RENAN MARTINS

INTEGRAÇÃO DE DISPOSITIVO COLETOR DE DADOS MÓVEL COM SOFTWARE ORIENTADO A PROCESSOS DE NEGÓCIO

LAVRAS - MG

2010

RENAN MARTINS

INTEGRAÇÃO DE DISPOSITIVO COLETOR DE DADOS MÓVEL COM SOFTWARE ORIENTADO A PROCESSOS DE NEGÓCIO

Monografia de Graduação apresentada ao Colegiado do Curso de Ciência da Computação, para obtenção do título de Bacharel em Ciência da Computação.

Orientador

Prof. Dr. André Vital Saúde

LAVRAS - MG

2010

RENAN MARTINS

INTEGRAÇÃO DE DISPOSITIVO COLETOR DE DADOS MÓVEL COM SOFTWARE ORIENTADO A PROCESSOS DE NEGÓCIO

Monografia de Graduação apresentada ao Colegiado do Curso de Ciência da Computação, para obtenção do título de Bacharel em Ciência da Computação.

Aprovada em *19 de Novembro de 2010*

Prof. Dr. Tales Heimfarth

Msc. Marlon Marcon

Prof. Dr. André Vital Saúde

Orientador

LAVRAS - MG

2010

À minha amada Nayara, dedico.

AGRADECIMENTOS

Agradeço primeiramente a Deus, pela vida e saúde.

Agradeço de forma muito especial aos meus pais, Rafael e Vera, pela educação, amor, confiança e incentivo depositados em mim durante minha vida. Não teria chegado até aqui se não fosse por vocês. Obrigado!

Agradeço aos meus queridos irmãos, Carolina e Rafael, pela amizade e bons exemplos. Me considero um caçula de muita sorte. Obrigado!

Agradeço à minha namorada Nayara pelo amor, carinho, amizade, estímulo, auxílio e compreensão durante toda essa jornada. Por todos esses anos, você foi a responsável pelos momentos de inspiração, motivação e felicidade. Obrigado!

Agradeço a todos os meus familiares: avós, tios, primos, cunhados e sogros pela força. Obrigado!

Agradeço a todos os meus amigos, em especial aos colegas da 2007/01, companheiros fiéis. Tive o prazer e o privilégio de fazer parte dessa turma. Obrigado!

Agradeço a todos os professores e funcionários do Departamento de Ciência da Computação da UFLA, em especial ao professor André Vital Saúde pela orientação, amizade, compreensão e paciência durante o desenvolvimento deste trabalho. Obrigado!

Que nunca falte paz, amor e alegria na vida dessas pessoas.

Muito Obrigado a todos!

RESUMO

Em muitos casos, informações que precisam estar alinhadas aos processos de negócio das corporações são obtidas através de equipamentos de *hardware* que funcionam de forma independente. Este é o caso de alguns sistemas de rastreabilidade, que utilizam equipamentos para coletar os dados do processo de forma automatizada. Diante disto, este trabalho explorou as dificuldades de realizar a integração de um dispositivo coletor de dados móvel, denominado Apix, com um *software* web, orientado a processos de negócio, denominado Labora. Juntos, eles possibilitam completamente a rastreabilidade de produtos apícolas na primeira etapa da cadeia produtiva. A integração foi implementada apoiando-se em boas práticas de desenvolvimento e padrões conhecidos, garantindo alta manutenibilidade e extensibilidade à solução desenvolvida. Além disso, através da integração realizada, objetivou-se demonstrar tecnologias e estratégias que podem ser utilizadas na realização de integrações entre equipamentos de *hardware* e sistemas web como um todo.

Palavras-chave: Integração de sistemas, Dispositivo coletor de dados, Sistemas Web, Service Oriented Architecture, Dispositivos móveis, SOA

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

CRM Customer Relationship Management

ERP Enterprise Resource Planning

TI Tecnologia da Informação

XML eXtensible Markup Language

XSD XML Schema Definition

SOA Service-Oriented Architecture

JAXB Java Architecture for XML Binding

API Application Programming Interface

JRE Java Runtime Environment

JCP Java Community Process

JNI Java Native Interface

UDDI Universal Description, Discovery and Integration

WSDL Web Services Description Language

SOAP Simple Object Access Protocol

WS-I Web Services Interoperability Organization

UTF-8 8-bit Unicode Transformation Format

JNLP Java Network Lauching Protocol

HTML HyperText Markup Language

JVM Java Virtual Machine

GPS Global Positioning System

BPMS Business Process Management Suite

IDE Integrated Development Environment

Java EE Java Platform, Enterprise Edition

SUMÁRIO

1 Introdução

Neste capítulo é fornecida uma visão global do assunto tratado por este projeto, de tal forma que fique claro quais objetivos pretendeu-se alcançar com sua execução. Além disso, os componentes do documento são descritos para facilitar sua utilização.

1.1 Motivação

A globalização dos mercados, a concorrência mundial e as rápidas mudanças nas condições dos negócios estão levando as empresas a se tornarem mais ágeis e a se comunicarem mais com parceiros, fornecedores e distribuidores. Nesse contexto, a integração e interoperabilidade de organizações e sistemas é uma preocupação sempre que duas ou mais empresas têm a necessidade de compartilhar informações (VERNADAT, 2009).

Segundo Roshen (ROSHEN, 2009), o desenvolvimento de novas aplicações ou modificações em sistemas já existentes afim de torná-los capazes de compartilhar dados e funcionalidades é chamado de integração de *software*.

Fazer com que todas as aplicações de uma organização funcionem de uma maneira integrada, lidando com dados unificados, é uma tarefa difícil pois pode envolver aplicações de diferentes tipos, como por exemplo: aplicações construídas com o uso de diferentes linguagens de programação (Java, C++, etc.), pacotes de aplicações como Customer Relationship Management (CRM), Enterprise Resource Planning (ERP) e também sistemas legados. Além disso, essas aplicações podem estar dispersas geograficamente e rodar em diferentes plataformas.

Atualmente, o alinhamento dos negócios com a Tecnologia da Informação (TI) ocupa posição de destaque nas grandes empresas. Nessas corporações, os processos de negócio, que consistem em um conjunto de atividades que são exe-

cutadas para realizar um objetivo de negócio (WESKE, 2007), são essenciais para entender como elas operam. Tais processos possuem um importante papel no desenvolvimento de sistemas de informação flexíveis, capazes de se adaptar rapidamente a uma nova funcionalidade que o mercado impõe.

Sempre que mudanças acontecem no ambiente corporativo, as informações correspondentes têm que, consequentemente, sofrer alterações para ficarem alinhadas aos processos de negócio. Porém, alterar um sistema de informação existente é uma tarefa muito demorada e propensa a erros caso não se utilize uma abordagem sistemática. Diante disso, adequar os processos de negócio às possíveis alterações de forma ágil é, portanto, um desafio (ZHANG, 2008).

Em muitos casos, as informações que precisam estar alinhadas aos processos de negócio são obtidas através de equipamentos de *hardware* que funcionam de forma independente. Este é o caso de alguns sistemas de rastreabilidade, que utilizam equipamentos para coletar os dados do processo de forma automatizada, visando uma maior usabilidade. De alguma forma, as informações armazenadas nesses equipamentos devem ser sincronizados com um sistema no qual elas têm significado e possam, assim, ser alinhadas aos processos de negócio.

1.2 Objetivo Geral

Diante do que foi exposto, este trabalho teve como objetivo realizar a integração de um dispositivo coletor de dados móvel configurável, denominado Apix, com um *software* orientado a processos de negócio, denominado Labora, que através da utilização de serviços consegue fornecer funcionalidades para outras aplicações através de interfaces simples. Além disso, através da integração realizada, objetivou-se demonstrar tecnologias e estratégias que podem ser utilizadas na rea-

lização de integrações entre equipamentos de hardware e sistemas web como um todo.

