para segurança é que o HTTPS garante somente a comunicação segura ponto a ponto entre os componentes. Este fato trás algumas implicações dada a natureza em camadas da Web. Um cliente que quiser se comunicar com um servidor e tiver que passar por componentes intermediários, não tem outra al-

ternativa a não ser confiar nos intermediários. Isto pode causar problemas, por exemplo um *proxy* malicioso poderia ao receber uma requisição via conexão segura, repassá-la para um componente não autorizado.

Mecanismos de tolerância a faltas para garantir a confiabilidade da arquitetura não foram implementados, devido ao escopo deste trabalho. Contudo foi possível mostrar através de algumas referências bibliográficas que este requisito pode ser cumprido dentro dos princípios arquiteturais da Web. Com relação ao tempo-real, a arquitetura projetada não é apropriada para processos que apresentam este requisito devido a natureza não-determinística da Internet.

A implementação contribuiu para mostrar como é estruturada a arquitetura de software de um SCADA existente para a Web. O Mango M2M, aplicativo objeto do estudo, possui escassa documentação no que se refere a sua arquitetura, portanto a análise que foi realizada pode ser utilizada por desenvolvedores interessados no sistema. Outro ponto que pode ser destacado, foi a abordagem de realizar modificações na arquitetura do Mango para que os recursos pudessem ser implementados, aproveitando toda a infraestrutura fornecida para a aquisição e armazenamento dos dados.

5.4 Conclusão do Capítulo

Neste capítulo foi apresentada a proposta de ROA para aplicações típicas de SCADA, mostrando como os requisitos destas aplicações podem ser atendidos dentro dos princípios arquiteturais que fundamentam a Web. Primeiramente foi exposta uma visão geral da arquitetura, onde foram identificados os componentes presentes nas aplicações SCADA e como estes podem ser mapeados para os componentes encontrados na Web. Em seguida foi realizado o projeto dos recursos, com base na metodologia ROA, mostrando como as funcionalidades típicas de SCADA são cumpridas. Com o objetivo de expor algumas propriedades foi realizada uma implementação, que possibilitou que fossem desenvolvidas aplicações que interagem com os recursos projetados. No final do capítulo foram apresentados alguns dos resultados extraídos da arquitetura. No capítulo seguinte a arquitetura projetada será comparada com uma arquitetura existente, buscando mostrar algumas contribuições e limitações do trabalho.

6 Estudo Comparativo de Arquiteturas para uma Aplicação SCADA na Web

Neste capítulo é feito um estudo comparativo entre duas arquiteturas de software para uma aplicação SCADA na Web. Primeiramente é apresentada a aplicação SCADA, o processo de transferência de um líquido entre tanques. Em seguida são descritas duas arquiteturas que modelam esta aplicação, uma baseada em Web Services RPC e a arquitetura ROA descrita no capítulo anterior. As duas abordagens são então analisadas e comparadas com base nos requisitos típicos de aplicações SCADA e no quão integradas elas são com a Web. O capítulo é concluído com os resultados desta análise.

6.1 A Aplicação SCADA dos Tanques

Uma análise do uso combinado das tecnologias Java, XML para favorecer requisitos de portabilidade, performance e interoperabilidade de sistemas SCADA na Web é feita em (FAN; CHEDED; TOKER, 2005). Nesse trabalho é apresentado um exemplo de aplicação SCADA integrada com a Web, o processo de transferência de um líquido entre tanques.

A transferência acontece de uma planta de origem para uma planta de destino. A planta de origem possui um tanque e uma bomba, que é utilizada para retirar o liquido deste tanque enviando-o através de um duto para a planta de destino. A planta de destino possui um tanque de compensação para lidar com variações de pressão e um tanque de recepção, onde o liquido é depositado. A central de transferência é responsável por realizar pedidos de transferência e pela supervisão de todo processo. As plantas de envio e recebimento gerenciam os pedidos, e realizam a transferência através do controle de seus equipamentos (válvulas e bombas). O processo é supervisionado por um operador em uma central de transferências e por operadores nas plantas de origem e recepção.

Cada operador possui um papel, exercendo diferentes funcionalidades conforme pode ser observado no diagrama de casos de uso da figura 39 (FAN; CHEDED; TOKER, 2005). O operador na central de transferências pode criar pedidos de transferência com um determinado volume e monitorar o processo. O operador da planta de envio pode iniciar, supervisionar ou parar um envio assim que o volume requisitado tiver sido transferido. Ao operador na central de recebimento cabe preparar a planta para o recebimento, iniciar este processo, supervisioná-lo e encerrá-lo.

A interação entre os operadores e as plantas de envio e recebimento é típica de aplicações SCADA. Nessa interação os operadores monitoram o estado dos equipamentos (válvulas, tanques e bomba) e efetuam ações de controle sobre eles para coordenar a transferência entre os tanques. O papel do operador na central de transferência é o de criar pedidos de transferência, uma funcionalidade típica de aplicações MES (MESA, 1997), pois trata-se de uma ação de planejamento que será executada pelos operadores do SCADA nas plantas.

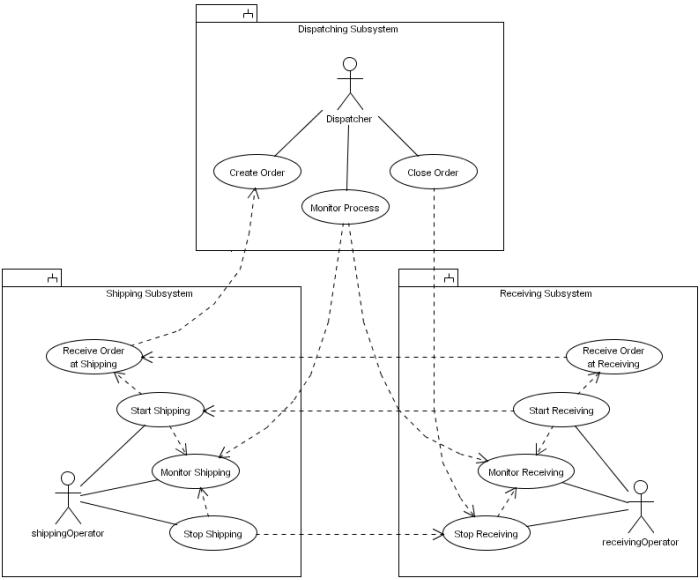

Figura 39: Caso de uso envolvendo os operadores das plantas

6.2 Arquitetura de Web Services Baseada em RPC

Na arquitetura proposta em (FAN; CHEDED; TOKER, 2005) a aplicação dos tanques é dividida em componentes de transferência, envio e recepção. Os componentes estão localizados em diferentes nós de uma rede conforme ilustrado na figura 40 (FAN; CHEDED; TOKER, 2004). A interação entre os componentes acontece através da troca de mensagens utilizando o protocolo

HTTP. Também é possível verificar a partir da figura que os componentes de envio e recebimento interagem localmente com os equipamentos em suas respectivas plantas, implementando a transferência do líquido entre os tanques.

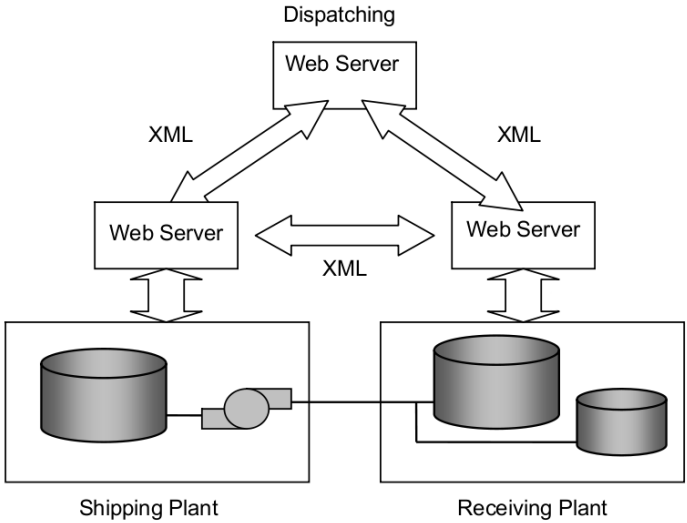

Figura 40: Interação entre plantas e central

Os componentes são organizados no estilo cliente-servidor, onde três clientes se comunicam com três servidores, correspondentes a central de transferência e as plantas de envio e recebimento. A interação entre clientes e servidores se dá através de conectores RPC, que trocam mensagens SOAP transportadas através do protocolo HTTP. A IHM é apresentada nos navegadores dos operadores através de código móvel (Applets Java) baixado dos servidores. A figura 41 ilustra os componentes e conectores da arquitetura. Analisando as restrições arquiteturais empregadas podemos classificar esta arquitetura como possuindo um estilo cliente-servidor sem-estado com código móvel.

O diagrama de componentes da figura 42 (FAN; CHEDED; TOKER, 2004) apresenta um nível mais baixo de abstração da arquitetura, mostrando como os componentes são implementados. Os componentes Dispatching, Shipping e Receiving representam respectivamente os componentes de transferência, envio e recepção. Os componentes DispatchingBusinessWebService, ShippingBusinessWebService e ReceivingBusinessWebService são responsáveis pelos envio e recebimento dos pedidos de transferência. O com-

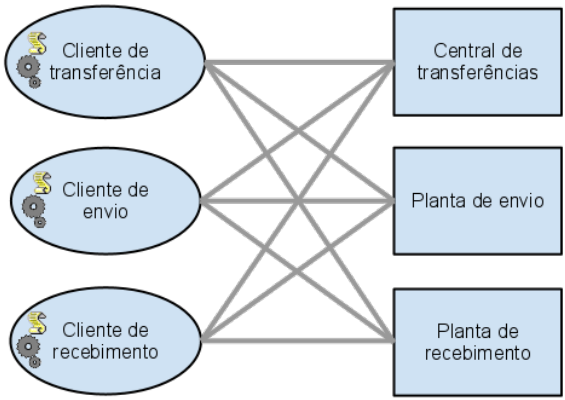

Figura 41: Arquitetura em nível de rede para o problema dos tanques

ponente DispatchingControlWebService é responsável pelo monitoramento do processo de transferência na central. Já os componentes ShippingControlWebService e ReceivingControlWebService realizam o processo envio e recebimento respectivamente, efetuando o controle dos equipamentos (bomba e válvulas dos tanques).

Nesta interação baseada em RPC cada componente possui dois conectores, um cliente e um servidor, podendo receber chamadas remotas de procedimento e também realizá-las. Observando o componente ShippingControlWebService é possivel visualizar como se dá esta comunicação com os demais componentes. Para se comunicar com os componentes DispatchingControlWebService e ReceivingControlWebService ele utiliza dois conectores clientes, o ProxyReceivingControlWS e o ProxyShippingControlWS, que realizam as chamadas de procedimento remotas para estes componentes. Para receber as chamadas ele utiliza um conector de servidor, chamado de InterfaceDispatchingControlWS. Estes conectores são os stubs que provêm a implementação necessária para a comunicação através da rede e interpretação das mensagens SOAP.

A modelagem orientada a objetos do sistema pode ser vista no diagrama de classes da figura 43 (FAN; CHEDED; TOKER, 2004). Os componentes DispatchingControlWebService, ShippingControlWebService e ReceivingControlWebService são objetos da classe ControlWebService. Para supervisionar e controlar os equipamentos eles trocam mensagens locais com objetos das classes Pump e Tank, que fazem interface com a bomba e o tanque

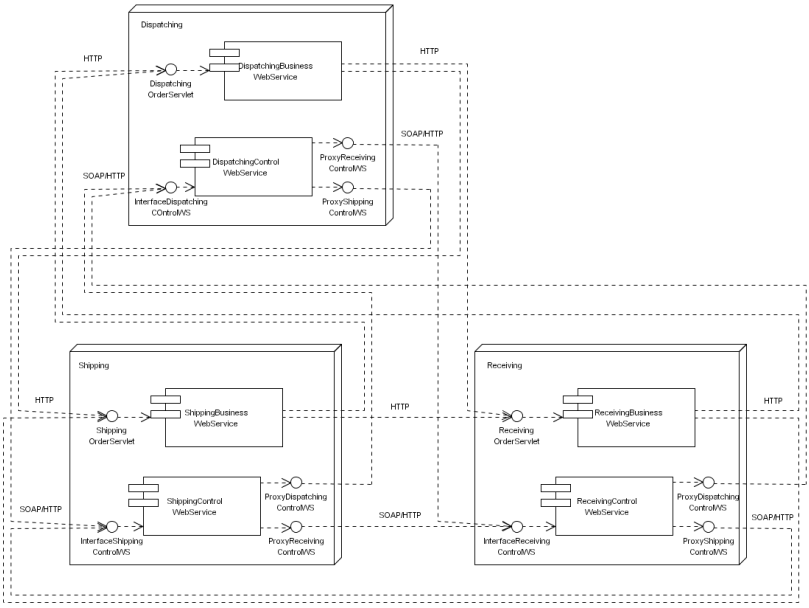

Figura 42: Diagrama de componentes da arquitetura

respectivamente. O processo de monitoramento nas plantas acontece através da varredura cíclica do estado dos equipamentos (mensagens para os objetos das classes pump e tanque) que é realizada pelos objetos da classe ControlWebService. Assim que ocorrem atualizações estes objetos avisam o navegador através de chamadas remotas do procedimento updateHMI.

6.3 Arquitetura Orientada a Recursos para Aplicações Típicas de SCADA

Passa-se agora para a modelagem da aplicação dos tanques utilizando a arquitetura ROA para aplicações típicas de SCADA descrita no capítulo anterior. Nesta linha a arquitetura da aplicação dos tanques pode ser definida como uma configuração específica da arquitetura geral para aplicações típicas de SCADA da figura 18, ou seja, um arranjo de seus componente, conectores e dados.

Na arquitetura as plantas podem assumir o papel de servidores de origem, que expõe recursos com os quais os operadores interagem através dos

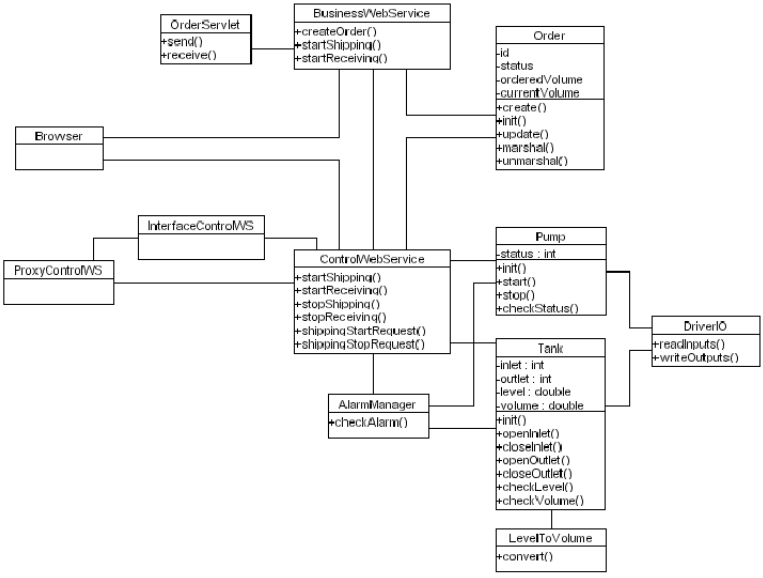

Figura 43: Diagrama de classes do sistema dos tanques

seus agentes de usuário (navegadores). A aplicação MES que gerencia os pedidos de transferência é incluída como outro agente de usuário interessado nos recursos oferecidos pelas plantas. Como último detalhe, para favorecer a escalonabilidade e performance do sistema é incluída uma cache compartilhada entre os clientes. A figura 44 ilustra a arquitetura proposta.

Definidos os componentes e conectores, passa-se agora para a seleção dos recursos que irão cobrir as funcionalidades da aplicação. Os dispositivos de campo utilizados para adquirir dados e controlar as válvulas e a bomba foram modelados como coletores, o que permite que suas configurações de aquisição possam ser lidas ou alteradas através da Web. Os equipamentos nas plantas (válvulas e bomba) foram modelados como variáveis do processo, o que torna possível a realização de operações de controle sobre os equipamentos, como abrir e fechar uma válvula ou ativar e desativar o funcionamento da bomba. As informações do nível e volume dos tanques também são modeladas como variáveis do processo, desta forma é possível consultar a última leitura e histórico destas informações. O mapeamento entre os equipamentos

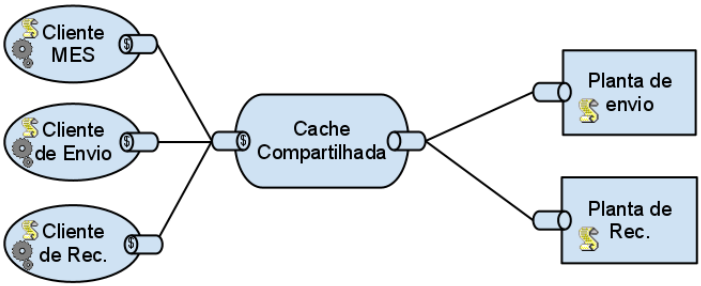

Figura 44: Arquitetura ROA para a aplicação dos tanques

Recurso	URI
Dispositivo de aquisição da planta de envio	/coletor/rtu
Volume do tanque de envio	/variavel/volumeDoTanque
Nível do tanque de envio	/variavel/nivelDoTanque
Válvula de saída	/variavel/valvulaDeSaida
Bomba	/variavel/bomba

Tabela 9: Recursos da planta de envio

das plantas de envio e recebimento e os recursos pode ser visto nas tabelas 9 e 10.