1.3 Objetivos Específicos

Para alcançar o objetivo geral, foi necessário:

Objetivo 1. Criar um protocolo de comunicação entre o dispositivo e o software web;

Foi criado um protocolo que governa a sintaxe, semântica e sincronização da comunicação entre o dispositivo e o *software*. O protocolo foi descrito utilizando o *eXtensible Markup Language (XML) Schema*, que define as regras de validação em documentos XML.

Objetivo 2. Criar uma aplicação *desktop* para se comunicar com o dispositivo;

Diante da impossibilidade de comunicação entre o *browser* e o dispositivo, uma aplicação *desktop* foi criada para se comunicar com o *hardware* através da porta serial do computador.

Objetivo 3. Criar um *driver*, que esconde da aplicação detalhes do dispositivo de *hardware*;

Afim de aumentar a extensibilidade e a manutenibilidade da solução de integração proposta por este trabalho, um *driver* foi criado.

Objetivo 4. Permitir que a aplicação *desktop* seja executada sob demanda através do sistema web;

Usando a tecnologia *Java Web Start*, aplicações de *software* desenvolvidas em Java podem ser executadas com um único clique através da rede.

Objetivo 5. Criar novos serviços no sistema web para enviar e receber informações da aplicação *desktop*;

> A aplicação *desktop*, em determinado momento, é acionada via *Java Web Start*, realiza a troca de dados com o dispositivo e, utilizando a Application Programming Interface (API) [1] fornecida pelo *software* web, invoca tais serviços enviando ou recebendo dados para que sejam processados.

1.4 Estrutura do Trabalho

O presente trabalho contem, além deste, mais quatro capítulos descritos abaixo:

- *Capítulo 2*: Referencial Teórico - As tecnologias utilizadas no desenvolvimento deste projeto são explicadas nesse capítulo, utilizando-se de fatos existentes na literatura e em trabalhos relacionados.
- *Capítulo 3*: Metodologia e Desenvolvimento - Os materiais, técnicas e métodos utilizados para conduzir o trabalho são descritos de maneira detalhada nesse capítulo. Além disso, a pesquisa é classificada quanto ao seu tipo teórico.
- *Capítulo 4*: Discussão - Os resultados alcançados com este projeto são discutidos nesse capítulo.
- *Capítulo 5*: Conclusões - Uma síntese do trabalho realizado, os elementos de contribuição do autor ao tema tratado e possíveis trabalhos futuros são apresentados nesse capítulo.

[1] Interface de Programação de Aplicativos (API) é um conjunto de rotinas e padrões estabelecidos por um *software* para a utilização das suas funcionalidades por programas aplicativos que não querem envolver-se em detalhes da implementação do *software*, mas apenas usar seus serviços.

2 Referencial Teórico

Neste capítulo são apresentados os conceitos necessários para o entendimento do projeto. O capítulo está divido nas seguintes seções:

- **2.1 "*Service-Oriented Architecture* (SOA)".** Será apresentada a Arquitetura Orientada a Serviços, suas características e os papéis envolvidos neste modelo de arquitetura.
- **2.2 "*eXtensible Markup Language* (XML)".** Será apresentado o básico da XML, uma tecnologia essencial para quem lida com integração de *software*.
- **2.3 "*XML Schema*".** Será apresentado o básico do XML *Schema*, utilizado para criar regras de validação em documentos XML.
- **2.4 "*Java Architecture for XML Binding* (JAXB)".** Será apresentada a Java Architecture for XML Binding (JAXB), criada para facilitar o acesso e manipulação de documentos XML utilizando a linguagem Java.
- **2.5 "*Java Web Start* ".** Será apresentada a tecnologia *Java Web Start*, que foi utilizada para atender a um requisito da integração que o projeto realizou.
- **2.6 "*Java Native Interface*".** Será apresentada a Java Native Interface (JNI), que permite que a máquina virtual da linguagem Java acesse bibliotecas construídas com o código nativo de um sistema.
- **2.7 "O Dispositivo Coletor de Dados".** Será apresentado o dispositivo coletor de dados, o Apix.
- **2.8 "O Software Orientado a Processos de Negócios".** Será apresentado o Labora, assim como o *framework* utilizado em sua construção, base do sistema, o Iguassu.

2.1 Service-Oriented Architecture (SOA)

Service-Oriented Architecture (SOA) ou Arquitetura orientada a serviços é um modelo de arquitetura na qual as funcionalidades implementadas pelas aplicações devem ser fornecidas na forma de serviços (GROUP, 2006).

A SOA conceitua um serviço como sendo a representação lógica de uma atividade empresarial, que é executada repetidas vezes e possui um resultado específico (GROUP, 2006). No mundo real, são exemplos de serviços:

- Consulta metereológica de determinada região;
- Verificar situação de veículo;
- Realizar um saque da conta bancária.

Tais serviços podem ser compostos por outros serviços, devem ser autocontidos e vistos como uma ”caixa preta” pelos seus consumidores. Ou seja, devem esconder a maneira como implementam determinada funcionalidade (GROUP, 2006), como mostrado pela Figura 1.

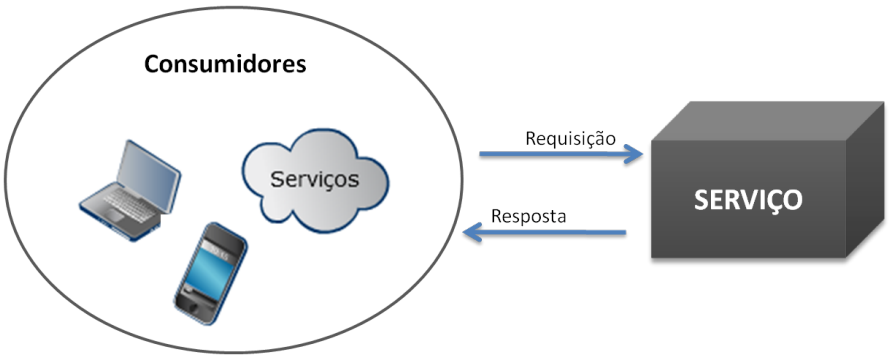

Figura 1: Demonstração de como um serviço é visto como uma ”caixa preta” pelos seus consumidores

É importante ressaltar que o modelo de arquitetura SOA é baseada no *design* dos serviços, que espelham atividades de negócio das corporações. Essas atividades, por sua vez, compõem os processos de negócio (WESKE, 2007).

2.1.1 Papéis na SOA

Para entender melhor a SOA, é necessário definir os papéis envolvidos nesse tipo de arquitetura. A Figura 2 mostra esses papéis e a relação entre eles. Cada entidade na SOA pode ter uma ou mais das três funções: *service provider*, *consumer* e *registry* (JÚNIOR, 2008).

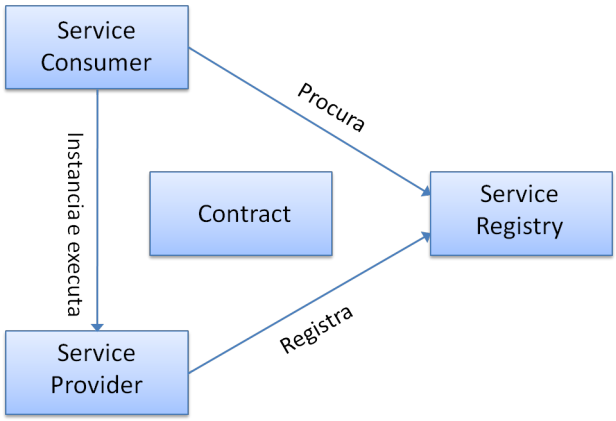

Figura 2: Papéis na *Service-oriented Architecture* - Paradigma *find-bind-execute*

- ***Service Provider***. Provê o serviço que é invocado pelo *Service Consumer*. O *Service Provider* é uma entidade com um endereço na rede que aceita e executa requisições dos consumidores. O *Service Provider* publica o contrato de seus serviços no *Service Registry* para que os consumidores os encontrem;

- ***Service Consumer***. É a entidade que consome o serviço. Pode ser uma aplicação, um componente ou outro serviço. Ela busca a localização do serviço no *Service Registry*, realiza a ligação com o serviço através de um meio de transporte e o executa através de uma requisição formatada de acordo com o contrato;

- ***Service Registry***. É um diretório de rede onde os *Service Providers* publicam seus serviços e os *Service Consumers* buscam por serviços;

- ***Contract***. É um contrato entre o consumidor e o provedor do serviço. Este contrato especifica as informações necessárias sobre o serviço, como operações, condições ”pré e pós” execução, formato da requisição, da resposta e informações sobre a qualidade do serviço (*Quality of Service*).

2.1.2 Caracaterísticas da SOA

A SOA possui um conjunto de características que fazem dela um tipo especial de arquitetura. Tais características são apresentadas abaixo (JÚNIOR, 2008):

- **Design voltado à interface**

  Pode-se dizer que o aspecto mais importante da SOA é que ela separa a implementação do seviço de sua interface. Em outras palavras, ela separa o ”o que” do ”como”. Consumidores do serviço não se preocupam em como o serviço executará a requisição (JOHNSTON; KELLY; BROWN, 2002). O que lhes interessa é o que o serviço fará. Essa característica foi exemplificada anteriormente neste trabalho com a utilização da Figura 1.

- **Capacidade de Descoberta ou Discoverability**

Serviços precisam ser encontrados tanto no projeto como em tempo de execução, não apenas por um código identificador, mas também por sua interface e seu tipo de serviço. Implementar a *discoverability* em um nível de serviços requer o uso de tecnologias de diretório (ERL, 2005), como o Universal Description, Discovery and Integration (UDDI) [2] dos *Web Services*.

- **Auto-contida e Modular**

  Um dos mais importantes aspectos da SOA é o conceito de modularidade. Um serviço suporta um conjunto de interfaces que devem ser coesas, e para isso se relacionam umas com as outras dentro do contexto de um módulo (MCGOVERN *et al.*, 2003).

- **Interoperabilidade**

  A capacidade de sistemas que utilizam diferentes plataformas e linguagens se comunicarem entre si é chamada de interoperabilidade. Para que a SOA seja utilizada de forma efetiva, é necessário prover protocolos para que os papéis da SOA possam interoperar. *Web Services* são um exemplo de tecnologia que provê um framework padronizado de comunicação que estabelece o potencial de promover a interoperabilidade entre os serviços.

- **Orientada a Processos de Negócio**

  Por possuirem interfaces bem definidas, serviços oferecem um melhor modo de expor funções do negócio. Portanto, se tor-

[2] UDDI é um exemplo de protocolo que especifica um método para publicar e descobrir diretórios de serviços em uma arquitetura orientada a serviços

nam uma boa opção ao se desenvolver aplicações que suportam processos de negócio (JOHNSTON; KELLY; BROWN, 2002). Neste caso, cada serviço representa uma funcionalidade que é mapeada explicitamente a um passo no processo de negócio da corporação (GROVES, 2005).

- **Baixo Acoplamento**

  Acoplamento faz referência ao número de dependências entre os módulos. Módulos com baixo acomplamento possuem poucas dependências, bem conhecidas. A SOA promove o baixo acomplamento entre os consumidores e os provedores dos serviços e a idéia de terem apenas poucas e bem conhecidas dependências entre eles (MCGOVERN *et al.*, 2003).

- **Interfaces com Granularidade Grossa**

  O conceito de granularidade se aplica a serviços de dois modos: Primeiro, ela se aplica ao escopo do domínio de tudo que o serviço implementa; Segundo, ela se aplica ao escopo do dominio de cada método da interface que o serviço implementa. O grau apropriado de granularidade de um serviço e seus métodos é relativamente grossa, pois, geralmente, um serviço corresponde a um conceito único e distinto do negócio que pode ser reutilizado em larga escala (MCGOVERN *et al.*, 2003).

- **Agregabilidade**

  Serviços podem invocar outros serviços. Isto é, muitos tipos de funcionalidades de negócio podem ser divididas em passos me-

> nores, que também são serviços. Isso ocorre pois um sistema baseado na SOA conterá, muitas vezes, uma taxonomia de serviços a fim de categorizá-los de acordo com seus objetivos (ERL, 2005).

Atualmente, existem várias tecnologias que implementam SOA. Entre elas estão os *Web Services*, que garantem a interoperabilidade através do uso de um padrão de protocolos baseados na XML. Este é um dos motivos pelos quais os *Web Services*, na maioria dos casos, são a escolha na utilização da SOA nas corporações (ROSHEN, 2009).

2.1.3 SOA e Web Services

Grande parte das organizações possuem aplicações que são fragmentadas em infra-estruturas distintas, com um vasto número de aplicações clientes que rodam em diferentes plataformas. Neste contexto, os *Web Services* podem facilitar a integração entre sistemas de *software* heterogêneos. Além disso, como mencionado anteriormente, a SOA é um tipo de arquitetura focada no design das interfaces dos serviços. Essa característica facilita a integração de aplicações através de contratos entre os consumidores e provedores de serviços.

Os *Web Services* são sistemas de *software* desenvolvidos para suportar a interoperabilidade de sistemas pela rede. Essa interoperabilidade é obtiva através de um conjunto de padrões baseados na XML e no XML *Schema*, como o Web Services Description Language (WSDL), o Simple Object Access Protocol (SOAP) e o UDDI. Tais padrões provêm uma abordagem comum para a definição, publicação e utilização de *Web Services* (MAHMOUD, 2005). Portanto, SOA é normalmente utilizada através dos *Web Services*.

A interoperabilidade alcançada com o uso dos *Web Services* só é possível pois os padrões utilizados são definidos por uma organização da indústria, a Web Services Interoperability Organization (WS-I). A WS-I estabelece as melhores práticas para a interoperabilidade de *Web Services* e orienta empresas e desenvolvedores a construirem *Web Services* interoperáveis (JÚNIOR, 2008).

A Figura 3 demonstra como os *Web Services* implementam o paradigma "*find-bind-execute*" da SOA utilizando as tecnologias definidas pela WS-I:

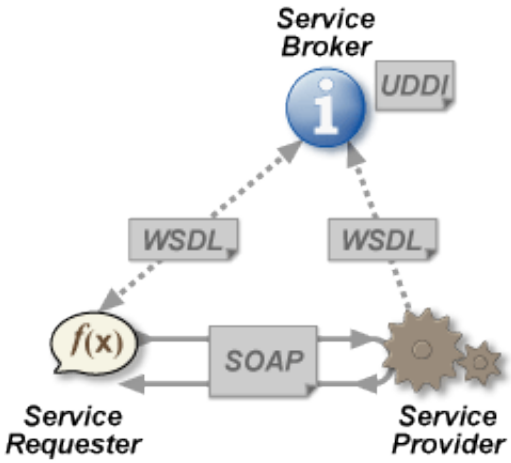

Figura 3: *Web Services* - Implementação do paradigma *find-bind-execute*

2.2 eXtensible Markup Language (XML)

A XML é baseada em regras simples, independentes de plataforma, para representação de informação textual estruturada. Por ser independente de plataforma, a XML se torna um formato ideal para troca de informações entre aplicações diferentes, permitindo assim a interoperabilidade (VOHRA; VOHRA, 2006).

Conforme mencionado na seção anterior, os *Web Services* e suas tecnologias são construídos com base na XML. Ela é utilizada para fornecer a descrição, o armazenamento e o formato da transmissão para a troca de dados (MAHMOUD, 2005).

A Figura 4 demonstra como um documento XML é formado e A Figura 5 mostra um exemplo de documento XML.