O processo de criação dos recursos dos coletores e das variáveis para esta aplicação é realizado pelos navegadores dos operadores das plantas, que realizam requisições POST para os servidores que hospedam os recursos. As representações utilizadas nas requisições possuem as informações necessárias para criação dos recursos. Os primeiros recursos criados são os dos coletores, depois são criados os recursos das variáveis para os equipamentos e dados sobre os tanques. A sequência de operações para criação dos recursos pode ser vista na figura 45.

A interação entre as plantas e a central de transferências se dá através dos pedidos de transferências. Um operador na central de transferências pode criar um pedido de transferência, solicitando que um volume seja transferido da planta de envio para a planta de recebimento. O operador pode monitorar o estado deste pedido para saber quando o mesmo foi concluído. Os operadores das plantas por sua vez monitoram o último pedido feito na central de trans-

Recurso	URI
RTU da planta de recebimento	/coletor/rtu
Volume do tanque de recebimento	/variavel/volumeDoTanque1
Nível do tanque de recebimento	/variavel/nivelDoTanque1
Válvula de entrada do tanque de recebimento	/variavel/valvulaDeEntradaDoTanque1
Volume do tanque de compensação	/variavel/volumeDoTanque2
Nível do tanque de compensação	/variavel/nivelDoTanque2
Válvula de entrada do tanque de compensação	/variavel/valvulaDeEntradaDoTanque2

Tabela 10: Recursos da planta de recebimento

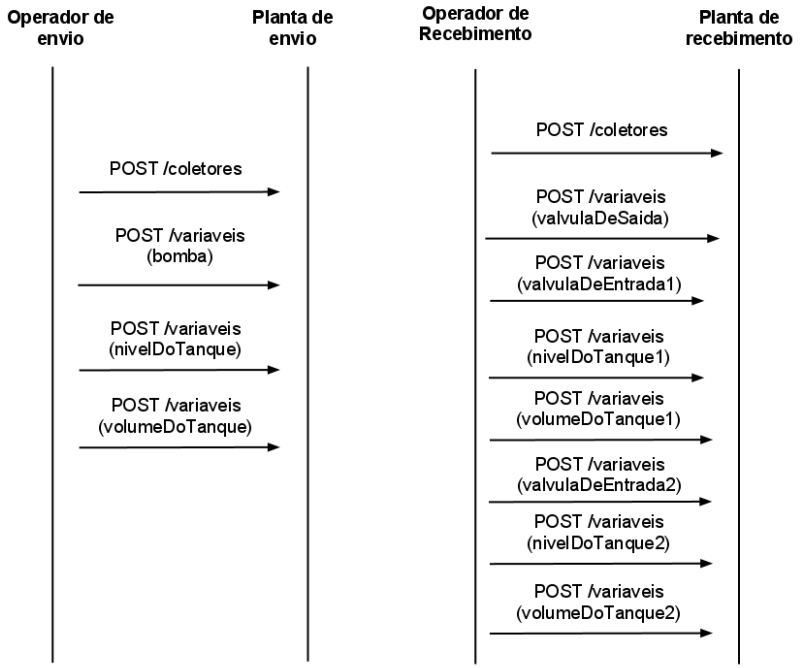

Figura 45: Sequência de operações nos recursos para configuração do processo

ferência, e quando existe um novo pedido interagem para que a transferência seja realizada.

A arquitetura ROA para aplicações típicas de SCADA não possui nenhum recurso que se encaixe com o conceito dos pedidos de transferência, como exposto anteriormente este conceito esta relacionado com aplicações típicas MES, e portanto fora do escopo deste trabalho. Entretanto para demonstrar como se dá uma integração MES/SCADA nesta arquitetura iremos projetar um recurso que serve como ponte entre os dois sistemas. Para isto foi projetado o recurso que representa um pedido de transferência feito pelo cliente MES para o SCADA. Este pedido pode ser criado através de uma requisição PUT, pois o cliente tem o controle da URI onde o recurso é criado. A figura 46 ilustra como o cliente MES pode criar o pedido de transferência enviando as requisições PUT para os servidores localizados nas plantas.

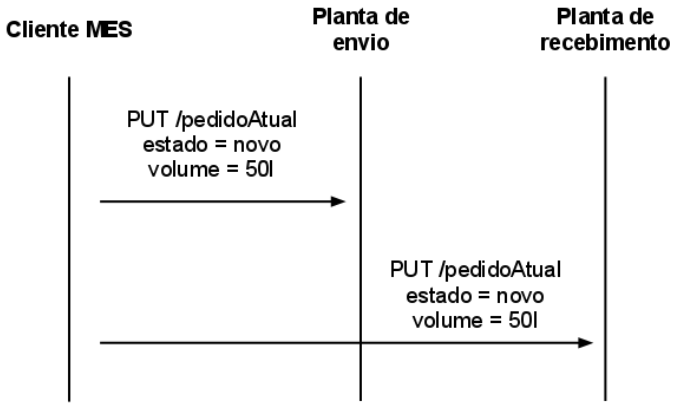

Figura 46: Criação de pedido de transferência solicitada pelo cliente MES

Depois de criado, um pedido pode ter o seu estado atualizado. Somente uma transferência pode ocorrer ao mesmo tempo, por isto qualquer tentativa de criar um novo pedido enquanto o pedido atual ainda não foi concluído ocasionará em um erro. O recurso do pedido atual, sua URI e as operações que podem ser realizadas nele podem ser vistos na tabela 11. A representação do recurso do pedido de transferência possui como informações o instante de sua criação, o volume requisitado para transferência, um estado (novo, em andamento, parado e completo) e o instante da última transição de

Recurso	URI	GET	POST	PUT	DELETE
Pedido de transferência atual	/pedidoAtual	X		X	

Tabela 11: Recurso de pedido do transferência atual

estado, conforme pode ser visto na figura 47.

```
{
 "iniciado": "31-07-2011 23:04:02",
 "volumeRequisitado": "50",
 "estado": novo,
 "ultimaAlteração": "31-07-2011 23:04:02",
}
```

Figura 47: Representação do pedido de transferência atual

Os clientes das plantas e da central monitoram o estado do pedido de transferência atual e dos equipamentos através de sucessivas requisições GET nos recursos associados. As informações obtidas das representações dos recursos podem ser utilizadas para montar um sinótico, que reflete o estado atual da transferência para os operadores. O processo de monitoramento, incluindo uma sequência de operações GET nos recursos é exibido na figura 48. A possibilidade de uso de código móvel (Applets ou Javascript) permite a construção de interfaces gráficas dinâmicas através do uso de requisições HTTP assíncronas.

O processo de transferência ocorre da seguinte forma: os operadores nas plantas monitoram o recurso do pedido de transferência atual, verificando se este é um novo pedido, se sim, eles iniciam o processo de transferência. Quando o operador na planta de recebimento descobre o novo pedido, ele abre a válvula de entrada do seu tanque de recebimento, atualiza o estado do pedido de transferência sinalizando que o mesmo está em andamento e cria um alarme que irá disparar quando o operador de envio fechar sua válvula, ou seja quando a transferência for finalizada. Ao descobrir que a válvula na planta de recebimento está aberta o operador na planta de envio obtêm a última leitura do volume do tanque da planta de recebimento, criando um alarme que irá disparar quando o volume do tanque da planta de recebimento atingir o esperado, notificando-o da necessidade de encerrar a transferência. Depois de criar o alarme ele abre a válvula de saída do seu tanque, inicia o funcionamento da bomba e atualiza o estado do pedido de transferência, sinalizando o seu andamento. A figura 49 ilustra como se dá este processo

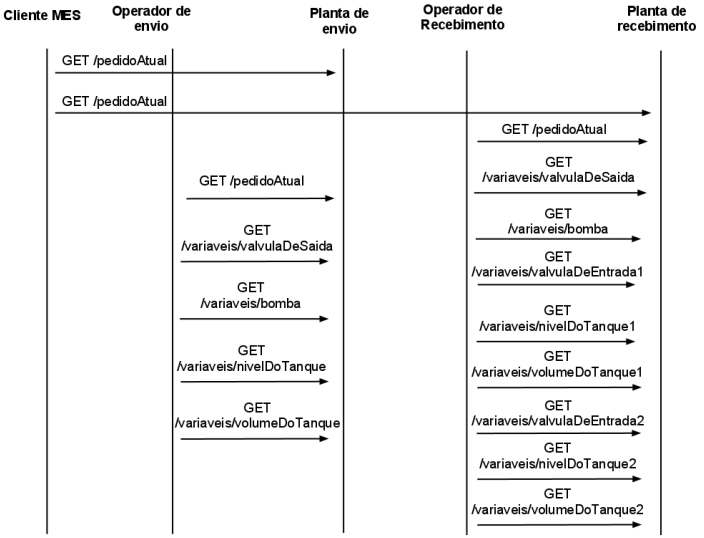

Figura 48: Sequência de operações nos recursos para monitoramento de uma transferência

através da sequência de operações nos recursos.