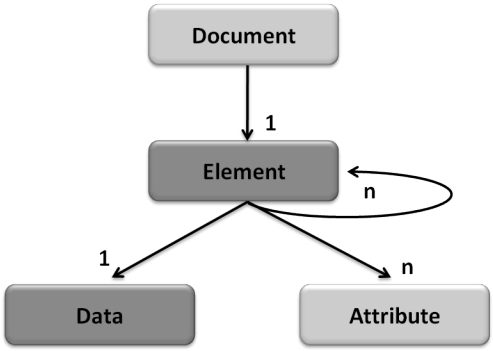

Figura 4: Estrutura de um documento XML

Antes de explicar os elementos que compõem um documento XML, é importante ter em mente que todas as construções sintáticas dentro de um documento XML são delimitadas pelos caracteres '<' e '>' e que sequências desses caracteres não são permitidas.

```
<?xml version='1.0' ?>
<livros>
   <livro isbn=''12345''>
      <nome>Livro 1</nome>
      <editora>Editora 1</editora>
      <ano>2009</ano>
   </livro>

   <livro isbn=''54321''>
      <nome>Livro 2</nome>
      <editora>Editora 2</editora>
      <ano>2010</ano>
   </livro>
</livros>
```

Figura 5: Exemplo de documento XML

2.2.1 Declarações XML

Um documento XML válido pode começar com uma declaração XML. Uma declaração XML pode ser omitida, mas caso ela apareça, deve ser a primeira coisa dentro do documento XML. Uma declaração XML é criada como segue:

```
<?xml version='1.0' ?>
```

O atributo *version* especifica a versão do XML e é um atributo obrigatório. Uma declaração XML pode incluir os seguintes atributos adicionais: *encoding* e *standalone*, como segue:

```
<?xml version='1.0' encoding='UTF-8' standalone='yes' ?>
```

O atributo *encoding* especifica a codificação usada nos caracteres dos dados em um documento XML. O valor padrão desse atributo é 'UTF-8' - 8-bit Unicode Transformation Format (UTF-8). O atributo *standalone* especifica se o documento XML referencia entidades externas. Se não houver referências externas, deve ser atribuido o valor "yes" ao atributo.

2.2.2 Elementos

A construção sintática básica de um documento XML é um elemento, que é delimitado por uma *tag* inicial e uma *tag* final. Um exemplo de elemento XML é o seguinte:

```
<autor></autor>
```

A tag inicial de um elemento é delimitada pelos caracteres '<' e '>' e um nome entre eles. No exemplo anterior, o nome é "autor". A tag final de um elemento é delimitada por '</' e '>' e possui o mesmo nome entre eles.

Um documento XML deve ter um único elemento pai, conhecido como *document element*.

Um elemento pode conter outros elementos aninhados, como mostrado na Figura 5, onde o elemento "nome" está dentro do elemento "livro". Pode-se perceber também que elementos podem possuir conteúdo texto entre suas tags inicial e final.

Elementos vazios, ou que não possuem texto entre as tags inicial e final, podem ser declarados da seguinte forma simplificada: <livro />. Essa representação é o mesmo que fazer <livro></livro>.

2.2.3 Comentários

Comentários podem aparecer em qualquer lugar fora dos caracteres '<' e '>' de um documento XML e são definidos como no exemplo abaixo:

```
<!--Exemplo de comentario -->
```

2.3 XML Schema

O XML *Shema* especifica a estrutura de um documento XML e limita seu conteúdo, criando classes de documentos XML válidos. Um documento XML válido é formalmente referenciado como sendo uma instância de um documento *Schema*. Numa analogia grosseira, o que uma classe Java é para um objeto Java, um *schema* é para um documento XML (VOHRA; VOHRA, 2006).

É importante ressaltar que um *Schema* é também um documento XML e através dele definem-se os elementos e atributos que podem existir em um documento XML. Define-se também a ordem e o número de elementos que são elementos filhos. Além disso, pode-se demarcar se um elemento deve ser vazio ou conter texto, os tipo de dados (precedidos por ”xsd:”) e os valores padrões dos elementos e atributos do documento XML (SCHOOLS, 2010).

A Figura 6 mostra um exemplo de arquivo XML *Schema* e na Figura 7 é mostrado um documento XML que utiliza o exemplo.

```
<?xml version=''1.0'' encoding=''UTF-8''>
<xsd:schema xmlns:xsd=''http://www.w3.org/2001/XMLSchema''
   targetNamespace=''http://simple.example.com/CInfoXmlDoc''
    xmlns=''http://simple.example.com/CInfoXmlDoc
    elementFormDefault=''qualified''
   <xsd:complexType>
      <xsd:sequence>
         <xsd:element name=''Name'' type=''xsd:string''/>
         <xsd:element name=''Address''>
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name=''Street''
                      type=''xsd:string''/>
                   <xsd:element name=''City''
                     type=''xsd:string'' />
                  <xsd:element name=''State''
                      type=''xsd:string'' />
                  <xsd:element name=''Country''
                      type=''xsd:string'' />
               </xsd:sequence>
            </xsd:complexType>
          </xsd:element>
         <xsd:element name=''HomePhone'' type=''xsd:string'' />
         <xsd:element name=''Email'' type=''xsd:string'' />
      </xsd:sequence>
   </xsd:complexType>
 </xsd:schema>
```

Figura 6: Exemplo de um documento XML *Schema*

```
<?xml version='1.0' encoding='UTF-8'?>
<ContactInformation
 xmlns='http://simple.example.com/CInfoXmlDoc'
 xmlns:xsi:'http://www.w3.org/2001/XMLSchema-instance'
 xsi:schemaLocation='http://simple.example.com/CinfoXmlDoc
 file:./CInfoXmlDoc.xsd'>
   <Name>John Smith</Name>
    <Address>
      <Street>45 Walcut St</Street>
      <City>Dublin</City>
      <State>Ohio</State>
      <Country>USA</Country>
   </Address>
   <HomePhone>9891234567</HomePhone>
   <Email>xyz@abc.com</Email>
</ContactInformation>
```

Figura 7: Exemplo de documento XML que é instância de um XML *Schema*

A Tabela 2.3 mostra os tipos mais utilizados no XML *Schema*:

Tipo de Dado	Descrição	Exemplo
string	Uma palavra	Lavras
int	–2147483648 a 2147483647	-10
double	Número ponto flutuante (64-bit)	-345.e-7
decimal	Um número decimal válido	-40.9
date	Uma data no formato YYYY-MM-DD	2010-05-05
time	Tempo no formato hh:mm:ss-hh:mm	10:30:24-03:00

Tabela 1: Tipos mais utilizados no XML *Schema*

O elemento pai de um *schema* é chamado ”*schema*”. O *namespace* para a definição de um *schema* é ”http://www.w3.org/2001/XMLSchema”, que é ligado ao prefixo XML Schema Definition (XSD). O *namespace* citado é usado convencionalmente para denotar definições de XML *Schemas*, porém qualquer outro prefixo pode ser usado (ROSHEN, 2009).

O elemento "*schema*" pode conter vários tipos de elementos subordinados (ROSHEN, 2009), listados abaixo. A relação entre eles e o elemento pai "*schema*" é mostrado na Figura 8.

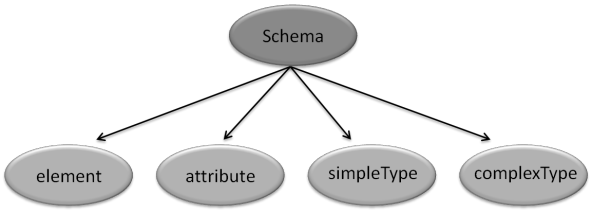

Figura 8: Elementos básicos de um XML *Schema*

- **element**. Esse sub-elemento declara um elemento usado em um documento XML instância do esquema. A declaração inclui o nome e o tipo do elemento;

- **attribute**. Esse sub-elemento declara um atributo usado em um documento XML instância do esquema. A declaração inclui o nome o tipo do atributo;

- **simpleType**. Esse sub-elemento define um tipo simples. Um tipo simples em um XML *Schema* é usado para adicionar restrições customizadas aos elementos. Caso um documento XML, instância do *schema*, utilize um tipo inválido, o documento XML não será válido;

- **complexType**. Esse sub-elemento define um tipo complexo. Um tipo complexo normalmente contem outros elementos XML e atributos. Ele define a ordem, o tipo e a cardinalidade entre o elemento complexo e seus sub-elementos;

- **include**. Esse sub-elemento é utilizado para importar uma definição de um elemento declarada em outro *schema*, que deve pertecenter ao mesmo XML *namespace*;

- **import**. Esse sub-elemento é utilizado para importar uma definição de um elemento declarada em outro *schema* pertencente a um *namespace* diferente.

Percebe-se que através do uso dos sub-elementos *include* e *import* obtem-se o reúso com definições XML *Schema* (ROSHEN, 2009).

Dos sub-elementos apresentados, o mais importante é o *complexType*. A estrutura desse sub-elemento é mostrada na Figura 9. O complexType pode ter vários atributos mas apenas um dos itens a seguir: *sequence*, *all*, *choice* ou *group*. Ao utilizar o *sequence*, pode-se ter um número indeterminado de elementos, que devem seguir a ordem declarada no XML *Schema*. Ao utilizar o *all*, pode-se ter um número indeterminado de elementos, fora de ordem. A única restrição ao utilizar o *all* é que todos devem aparecer. Ao utilizar o *choice*, apenas um dos elementos declarados devem aparecer no documento XML, instância do *Schema*. Um exemplo do uso do sub-elemento *complexType* é mostrado na Figura 6 (SCHOOLS, 2010).

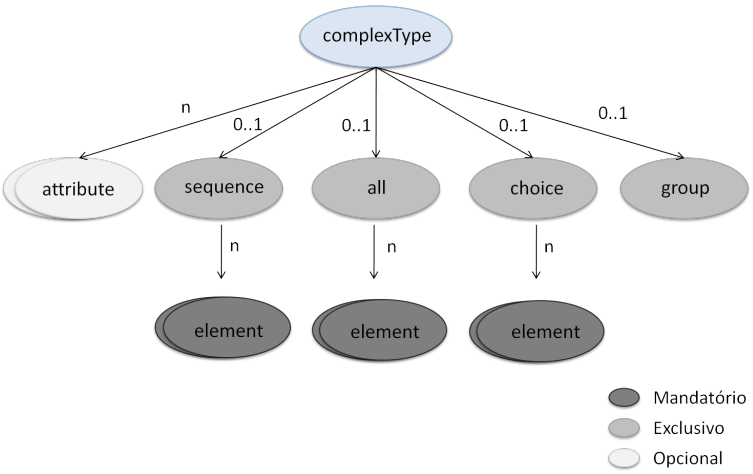

Figura 9: Estrutura do sub-elemento complexType do XML *Schema*

2.4 Java Architecture for XML Binding (JAXB)

A tecnologia Java, muito utilizada pelas empresas por prover uma plataforma que permite que aplicações portáveis sejam desenvolvidas, pode ser utilizada juntamente com a XML para criar aplicações interoperáveis. Essa parceria é particularmente importante para os *Web Services* que, conforme visto, fornecem formas de expor funcionalidades de suas aplicações através da web. Diante disso, Java e XML são reconhecidas como tecnologias ideais para o desenvolvimento de *Web Services* e de aplicações que acessam *Web Services* (MCLAUGHLIN, 2002).

Uma API chamada Java Architecture for XML Binding (JAXB) pode facilitar o acesso a documentos XML às aplicações desenvolvidas em Java. A JAXB permite que desenvolvedores Java acessem e processem arquivos XML sem ter conhecimento sobre como tal processamento é feito (MCLAUGHLIN; EDELSON, 2006). Para isso, a JAXB fornece um modo conveniente de ”traduzir” dados representados em XML para Java e vice-versa. A Figura 10 mostra os componentes que fazem parte da implementação da JAXB.

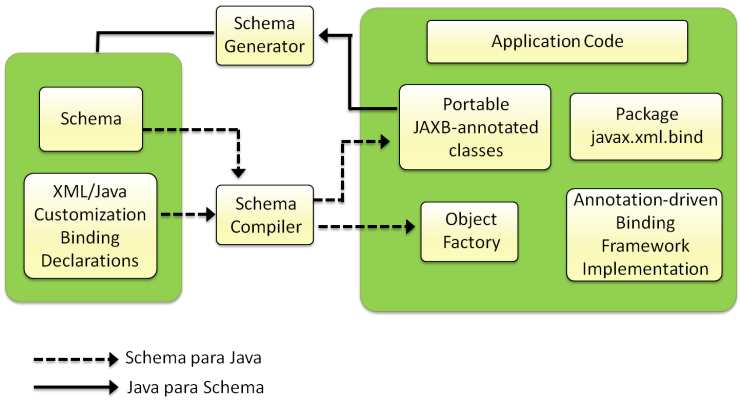

Figura 10: Elementos que compõem a arquitetura da API JAXB

Uma implementação da JAXB consiste dos seguintes componentes (MCLAUGHLIN; EDELSON, 2006):

- **Compilador *Schema***. Através de um documento *Schema*, é capaz de gerar classes correspondentes.
- **Generator *Schema***. Através de classes, é capaz de gerar documentos XML *Schemas* correspondentes.
- **Framework de ligação em tempo de execução**. Provê, através das funcionalidades de *Unmarshal* e *Marshal* (explicadas mais adiante neste documento), operações para acesso, manipulação e validação de documentos XML e objetos Java relacionados.

Resumidamente, os passos necessários para ligação entre documentos XML e objetos Java (Figura 11) são os seguintes, em ordem (VOHRA; VOHRA, 2006; MCLAUGHLIN, 2002; MCLAUGHLIN; EDELSON, 2006):

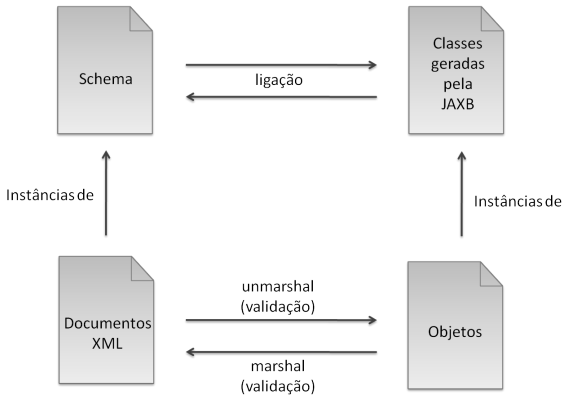

Figura 11: Demonstra o processo de ligação entre XML e Java feito pela API JAXB

Passo 1 - Geração das Classes através de XML *Schemas*. Um XML *Schema* é utilizado como entrada para o Compilador para a geração de classes Java correspondentes;

Passo 2 - Compilação das Classes. Todas as classes geradas no passo anterior devem ser compiladas;

Passo 3 - *Unmarshal*. Nome dado ao processo de transformação de documentos XML (criados seguindo os *Schemas* citados no passo 1) em objetos, instâncias das classes geradas e compiladas anteriormente (Figura 12).

Passo 4 - Utilização dos Objetos. Na aplicação, os objetos criados no passo anterior podem ser alterados e utilizados como qualquer outro objeto;

Passo 5 - *Marshal*. Nome dado ao processo de transformação de objetos Java em seu documento XML correspondente (Figura 13). Como o objeto pode ter sido alterado de várias maneiras, uma validação pode ser aplicada a este passo para garantir que o XML gerado seja válido, de acordo com as restrições impostas pelo *Schema* que ele segue.

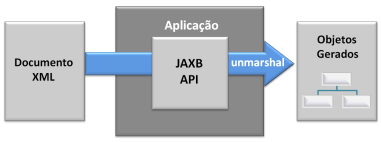

Figura 12: *Unmarshal* da JAXB - Processo de transformação de documentos XML em objetos Java

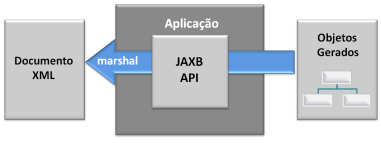

Figura 13: *Marshal* da JAXB - Processo de transformação de objetos Java em documentos XML

2.5 Java Web Start

Java Web Start é uma tecnologia desenvolvida para invocar aplicações baseadas em Java com o navegador. Quando um usuário clica em um link que referencia um arquivo Java Network Lauching Protocol (JNLP), o navegador invoca o *Java Web Start*, que automaticamente realiza o download e executa a aplicação utilizando o Java Runtime Environment (JRE) instalada na máquina do usuário, conforme demonstra a Figura 14 (SUN, 2004).