A transferência pode ser parada pelo operador na planta de envio. Para parar uma transferência o operador na planta de envio sinaliza o fechamento da válvula de saída do tanque e a interrupção do funcionamento da bomba. Esta sequência de operações é ilustrada pela figura 50.

Quando o volume requisitado no pedido de transferência é atingido, o operador na planta de envio deve parar de transferir. Esta situação ocorre através do disparo do alarme criado pelo operador da planta de envio, que sinaliza que o tanque na planta de recebimento está com o volume solicitado. Ao receber este alarme o operador na planta de envio solicita que a transferência pare, e ao receber a confirmação de que a transferência parou o operador na central de recebimento pode dar a transferência como encerrada, fechando a válvula de entrada do tanque e atualizando o pedido atual, marcando-o como completo. Esta sequência de operações é ilustrada pela figura 51.

Finalizada a descrição de como os recursos projetados cobrem os requisitos funcionais da aplicação do tanques, passa-se agora para uma análise comparativa das arquiteturas. Esta análise busca explicitar as diferenças arquiteturais entre elas, mostrando o impacto destas diferenças em relação aos

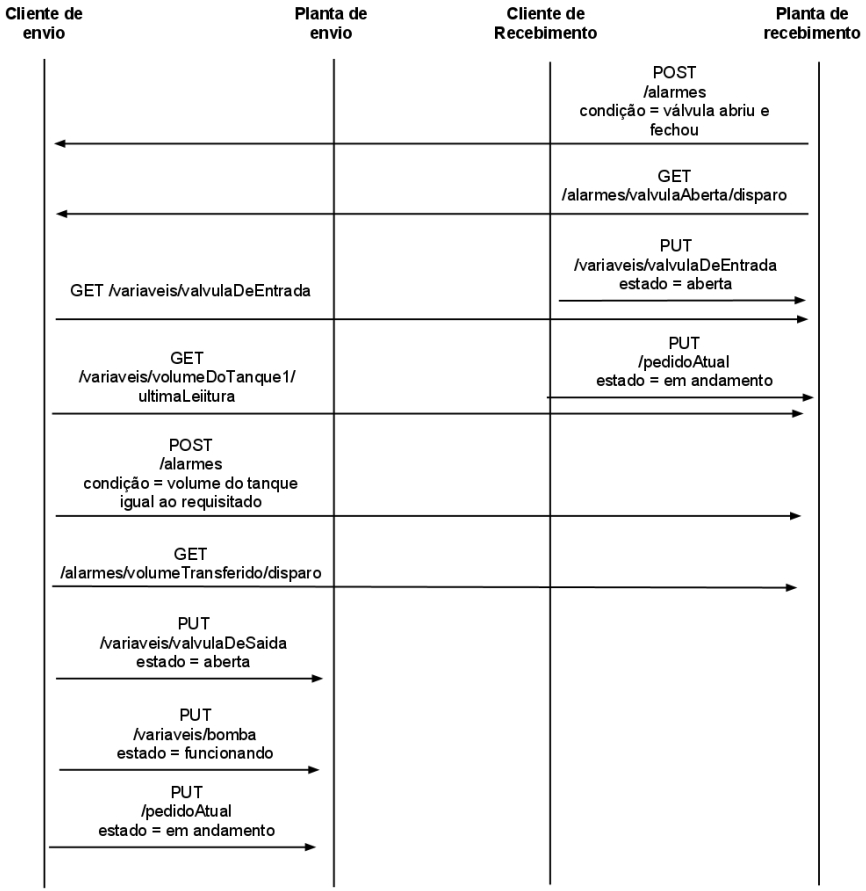

Figura 49: Sequência de operações nos recursos para realizar uma transferência

requisitos típicos de SCADA e integração com a Web. A fundamentação desta análise está presente no capítulo 4 onde são analisados os aspectos de integração com a Web das arquiteturas baseadas em RPC.

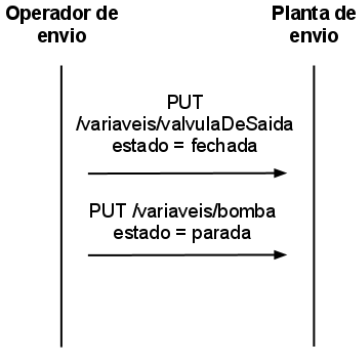

Figura 50: Sequência de operações nos recursos para parar uma transferência

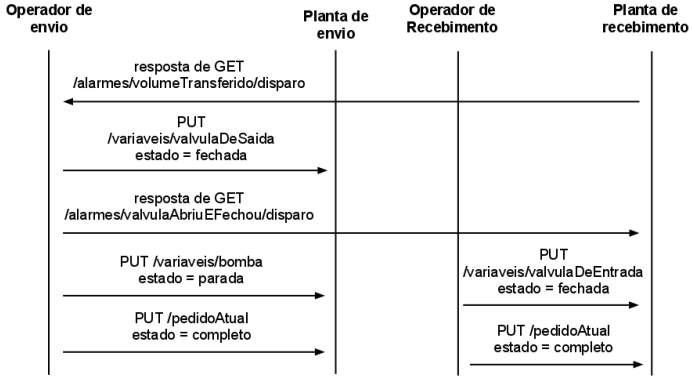

Figura 51: Sequência de operações nos recursos para finalizar uma transferência

6.4 Análise Comparativa das Arquiteturas

As arquiteturas apresentadas possuem algumas semelhanças, pois compartilham um estilo arquitetural cliente-servidor sem-estado com código móvel. Ambas se comunicam através do HTTP e utilizam URI para identificar os

destinos das requisições (sejam serviços ou recursos). Entretanto a arquitetura ROA difere da arquitetura de Web Services RPC pois possui interface uniforme, cache e sistema em camadas.

A restrição da interface uniforme torna as duas abordagens diferentes em termos de modelagem. Na abordagem de Web Services RPC cada componente servidor expõe uma série de procedimentos (startShipping, stopShipping, startReceiving...) que podem ser invocados pelos clientes. Alguns destes procedimentos recebem como parâmetros objetos do modelo do domínio da aplicação. A interação entre os componentes se dá através de envelopes SOAP, transportados através do protocolo HTTP. Desta forma os métodos HTTP utilizados nas requisições pouco importam, pois a semântica das operações efetuadas está descrita no conteúdo dos envelopes.

Na ROA cada recurso é exposto no servidor através de sua URI. Clientes interessados em acessar ou manipular um recurso podem escolher entre um conjunto limitado de operações do HTTP, cada qual com uma semântica definida pelo protocolo. A troca de dados entre cliente e servidor se dá através de representações de recursos, e o cliente pode explorar outros recursos através da navegação pelos enlaces presentes nestas representações. Esta dinâmica de interação entre os componentes, que envolve a identificação de recursos por URIs, manipulação de recursos através de representações, mensagens auto-descritivas e hipermídia é o que lhe confere a interface uniforme. Como consequência de sua interface uniforme a ROA consegue desfrutar de uma maior integração com a Web, conforme foi levantado no capítulo 4, e conforme será exemplificado a seguir.

6.4.1 Integração com a Web

Um ponto que distingue as arquiteturas quanto a sua integração com a Web é a possibilidade de utilizar cache e componentes intermediários. Caso as informações da aplicação dos tanques fossem disponibilizadas publicamente na Web, poderiam haver problemas relacionados a escalonabilidade se aumentasse muito o número de clientes ou se ocorressem fenômenos de sobrecarga transiente. A ROA pode lidar com esses problemas através dos mecanismos de cache presentes no HTTP. Para isto, além dos conectores de cache dos clientes e servidores, podem ser utilizados componentes intermediários atuando como caches compartilhadas entre diferentes clientes.