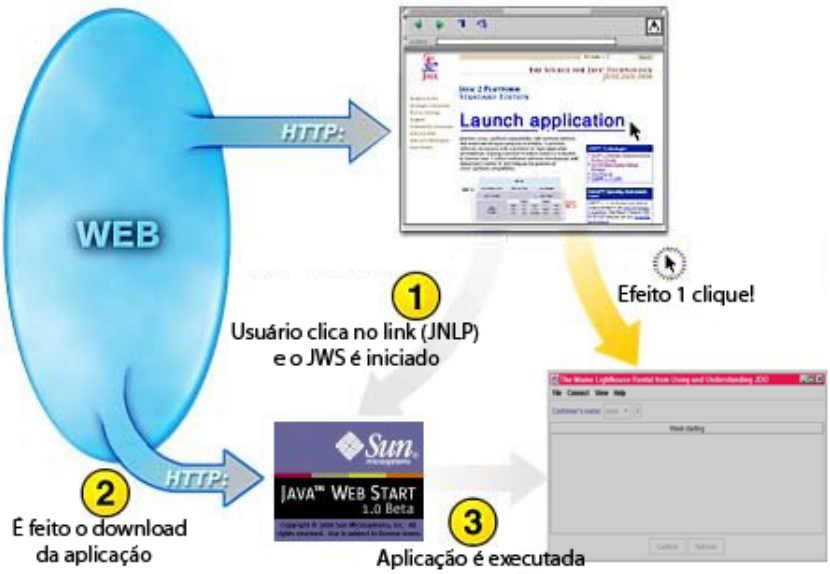

Figura 14: Visão geral do Java Web Start

A tecnologia por trás do *Java Web Start* é o JNLP. Esta tecnologia, desenvolvida atualmente pela Java Community Process (JCP), permite que uma aplicação possa ser executada em uma máquina cliente usando recursos hospedados em um servidor remoto. A especificação JNLP define, entre outras coisas, um formato padrão de arquivo, com extensão ".jnlp", que descreve como executar uma

aplicação. As tecnologias *Java Web Start* e *Java Plug-in* são consideradas implementações da especificação JNLP.

A *Java Web Start* oferece uma gama de benefícios que a tornam interessante (SUN, 2004):

- Foi construida para executar aplicações desenvolvidas para a plataforma Java. Portanto, uma única aplicação pode ser hospedada em um servidor e executada em diversas plataformas distintas. A plataforma Java vem provando ser uma plataforma robusta e produtiva, minimizando os custos com o desenvolvimento e testes das aplicações;

- Suporta o uso de diferentes versões da plataforma Java, realizando o download e a instação da versão requerida pela aplicação de forma transparente para o usuário;

- Permite que aplicações sejam executadas indepentende do navegador utilizado e em caso da impossibilidade do uso de um navegador para executar a aplicação, ela poderá ser executada através de atalhos no *desktop* do usuário;

- Por rodar sobre a plataforma Java, automaticamente garante que a aplicação rodará em um ambiente seguro, com acesso restrito ao disco local e a recursos de rede. Com isso, o usuário poderá executar de forma segura aplicações vindas de fontes não-confiáveis;

- As aplicações, uma vez executadas, são mantidas em um cache local, garantindo que o usuário somente realizará o download novamente caso uma nova versão da aplicação esteja disponível.

Conforme mencionado anteriormente, com o uso do *Java Web Start*, uma aplicação desenvolvida em Java pode ser executada a partir de um navegador. Para

isso, basta que um *link* apontando para uma aplicação JNLP exista e que o usuário clique sobre ele. O *link* citado é um *link* HyperText Markup Language (HTML) padrão. Entretanto, ao invés de apontar para outra página web, ele aponta para um arquivo com extensão ".jnlp". O navegador do usuário examina a extensão do arquivo e verifica que arquivos desse tipo devem ser executados pelo *Java Web Start*. Finalmente, o *Java Web Start* é invocado pelo navegador e realiza o download, armazena a aplicação em um cache local e a executa, seguindo as instruções do arquivo JNLP. Um exemplo de *link* utilizado para executar a aplicação "teste.jnlp" seria:

```
<a href=http://www.empresa.com.br/teste.jnlp>Executar aplicativo</a>
```

Para que a aplicação Teste, do exemplo anterior, seja executada na máquina do usuário, é necessário que o *Java Web Start* esteja instalado em sua máquina.

2.6 Java Native Interface (JNI)

A JNI é uma API bastante poderosa existente na plataforma Java. Com ela, aplicações java podem incorporar código nativo, escritos em linguagens como C e C++, de forma fácil. Da mesma forma, é possível utilizar funcionalidades de aplicações escritas em Java em códigos nativos. A JNI permite que desenvolvedores tirem vantagem do poder da plataforma Java, sem que abram mão de seus códigos nativos (LIANG, 1999).

Antes de explicar o funcionamento da JNI e sua utilização, é importante ressaltar as diferenças entre os ambientes de aplicações nativas, escritas em linguagens como C e C++, e de aplicações escritas em Java. A plataforma Java é um ambiente que consiste da Java Virtual Machine (JVM) e da API Java. Aplicações Java são escritas utilizando a linguagem de programação Java e compiladas para um formato binário (".class"), independente de plataforma. Este binário pode ser executado em qualquer implementação da JVM. Esta característica garante a portabilidade de aplicações Java, bastando que o ambiente usado para executá-las possua uma implementação da JVM instalada (GOSLING *et al.*, 2005).

Por outro lado, aplicações nativas são escritas em linguagens de programação nativas, compiladas em um código binário específico e ligadas a bibliotecas nativas do ambiente onde são executadas. Uma aplicação escrita em C para um sistema operacional específico geralmente não irá funcionar em outros sistemas operacionais (GOSLING *et al.*, 2005).

Diante disso, a motivação da construção da JNI surgiu da necessidade de estreitar a comunicação entre aplicações Java, que rodam sobre a Plataforma Java, e aplicações nativas. A JNI foi criada para que desenvolvedores possam aproveitar os benefícios da plataforma Java e, mesmo assim, utilizar códigos nativos escritos em outras linguagens. A Figura 15 mostra uma visão geral da JNI.

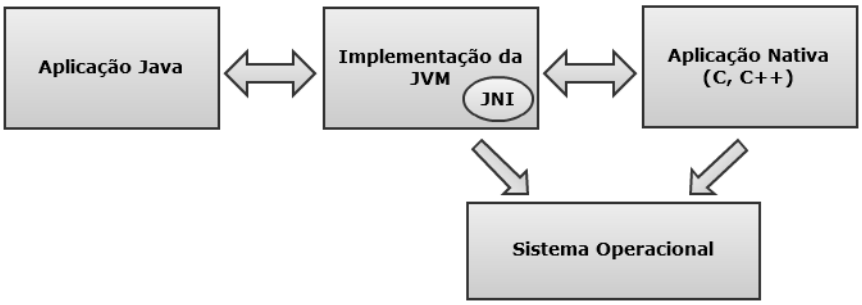

Figura 15: Visão geral da Java Native Interface (JNI)

Conforme mencionado, a JNI foi criada para ajudar em situações onde é necessário combinar aplicações Java com código nativo e suporta tanto a utilização de código nativo em aplicações Java como de aplicações Java em códigos nativos. Pode-se usar a JNI para escrever métodos nativos que permitem que aplicações Java invoquem funções implementadas em código nativo. Aplicações Java invocam métodos nativos da mesma forma que invocam métodos implementados na linguagem Java. Entretanto, de forma transparente, métodos nativos são implementados em outra linguagem e residem em bibliotecas nativas.