Como as mensagens são auto-descritivas os componentes intermediários tem como saber a semântica de cada operação realizada, fazendo que a cache funcione corretamente. Desta forma respostas para operações seguras (FIELDING et al., 1999) como o método GET podem ser guardadas

em cache e enviadas aos clientes caso o recurso não tenha sido atualizado (o que pode ser verificado através dos cabeçalhos HTTP), melhorando a performance percebida pelo usuário e a escalonabilidade. Já as operações não seguras (PUT, POST e DELETE) que alteram o estado dos recursos, tem seu efeito esperado, invalidando qualquer resposta guardada em cache para o recurso em questão. Mais caches podem ser adicionadas na medida que cresce o número de clientes interessados nos recursos.

O problema da abordagem de Web Services RPC é que suas mensagens não podem ser totalmente compreendidas pelas caches e intermediários em geral como proxies e gateways. Por não existir uma interface uniforme, os componentes intermediários teriam que se valer de informações externas ao protocolo HTTP e específicas da aplicação em questão para decidir o efeito das mensagens. Seria necessário por exemplo que uma cache compartilhada soubesse qual é a semântica dos procedimentos startShipping, stopShipping e updateHMI e se estes são seguros ou não, para que possa decidir se uma resposta pode ser guardada em cache, ou mesmo para determinar se estes procedimentos invalidam respostas na cache.

A existência de enlaces nas representações também é um diferencial da ROA, pois permite que os recursos da aplicação possam ser integrados de forma trivial a outros recursos da Web. Os enlaces permitem que qualquer cliente da Web possa vir a descobrir um enlace para a aplicação, explorando-a incrementalmente através da navegação pelos enlaces subsequentes. Na abordagem de Web Services RPC não existem estas ligações entre os serviços, de forma que o cliente precisa conhecer a priori a interface e endereço do servidor, assim como a ordem correta das operações que deseja realizar.

Outro ponto é a questão da evolução independente, um dos requisitos da Web. Na abordagem de Web Services RPC os componentes necessitam compartilhar uma infraestrutura de software (os stubs) específica de cada aplicação para que possam ser comunicar. Por exemplo, para que o componente shippingControlWS possa se comunicar com o componente receivingControlWS ele depende do stub proxyReceivingControlWS, que atua como um conector cliente, enviando chamadas remotas para o conector servidor do componente receivingControlWS. Como é possível notar as interfaces dos componentes estão intrinsecamente ligadas, sendo necessário que cada cliente possua um conector específico para se comunicar com cada servidor.

Esta dependência introduz um forte acoplamento na arquitetura. Mudanças na interface dos componentes muitas vezes tornam necessário que os stubs sejam gerados e publicados novamente para que seja mantida a compatibilidade. O custo deste acoplamento cresce com o número de componentes envolvidos, comprometendo a evolução independente. Além disso na Web os componentes podem estar sob domínio de organizações diferentes, tornando

o controle destas mudanças problemático.

Outro aspecto do acoplamento está no compartilhamento de um modelo de dados comum. Na arquitetura de Web Services isto é visível através da troca de objetos da classe Order entre os componentes. Para realizar esta troca, os objetos desta classe são codificados em documentos XML, sendo novamente transformados em objetos assim que recebidos pelos stubs de seus destinatários. Desta forma mudanças no modelo de dados devem ser refletidas em todos os componentes que compartilham objetos desta classe, assim como nos stubs que realizam a conversão/desconversão destes objetos em documentos XML.

Na ROA os componentes não estão acoplados a uma infraestrutura de software específica da aplicação para que possam se comunicar. A comunicação entre os componentes se dá através de conectores HTTP genéricos. Isto implica que qualquer cliente dotado de um conector HTTP (qualquer navegador por exemplo) pode usufruir dos recursos das plantas, ou de qualquer outro servidor da Web, não sendo necessária recompilação ou instalação de módulo adicional.

Também não existe a dependência de um modelo de dados compartilhado ou de um formato de dados específico como o XML, dado que os clientes podem escolher diferentes representações para o mesmo recurso, e processá-las através de bibliotecas existentes para o tipo de mídia escolhido. Neste caso o fraco acoplamento da arquitetura traz vantagens quando se trata de um ambiente heterogêneo como é o da Web, onde existem clientes com diferentes necessidades e em evolução constante.

A desvantagem da interface uniforme é que ela pode degradar a eficiência, pois as informações são transferidas de forma padronizada ao invés de especificamente para as necessidades de cada aplicação (FIELDING, 2000). Na aplicação dos tanques este aspecto fica claro quando são utilizadas as operações PUT, sendo necessário enviar na representação o estado completo do recurso que se deseja atualizar. Na abordagem RPC as atualizações são efetuadas por procedimentos como startShipping que atualizam o estado do servidor sem que sejam enviados parâmetros de atualização na requisição.

Outro ponto está no número de requisições efetuadas para se cumprir com determinada funcionalidade, que foi maior na arquitetura ROA. Contudo este fato está mais relacionado com a granularidade dos recursos da modelagem proposta do que com a interface uniforme propriamente dita.

As diferenças entre as arquiteturas quanto a sua integração com a Web estão sintetizadas na tabela 12, onde podem ser vistos quais aspectos da Web são aproveitados por cada uma delas. Na tabela é possível verificar que arquiteturas de Web Services RPC se aproveitam parcialmente das características da Web, utilizando-a basicamente para o transporte de suas mensagens. Por

Aspecto da Web	Web Services RPC	ROA
HTTP	X	X
URI	X	X
Tipos de mídia		X
Componentes intermediários		X
Enlaces		X
Cache		X

Tabela 12: Comparação entre as arquiteturas com relação a integração com a Web

outro lado a ROA consegue se aproveitar da Web como o um todo, pois ela foi projetada com base nos mesmos princípios de arquitetura.

6.4.2 Requisitos não-funcionais de SCADA

Conforme foi possível observar existem diferenças entre as duas abordagens que impactam na forma na qual elas se integram com a Web. Contudo, é importante direcionar esta análise para que possamos comparar as arquiteturas em relação aos requisitos típicos de SCADA, expostos no capítulo 2. Como a arquitetura de Web Services RPC não foi implementada, não foi possível realizar experimentos que a comparem com a ROA no cumprimento dos requisitos funcionais típicos de SCADA. Portanto a análise será feita somente com relação aos requisitos não-funcionais, ou seja: interoperabilidade, portabilidade, escalonabilidade, confiabilidade e segurança.

Ambas as arquiteturas usam o protocolo HTTP para a troca de mensagens além de permitirem a utilização de formatos de dados independentes de plataforma como o XML. Portanto ambas as arquiteturas são independentes de plataforma específica e possibilitam que aplicações em plataformas heterogêneas interajam, sendo equivalentes nos quesitos de interoperabilidade e portabilidade.

Com relação a escalonabilidade a arquitetura ROA é favorecida pelos estilos arquiteturais da Web. Os servidores não necessitam armazenar estado entre múltiplas requisições, facilitando a liberação de recursos. Conectores de cache podem ser utilizados por clientes e servidores, diminuindo o número de interações com a rede. Além disso, caches compartilhadas podem ser facilmente incorporadas para lidar com o aumento no número de clientes ou problemas de sobrecarga transiente. Já a arquitetura de Web Services RPC não possui uma interface uniforme, o que dificulta o processamento de suas men-

sagens pelos conectores de cache e componentes intermediários, conforme foi descrito anteriormente.

A segurança da arquitetura ROA foi coberta através do HTTPS, combinado com autenticação HTTP Basic e um sistema de papéis para autorização. Estes mecanismos possibilitam que as propriedades de disponibilidade, confidencialidade, integridade possam ser cumpridas pela arquitetura. Uma limitação desta abordagem é que o HTTPS garante somente a comunicação segura ponto a ponto entre os componentes. Esta limitação implica que clientes e servidores precisam confiar nos componentes intermediários para que estes possam atuar.

Na abordagem de Web Services RPC proposta por (FAN; CHEDED; TOKER, 2005) as tecnologias da Web (como o SSL) são citadas como solução para o requisito de segurança. O SSL é um dos protocolos de transporte seguros que podem ser utilizados com o HTTPS, sendo portanto a abordagem equivalente em termos de segurança. Contudo, dentro da mesma linha de Web Services adotada pelo autor existe a especificação WS-Security, que permite segurança fim-a-fim para as mensagens. Desta forma, utilizando WS-Security podem existir intermediários sem que seja necessário que estes sejam confiáveis. Já em relação a confiabilidade ambas as arquiteturas são equivalentes, pois permitem que mecanismos de tolerância a faltas sejam incorporados a elas.

As diferenças entre as arquiteturas em relação aos requisitos não-funcionais típicos de aplicações SCADA estão sintetizadas na tabela 13. Na tabela é possível verificar que as abordagens são equivalentes em termos de interoperabilidade, portabilidade e confiabilidade, mas diferem nos aspectos de escalonabilidade e segurança. No quesito da escalonabilidade as caches favorecem a ROA. Já o suporte a segurança da abordagem de Web Services é mais completo, pois permite segurança fim-a-fim das mensagens.