A JNI também fornece uma interface para invocação de funcionalidades da Máquina Virtual Java em códigos nativos. Aplicações nativas podem ser ligadas a uma biblioteca nativa (que implementa a Máquina Virtual Java) e, então, invocar a interface mencionada para acessar funcionalidades de aplicações escritas na linguagem de programação Java. Por exemplo, um navegador escrito em C pode executar aplicações Java chamadas de "*applets*"[3] através de uma implementação embutida da máquina virtual Java.

Ao utilizar a JNI, é importante saber que pode-se estar abrindo mão de duas vantagens da plataforma Java. Primeiro, conforme mencionado, aplicações

[3] Applet é um *software* aplicativo que é executado no contexto de outro programa (como por exemplo um web browser)

Java são portáveis e aplicações nativas não. Ao utilizá-las em conjunto, é possível que a aplicação nativa utilizada limite os ambientes nos quais elas poderão ser executadas. Segundo, sabe-se que a linguagem Java foi criada para ter tipagem forte e segura, o que não permite que programas acessem a memória de maneira inapropriada. Por outro lado, aplicações nativas podem não ter tal caracteristica, tornando a aplicação menos segura (LIANG, 1999).

Conhecendo as situações onde a utilização da JNI é apropriada e os riscos de sua utilização, os passos necessários para utilizá-la são apresentados a seguir, de forma resumida, através da Figura 16. Percebe-se que é fácil integrar um *software* já existente, desenvolvido em C, com uma aplicação escrita em Java. Para tanto, o arquivo ".c" criado no passo 4 deve invocar o sistema existente para usufruir das funcionalidades disponíveis. Dessa forma, a aplicação escrita em Java tem acesso às funcionalidades do sistema existente sem que este sofra alterações.

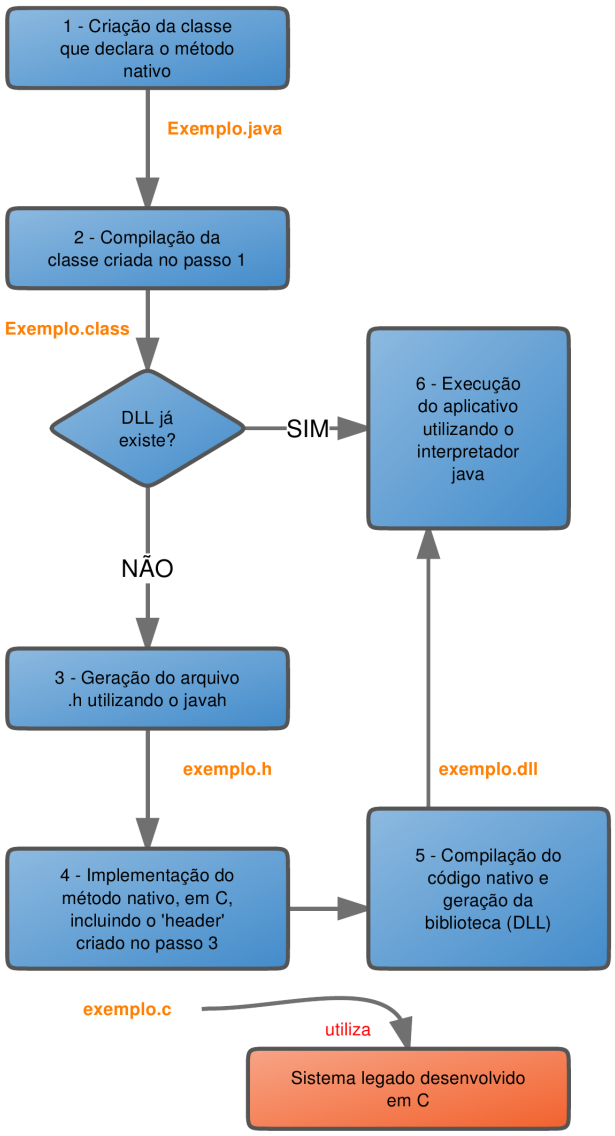

Figura 16: Demonstração dos passos necessários na utilização da JNI para invocar funções em código nativo

2.7 O Dispositivo Coletor de Dados

O coletor de dados utilizado neste trabalho é um equipamento móvel e configurável capaz de se adequar como coletor de atividades em qualquer elo de qualquer cadeia produtiva. Ele é levado pelo operador preso ao punho para realizar leituras de códigos de barras, que em determinada sequência, representam uma atividade. Além disto, junto às atividades registradas, o equipamento registra também data, hora e posição utilizando um Global Positioning System (GPS) (SAÚDE *et al.*, 2008a; SAÚDE *et al.*, 2008b).

O equipamento tem o foco na usabilidade, permitindo que as mãos do operador fiquem livres sem que outro recurso humano seja necessário para registrar as atividades. Além disso, guia o operador em procedimentos previstos e não necessita de interação humana, garantindo a autenticidade dos dados. Dessa forma, o equipamento permite que o operador colete dados detalhados, mesmo em situações complexas.

Um requisito importante, na maioria das cadeias produtivas, é que as atividades coletadas nas etapas do processo produtivo não sejam alteradas em hipótese alguma. Ou seja, os dados devem ter sua integridade garantida. O equipamento atende tal requisito implementando formas de garantir a integridade dos dados, como por exemplo *checksum* [4]. Além disso, implementa criptografia de *hardware*, fazendo com que a informação fique ilegível no caso de um acesso indevido.

O modelo de uso do equipamento é descrito a seguir (SAÚDE *et al.*, 2008a; SAÚDE *et al.*, 2008b):

[4] Checksum ou soma de verificação é um código usado para verificar a integridade de dados transmitidos através de um canal com ruídos ou armazenados em algum meio por algum tempo (COHEN, 1987).

Configurar o Equipamento

- Descrever o processo produtivo (feito por técnicos especialistas do setor e na linguagem de domínio destes).
- Transcrever a descrição acima para uma linguagem formal (feito por técnico conhecedor da linguagem formal de descrição de processos).
- Carregar a descrição formal no equipamento.

Coletar Dados

- Ler códigos identificadores e sensores (feito pelo operador do equipamento).
- Identificar e registrar atividades realizadas (feito automaticamento pelo equipamento).

Um exemplo de registro de atividade definida na etapa de configuração:

Considere a atividade ”Inserir quadro na colméia”. Essa atividade é definida como “COL,QUA”, onde COL é o código da classe de colméia e QUA é o código da classe de quadro. Ou seja, a atividade é definida pelo par: classe das colméias seguida da classe dos quadros, o maior elemento seguido do menor elemento. Quando o operador usa o equipamento para ler o código de barras de uma colméia e em seguida ler o código de barras de um quadro, o equipamento identifica e registra a atividade ”Inserir quadro na colméia” automaticamente.

Indo além do exemplo apresentado, o equipamento permite combinações mais complexas de classes de identificadores e a utilizaçao de códigos que identificam sensores. Por exemplo: A atividade ”medir temperatura de colméia” poderia ser definida pela seqüência ”COL,TMP”, onde COL é o código da classe de colméia e TMP é uma leitura do sensor de temperatura (SAÚDE *et al.*, 2008a).

A Figura 17 e a Figura 18 são fotos de apicultores utilizando o equipamento:

Figura 17: Apicultor consultando sua agenda de atividades no Apix

Figura 18: Apicultor utilizando o Apix durante atividades rotineiras

2.8 O Software Orientado a Processos de Negócio

Este trabalho propõe uma solução de integração entre um dispositivo coletor de dados e um software chamado Labora, que foi construído utilizando-se um *framework* para construção de aplicações orientadas a serviços, o Iguassu. Ao utilizá-lo, os desenvolvedores podem concentrar suas atenções nas necessidades do cliente e devem implementar apenas algumas partes do sistema, já que grande parte do código pode ser gerado e reaproveitado (SAÚDE *et al.*, 2010).