Com base nesta análise pode-se chegar a algumas conclusões. Caso somente os requisitos não-funcionais de SCADA sejam levados em conta, as abordagens se equilibram, sendo o fiel da balança a escolha entre a segurança fim-a-fim e uma maior escalonabilidade. Entretanto na questão de integração com a Web a arquitetura ROA apresenta mais vantagens do que a de Web Services RPC.

6.5 Conclusão do Capítulo

Neste capítulo foi apresentado um estudo comparativo entre arquiteturas de software para uma aplicação SCADA na Web. Primeiramente foi descrita a aplicação da transferência entre tanques, sendo propostas duas ar-

Requisito	Web Services RPC	ROA
Interoperabilidade	Tecnologias da Web	Tecnologias da Web
Portabilidade	Tecnologias da Web	Tecnologias da Web
Escalonabilidade	Sem cache	Com cache
Confiabilidade	Tolerância a faltas	Tolerância a faltas
Segurança	Fim-a-fim	Ponto-a-ponto

Tabela 13: Comparação entre as arquiteturas e requisitos não-funcionais de SCADA

quiteturas que modelam as funcionalidades desta aplicação. As duas arquiteturas foram então comparadas com base em sua integração com a Web e no cumprimento dos requisitos não-funcionais típicos de SCADA. Por fim concluiu-se que a ROA aproveita melhor os recursos da Web, e que ambas as arquiteturas apresentam um equilíbrio em relação ao cumprimento dos requisitos não-funcionais típicos de SCADA.

7 Conclusão

Esta dissertação mostrou como uma aplicação SCADA típica pode ser projetada de forma que seus requisitos sejam atendidos dentro dos princípios arquiteturais que fundamentam a Web. Foram analisadas as arquiteturas de software comumente propostas para SCADA, e as dificuldades que essas arquiteturas, baseadas em RPC, apresentam para realizar uma integração com a Web, devido a questões como acoplamento forte, manutenção de estado e uso de interfaces especializadas.

Tomando como cenário uma CFM, ambiente típico da indústria, foi projetada uma Arquitetura Orientada a Recursos que englobou as funcionalidades geralmente oferecidas por aplicações SCADA: aquisição de dados, controle em nível de supervisão, configurações, históricos, alarmes e IHMs. As funcionalidades foram cobertas de forma satisfatória, com exceção dos alarmes, cuja abordagem baseada em "polling" possui algumas limitações de escalonabilidade.

Na abordagem desenvolvida as funcionalidades foram modeladas por um conjunto de recursos, expostos através de servidores HTTP, que atuam como a interface pública do SCADA para aplicações clientes. Para lidar com dispositivos que não possuem suporte ao HTTP o servidor pode atuar como um *gateway* na comunicação com as estações remotas em campo. Os clientes podem exibir sinóticos em uma IHM para operadores, esperar pelo disparo de alarmes, controlar o processo, configurar dispositivos e abastecer com dados sistemas de MES/ERP. A arquitetura também permite a inclusão de componenentes intermediários como os proxies e caches, podendo favorecer aspectos de escalabilidade.

Os requisitos não funcionais geralmente enfatizados em SCADA são contemplados pela arquitetura. A interoperabilidade e portabilidade são favorecidas através do uso de tecnologias abertas da Web. A segurança foi coberta através do HTTPS, combinado com autenticação HTTP Basic e pelo modelo RBAC na autorização. A escalabilidade é favorecida pelo fato de a arquitetura estar embasada nos estilos arquiteturais que fundamentam a Web. Mecanismos para prover confiabilidade não foram implementados, contudo foi mostrado que os mesmos podem ser incorporados na arquitetura. Processos com requisitos de tempo-real não foram cobertos pela arquitetura devido a natureza não-determinística da Internet.

Através de uma implementação foram desenvolvidas aplicações clientes para interagir com os recursos. Estas aplicações mostraram aspectos de simplicidade, portabilidade e interoperabilidade da arquitetura. A implementação também contribuiu também para mostrar como é estruturada a

arquitetura de software de um aplicativo SCADA existente para a Web, o Mango M2M.

Na comparação com uma arquitetura existente, foram mostrados os aspectos de integração com a Web que diferenciam a abordagem ROA das arquiteturas baseadas em RPC, devido ao seu suporte a tipos de mídia, cache, componentes intermediários e enlaces. Nesta analise também foi mostrado que ambas as abordagens possuem suporte equivalente aos requisito típicos de SCADA nos aspectos da interoperabilidade, portabilidade e confiabilidade, mas diferem em termos de escalonabilidade e segurança. Outro aspecto interessante da análise foi demonstrar que a arquitetura pode modelar os requisitos de outra aplicação SCADA, expondo sua flexibilidade.

Dentre trabalhos futuros que podem ser desenvolvidos encontram-se:

- Avaliação da arquitetura para outras aplicações típicas de SCADA. Nesta avaliação podem surgir novos requisitos a serem incorporados, possibilitando que a arquitetura seja extendida, tornando-a mais flexível.

- Realização de experimentos que avaliem quantitativamente aspectos de performance e escalonabilidade da implementação da arquitetura, dado que a análise realizada foi feita somente com base em critérios qualitativos.

- Estudo e implementação de ferramentas para utilizar a arquitetura na educação de engenharia, em especial na integração com linguagens de programação de propósito educacional e redes sociais.

- Implementação de mecanismos para prover confiabilidade para a arquitetura, tais como algoritmos de tolerância a faltas.

- Estudo mais aprofundado dos aspectos de segurança da arquitetura, tais como mecanismos de gerenciamento de chaves criptográficas e políticas de controle de acesso.

- Integração de tecnologias de Syndication como o padrão Atom na arquitetura (SAYRE, 2005).

- Estudo de alternativas para o funcionamento dos alarmes, dado que a abordagem proposta possui algumas limitações. Neste aspecto pode-se destacar a avaliação de estilos de arquitetura baseados em eventos no cumprimento desta funcionalidade.

- Muitos esforços estão centrados em torno de criar grandes redes de dispositivos inteligentes, encontradas no mundo físico. Abordagens recentes propõem a integração destes dispositivos com a Web, formando

uma “Web das coisas” (GUINARD; TRIFA; WILDE, 2010). Estudar a integração da arquitetura proposta para SCADA com estas abordagens parece promissor.

Referências Bibliográficas

ADAMO, F. et al. Scada/hmi systems in advanced educational courses. *Instrumentation and Measurement, IEEE Transactions on*, v. 56, n. 1, p. 4 –10, feb. 2007. ISSN 0018-9456.

Altus. *"Manual de Utilização PO3047/PO3147/PO324 UCPs Série Ponto"*. [S.l.], 2011.

ALUR, D. et al. *Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies*. Mountain View, CA, USA: Sun Microsystems, Inc., 2003. ISBN 0131422464.

BAILEY, D.; WRIGHT, E. *Practical SCADA for industry*. [S.l.]: Newnes, 2003. ISBN 0750658053.

BALASUBRAMANIAN, J. et al. Adaptive failover for real-time middleware with passive replication. In: IEEE. *15th IEEE Real-Time and Embedded Technology and Applications Symposium*. [S.l.], 2009. p. 118–127.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. [S.l.]: Addison-Wesley Professional, 1997. ISBN 0201199300.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. 2. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN 0321154959.

BENTO, D. et al. *Maquete de uma Célula Flexível de Manufatura*. 2007.

BERNAT, G.; BURNS, A.; LIAMOSI, A. Weakly hard real-time systems. *Computers, IEEE Transactions on*, IEEE, v. 50, n. 4, p. 308–321, 2001.

BERNER-LEE, T. Information management: A proposal. Http://www.w3.org/History/1989/proposal.html.

BERNER-LEE, T. The world wide web: A very short personal history. Http://www.w3.org/People/Berners-Lee/ShortHistory.html.

BERNERS-LEE, T. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web*. [S.l.]: Harper Paperbacks, 2000. ISBN 006251587X.

BIRRELL, A. D.; NELSON, B. J. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, v. 2, p. 39–59, 1984.

BONY, B.; HARNISCHFEGER, M.; JAMMES, F. Convergence of opc ua and dpws with a cross-domain data model. In: *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*. [S.l.: s.n.], 2011. p. 187 –192.

BOX KAKIVAYA, e. a. Soap: Simple object access protocol. Http://scripting.com/misc/soap1.txt. 1999.

BOZDAG, E.; MESBAH, A.; DEURSEN, A. V. A comparison of push and pull techniques for ajax. In: IEEE. *Web Site Evolution, 2007. WSE 2007. 9th IEEE International Workshop on*. [S.l.], 2007. p. 15–22.

BRATAAS, G.; HUGHES, P. Exploring architectural scalability. In: *Proceedings of the 4th international workshop on Software and performance*. New York, NY, USA: ACM, 2004. (WOSP '04), p. 125–129. ISBN 1-58113-673-0.

CANDIDO, G. et al. Soa at device level in the industrial domain: Assessment of opc ua and dpws specifications. In: *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*. [S.l.: s.n.], 2010. p. 598 –603.

CASIMIRO, A. *Uma Panorâmica Sobre Sistemas SCADA*. [S.l.], 1995.

CHAN, F.; CHAN, H. A comprehensive survey and future trend of simulation study on fms scheduling. *Journal of Intelligent Manufacturing*, Springer, v. 15, n. 1, p. 87–102, 2004.

CHANG, F. et al. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, ACM, v. 26, n. 2, p. 1–26, 2008.

CHAU, T.; KHAI, N. Web-based data monitoring and supervisory control. In: *Proceedings of the int. conference ISEE*. [S.l.: s.n.], 2007.

DANEELS, A.; SALTER, W. What is SCADA? In: *Proceedings on the International Conference on Accelerator and Large Experimental Physics Control System, Trieste, Italy*. [S.l.: s.n.], 1999.

DAWSON, R. et al. Skma: a key management architecture for scada systems. In: AUSTRALIAN COMPUTER SOCIETY, INC. *Proceedings of the 2006 Australasian workshops on Grid computing and e-research-Volume 54*. [S.l.], 2006. p. 183–192.

DIERKS, T.; ALLEN, C. *The TLS Protocol Version 1.0*. United States: RFC Editor, 1999.

DOLLIMORE, J.; KINDBERG, T.; COULOURIS, G. *Distributed Systems: Concepts and Design (4th Edition)*. [S.l.]: Addison Wesley, 2005. ISBN 0321263545.

DWR. *DWR Documentation*. 2010. Disponível em: http://directwebremoting.org/dwr/ Acessado em: 07 de Outubro de 2010. Disponível em: <http://directwebremoting.org/dwr>.

EUGSTER, P. et al. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, ACM, v. 35, n. 2, p. 114–131, 2003.

FALLIERE, N.; MURCHU, L.; CHIEN, E. W32. stuxnet dossier. *Symantec Security Response,[Online], Accessed*, v. 14, 2010.

FAN, R.; CHEDED, L.; TOKER, O. *Investigation Of Internet-Based SCADA Systems: Design and Applications*. Dissertação (Mestrado) — King Fahd University Of Petroleum Minerals, 2004.

FAN, R.; CHEDED, L.; TOKER, O. Internet-based scada: a new approach using java and xml. *Computing Control Engineering Journal*, v. 16, n. 5, p. 22 – 26, oct.-nov. 2005. ISSN 0956-3385.

FAVARETTO, F. *Contribuição ao Processo de Gestão da Produção pela utilização da coleta de dados de chão de fábrica*. Tese (Doutorado) — Universidade de São Paulo, 2001.

FENG-PING, Y.; CHUN-HUA, F.; JIAN, W. Design of a software of drawing monitoring graphics based on java and svg [j]. *Relay*, 2008.

FERRAIOLO, D.; CUGINI, J.; KUHN, D. Role-based access control (rbac): Features and motivations. In: *Proceedings of 11th Annual Computer Security Application Conference*. [S.l.: s.n.], 1995. p. 11–15.

FIELDING, R. et al. *Hypertext Transfer Protocol – HTTP/1.1*. 1999. RFC 2616. Available from http://www.ietf.org/rfc/rfc2616.txt. Disponível em: <http://www.ietf.org/rfc/rfc2616.txt>.

FIELDING, R. T. *Architectural styles and the design of network-based software architectures*. Tese (Doutorado) — University of California, Irvine, 2000. AAI9980887.

FIELDING, R. T.; TAYLOR, R. N. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, ACM, New York, NY, USA, v. 2, p. 115–150, May 2002. ISSN 1533-5399.

FOWLER, M. *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0321127420.

FRANKS, J. et al. *HTTP Authentication: Basic and Digest Access Authentication*. United States: RFC Editor, 1999.

GARCIA, Y.; QUEIROZ, M. H. Formal synthesis, simulation and automatic code generation of supervisory control for a manufacturing cell. *International Congress of Mechanical Engineering*, 2009.

GARLAN, D.; SHAW, M. An introduction to software architecture. *Advances in software engineering and knowledge engineering*, Singapore, v. 1, p. 1–40, 1993.

GHEZZI, C.; JAZAYERI, M.; MANDRIOLI, D. *Fundamentals of Software Engineering*. 2nd. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2002. ISBN 0133056996.

GOLONKA, P.; GONZALEZ-BERGES, M. Integrated access control for pvss-based scada systems at cern. In: ICALEPCS. [S.l.], 2009.

GOMES, M. F. P. *Canal SCADA na Web*. Dissertação (Mestrado) — Universidade do Porto, 2003.

GUINARD, D.; TRIFA, V.; WILDE, E. A resource oriented architecture for the web of things. In: IEEE. *Internet of Things (IOT), 2010*. [S.l.], 2010. p. 1–8.

HADLICH, T. Providing device integration with opc ua. In: *Industrial Informatics, 2006 IEEE International Conference on*. [S.l.: s.n.], 2006. p. 263 –268.

HAMMER-LAHAV, E. *The oauth 1.0 protocol*. 2010.

HANNELIUS, T.; SALMENPERA, M.; KUIKKA, S. Roadmap to adopting opc ua. In: IEEE. *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*. [S.l.], 2008. p. 756–761.

HARJUNKOSKI, I.; NYSTROM, R.; HORCH, A. Integration of scheduling and control–theory or practice? *Computers Chemical Engineering*, v. 33, n. 12, p. 1909 – 1918, 2009. ISSN 0098-1354. FOCAPO 2008 - Selected Papers from the Fifth International Conference on Foundations of Computer-Aided Process Operations.

HONG, X.; JIANHUA, W. Using standard components in automation industry: A study on opc specification. *Computer Standards & Interfaces*, Elsevier, v. 28, n. 4, p. 386–395, 2006.

HOWELLS, R. Erp needs shop-floor data. *Manufacturing Engineering*, vol. 125, n. Issue 4, p. 54, 7 , 2c, 2000.

IEEE. *Ieee Standard Computer Dictionary*. City: Inst of Elect Electronic, 1991. ISBN 1559370793.

IZAGUIRRE, M. J. A. G.; LOBOV, A.; LASTRA, J. L. M. Opc-ua and dpws interoperability for factory floor monitoring using complex event processing. In: *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*. [S.l.: s.n.], 2011. p. 205 –211.

JSON. *JSON*. 2010. http://www.json.org/.

JUHASZ, B. *Developing M2M Applications With Mango*. [S.l.], 2009.

KAPSALIS, V.; KOUBIAS, S.; PAPADOPOULOS, G. Opc-sms: a wireless gateway to opc-based data sources. *Computer Standards & Interfaces*, Elsevier, v. 24, n. 5, p. 437451, 2002.

KAYE, D. *Loosely Coupled: The Missing Pieces of Web Services*. [S.l.]: RDS Press, 2003. ISBN 1881378241.

KHATHAIR, A. Y. *Design and Implementation of Web Based SCADA*. Dissertação (Mestrado) — Nahrain University, 2006.

KHATIB, A. et al. Thoughts on future Internet based power system information network architecture. In: IEEE. *Power Engineering Society Summer Meeting, 2000. IEEE*. [S.l.], 2000. v. 1, p. 155–160. ISBN 0780364201.

KRUTZ, R. L. *Securing SCADA Systems*. [S.l.]: Wiley, 2005. ISBN 9780764597879.

LI, D.; SERIZAWA, Y.; KIUCHI, M. Concept design for a web-based supervisory control and data-acquisition (scada) system. In: *Transmission and Distribution Conference and Exhibition 2002: Asia Pacific. IEEE/PES*. [S.l.: s.n.], 2002. v. 1, p. 32 – 36 vol.1.

MACLAREN, J.; KEOWN, M. M. *HARC: A Highly-Available Robust Co-scheduler, submitted to the 5th UK e-Science All Hands Meeting*. 2006.

MAJDALAWIEH, M.; PARISI-PRESICCE, F.; SANDHU, R. Rbac model for scada. *Innovative Algorithms and Techniques in Automation, Industrial Electronics and Telecommunications*, Springer, p. 329–335, 2007.

MARDEGAN, R.; AZEVEDO, R.; OLIVEIRA, J. Os benefícios da coleta automática de dados do chão de fábrica para o processo de negócio gestão da demanda. *XXII Encontro Nacional de Engenharia de Produção*, 2002.

MEDIDA, S.; SREEKUMAR, N.; PRASAD, K. SCADA-EMS on the Internet. In: IEEE. *Energy Management and Power Delivery, 1998. Proceedings of EMPD'98. 1998 International Conference on*. [S.l.], 1998. v. 2, p. 656–660. ISBN 0780344952.

MESA, I. Mes explained: A high level vision. *MESA International White Paper6*, v. 1, p. 25, 1997.

Modbus Organization. *Modbus Technical Resources*. [S.l.], 2011.

MORAES, P. M. Scada, model data combine to provide real-time commercial supervision at petrobras. *Pipeline Gas Journal*, 2005.

MYERS, G. J. *Composite Structure Design*. New York, NY, USA: John Wiley & Sons, Inc., 1978. ISBN 0442805845.

National Communications Systems. *Supervisory Control and Data Acquisition (SCADA) Systems*. [S.l.], 2004.

NELSON, T. H. Complex information processing: a file structure for the complex, the changing and the indeterminate. In: *Proceedings of the 1965 20th national conference*. New York, NY, USA: ACM, 1965. (ACM '65), p. 84–100.

NEWMAN, M. W. et al. Designing for serendipity: supporting end-user configuration of ubiquitous computing environments. In: *Proceedings of the 4th conference on Designing interactive systems: processes, practices, methods, and techniques*. New York, NY, USA: ACM, 2002. (DIS '02), p. 147–156. ISBN 1-58113-515-7.

OASIS. *OASIS Web Services Discovery and Web Services Devices Profile (WS-DD) Technical Committee*. http://www.oasis-open.org/committees/ws-dd/charter.php: OASIS, 2006.

PAUTASSO, C.; WILDE, E. Why is the web loosely coupled?: a multi-faceted metric for service design. In: *Proceedings of the 18th international conference on World wide web*. New York, NY, USA: ACM, 2009. (WWW '09), p. 911–920. ISBN 978-1-60558-487-4.

PERRY, D.; WOLF, A. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 17, n. 4, p. 40–52, 1992.

PIERI, G. et al. Telis: a programming tool set for beginners. In: *8th International Information and Telecommunication Technologies Symposium*. [S.l.: s.n.], 2009.

POPA, M.; POPA, A.; PATITOIU, C. A web connected smart sensor. In: *Applied Computational Intelligence and Informatics, 2007. SACI '07. 4th International Symposium on*. [S.l.: s.n.], 2007. p. 105 –110.

PRYCE, N. G. *Component Interaction in Distributed Systems*. 1998.

PUDER, A.; ROMER, K.; PILHOFER, F. *Distributed Systems Architecture: A Middleware Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005. ISBN 1558606483.

QIU, B.; GOOI, H. Web-based scada display systems (wsds) for access via internet. *Power Systems, IEEE Transactions on*, v. 15, n. 2, p. 681 –686, may 2000. ISSN 0885-8950.

QUEIROZ, M. H. D.; CURY, J. E. R. Modular supervisory control of large scale discrete event systems. In: *In Discrete Event Systems: Analysis and Control. Proc. WODES 2000*. [S.l.]: Kluwer Academic, 2000. p. 103–110.

RAZA, M.; HUSSAIN, F. k.; CHANG, E. A methodology for quality-based mashup of data sources. In: *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*. New York, NY, USA: ACM, 2008. (iiWAS '08), p. 528–533. ISBN 978-1-60558-349-5.

RICHARDSON, L.; RUBY, S. *Restful Web Services*. [S.l.]: O'Reilly Media, 2007. ISBN 0596529260.

SAYRE, R. Atom: The standard in syndication. *Internet Computing, IEEE*, IEEE, v. 9, n. 4, p. 71–78, 2005.

SEROTONIN. *Mango M2M*. 2011. Disponível em: http://mango.serotoninsoftware.com Acessado em: 31 de Janeiro de 2011. Disponível em: <http://mango.serotoninsoftware.com/>.

SHAW, M.; CLEMENTS, P. A field guide to boxology: Preliminary classification of architectural styles for software systems. In: PUBLISHED BY THE IEEE COMPUTER SOCIETY. *compsac*. [S.l.], 1997. p. 6.

SHAW, M.; CLEMENTS, P. The golden age of software architecture. *IEEE Software*, IEEE Computer Society, Los Alamitos, CA, USA, v. 23, p. 31–39, 2006. ISSN 0740-7459.

SLEMAN, A.; MOELLER, R. Integration of wireless sensor network services into other home and industrial networks; using device profile for web services (dpws). In: *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on*. [S.l.: s.n.], 2008. p. 1 –5.

SOUZA, R. B. de. *Uma Arquitetura para Sistemas Supervisórios Industriais e sua Aplicação em Processos de Elevação Artificial de Petróleo*. Dissertação (Mestrado) — UFRN, 2005.

SPRING. *Spring Reference Documentation*. 2010. Disponível em: http://www.springsource.org/documentation Acessado em: 07 de Outubro de 2010. Disponível em: <http://www.springsource.org/documentation>.

TORRISI, N.; OLIVEIRA, J. Remote control of cnc machines using the cyberopc communication system over public networks. *The International Journal of Advanced Manufacturing Technology*, Springer London, v. 39, p. 570–577, 2008. ISSN 0268-3768. 10.1007/s00170-007-1244-0.

TU, N. T. T. et al. Research and development of opc client-server architectures for manufacturing and process automation. In: *Proceedings of the 2010 Symposium on Information and Communication Technology*. New York, NY, USA: ACM, 2010. (SoICT '10), p. 163–170. ISBN 978-1-4503-0105-3.

URDANETA, G. et al. A reference software architecture for the development of industrial automation high-level applications in the petroleum industry. *Comput. Ind.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 58, p. 35–45, January 2007. ISSN 0166-3615.

VINOSKI, S. Demystifying restful data coupling. *IEEE Internet Computing*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 12, p. 87–90, March 2008a. ISSN 1089-7801.

VINOSKI, S. Serendipitous reuse. *IEEE Internet Computing*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 12, p. 84–87, January 2008b. ISSN 1089-7801.

W3C. A little history of the world wide web. Http://www.w3.org/History.html. 2000.

W3C. *Web Services Architecture*. [S.l.], February 2004.

WALLACE, D. Smart fields come of age with internet-based scada. *Pipeline & gas journal*, Oildom Publishing Co, 14515 Briar Hills Parkway, Houston, TX, 77077, USA,, v. 231, n. 2, p. 27–27, 2004.

WARD, M. P. et al. An architectural framework for describing supervisory control and data acquisition (scada) systems, (master's thesis, naval postgraduate school. In: *69 PAGE INTENTIONALLY LEFT BLANK 70 A SCADA MAC PREFIXES A.1 NMAP-MAC-PREFIXES FILE This*. [S.l.: s.n.], 2004.

WHITE, J. E. A high-level framework for network-based resource sharing. In: *Proceedings of the June 7-10, 1976, national computer conference and exposition*. New York, NY, USA: ACM, 1976. (AFIPS '76), p. 561–570. Disponível em: <http://doi.acm.org/10.1145/1499799.1499878>.

WINER, D. Dave's history of soap. Http://www.xmlrpc.com/stories/storyReader555.1999.

ZECEVIC, G. Web based interface to scada system. In: *Power System Technology, 1998. Proceedings. POWERCON '98. 1998 International Conference on*. [S.l.: s.n.], 1998. v. 2, p. 1218 –1221 vol.2.

Glossário

API	Application Programming Interface. 66
CFM	Célula Flexível de Manufatura. 20
CORBA	Common Object Request Broker Architecture. 60
CSV	Comma-separated Values. 51
DAO	Data Access Object. 96
DCOM	Distributed Component Object Model. 60
DNP3	Distributed Network Protocol. 30
DPWS	Devices Profile for Web Services. 64
DTO	Data Transfer Object. 96
DWR	Direct Web Remoting. 93
ERP	Enterprise Resource Planning. 17
FMS	Flexible Manufacturing System. 69
HTTP	HyperText Transfer Protocol. 37
IHM	Interface Homem-Máquina. 24
JSON	JavaScript Object Notation. 49
M2M	Machine to Machine. 69
MES	Manufacturing Execution Systems. 17
MVC	Model View Controller. 94
OPC-UA	OPC Unified Architecture. 64
PNG	Portable Network Graphics. 51
REST	Representational State Transfer. 20
ROA	Resource Oriented Architecture. 18
RPC	Remote Procedure Calls. 18
RTU	Remote Terminal Unit. 23

SCADA Supervisory Control and Data Aquisition. 17
SSL Secure Sockets Layer. 52
SVG Scalable Vector Graphics. 51

TLS Transport Layer Security. 52

URI Uniform Resource Identifier. 37

XML Extensible Markup Language. 18