Conforme visto anteriormente, há dois principais componentes nas arquiteturas orientadas a serviços: O *Service Requester* e o *Service Provider* (JÚNIOR, 2008). O Iguassu trata das funcionalidades básicas do *Service Provider*, de tal forma que seja leve, simples e atenda boa parte dos aplicativos corporativos.

Todas as funcionalidades são implementadas na forma de serviços, que podem ser de acesso a dados (*Data Services*) ou que implementam regras de negócio (*Business Services*). Os *Business Services* podem, além de implementar regras de negócio, interagir com bibliotecas de terceiros, outros serviços ou até mesmo com processos de negócios. Neste caso, é inviável a implementação *hardcoded* de processos de negócio, visto que eles se alteram frequentemente. Diante disso, os *Business Services* devem se comunicar com um sistema Business Process Management Suite (BPMS) externo, que gerencia a execução do processo. A Figura 19 mostra essa interação(SAÚDE *et al.*, 2010).

O Iguassu também fornece um conjunto de classes que podem ser utilizadas para fazer requisições aos serviços disponíveis nas aplicações construidas sobre ele, facilitando a comunicação entre diferentes aplicações.

No Iguassu, cada requisição feita a um serviço é tratada por um encadeamento de objetos receptores. Cada um desses objetos cumpre seu papel e passa a requisição adiante, até que ela seja completamente consumida. Essa cadeia de ob-

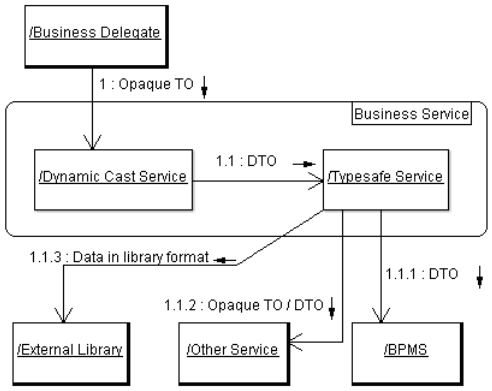

Figura 19: Demonstração dos estágios da execução de serviços de negócio (Business Services)

jetos com diferentes responsabilidades se dá no formato de uma cascata, mostrada na Figura 20 (SAÚDE *et al.*, 2010).

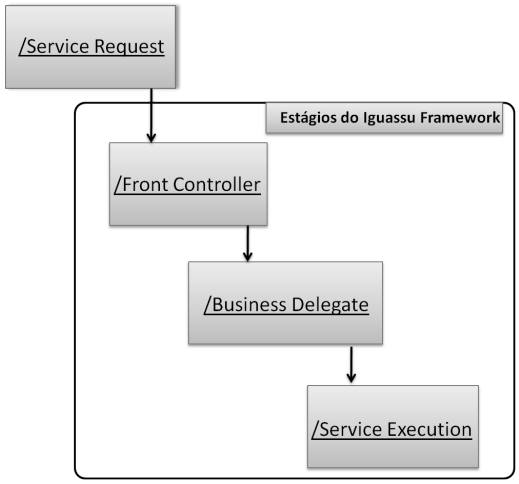

Figura 20: Os três principais objetos da cadeia de responsabilidades do Iguassu Framework

A Figura 21 fornece uma visão geral do funcionamento e dos componentes que compem o Iguassu e, portanto, das aplicações construídas sobre ele. Na figura, os elementos em branco são elementos externos. Os cinza-claros são ele-

mentos reutilizáveis, implementados apenas uma vez. Os componentes coloridos com um nível de cinza intermediário podem ser gerados automaticamente com as ferramenta de geração de código fornecida pelo *framework*. Finalmente, os componentes mais escuros são aqueles que devem ser implementados especificamente para as aplicações construídas sobre o Iguassu (SAÚDE *et al.*, 2010), como é o caso do software orientado a processos de negócio tratado neste trabalho, o Labora.

O Labora, em conjunto com o Apix, visa auxiliar na automação das atividades, na melhoria dos controles e consequente incremento da eficiência e produtividade apícola. Para isso, efetua-se por meio do hardware Apix a coleta de dados ao longo de toda a unidade produtiva. Em seguida, os dados são descarregados no Labora, onde é possível interagir com as principais informações acerca das ferramentas, materiais, embalagens e insumos. O Labora é também uma ferramenta para planejamento dos processos. É capaz, portanto, de administrar dados relativos a fornecedores, clientes e custos, além de mapear, registrar e informar a atividade diária dos funcionários envolvidos na produção, bem como as condições ambientais e outros detalhes técnicos da estrutura produtiva (VICTORIO *et al.*, 2008).

O registro e manutenção destes dados formam um conjunto de informações que podem ser reorganizadas por meio de relatórios específicos de acordo com a necessidade de cada ator de toda a cadeia produtiva apícola (produtor, industrial, distribuidor, varejista, exportador). Para o consumidor final, o Labora é capaz de fornecer, pela Internet, informações acerca das condições de produção e manejo do produto consumido. Em suma, o Labora, em conjunto com o Apix, realiza completamente a rastreabilidade de produtos apícolas na primeira etapa da cadeia produtiva. A solução Labora e Apix, funcionando juntos, é chamada de LaborApix (VICTORIO *et al.*, 2008).

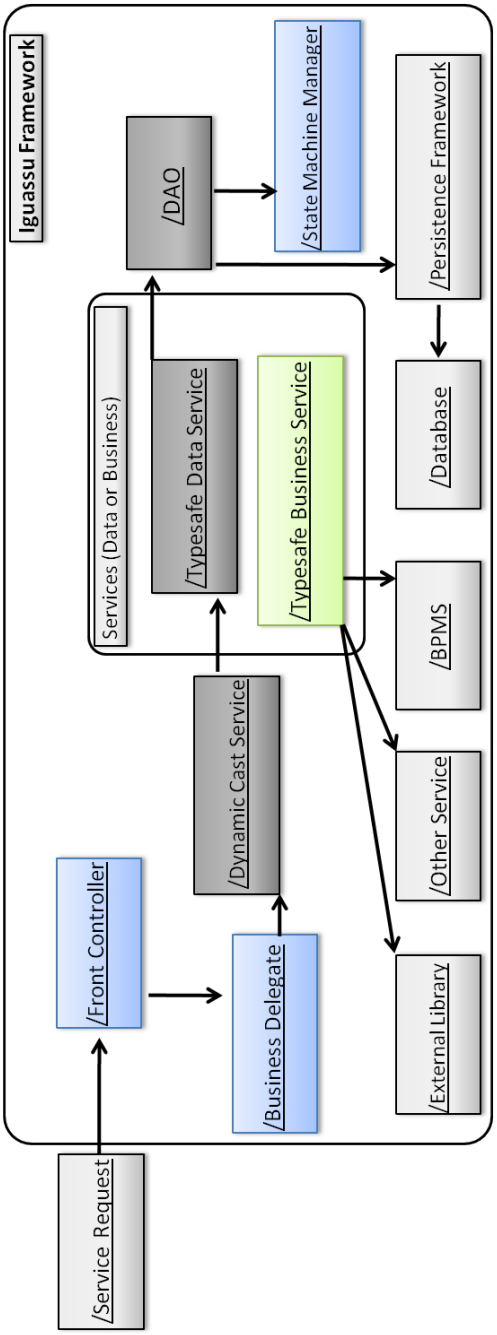

Figura 21: Visão geral do Iguassu *Framework*

3 Metodologia e Desenvolvimento

Este capítulo descreve o tipo de pesquisa e os procedimentos metodológicos adotados para o seu desenvolvimento. As atividades realizadas, bem como as técnicas utilizadas para alcançar os objetivos deste projeto, serão descritas de forma clara e completa. São apresentados diagramas, esquemas e figuras para facilitar o entendimento do leitor.

3.1 Tipo de Pesquisa

Conforme (JUNG, 2004) e (MARCONI, 2003), uma pesquisa pode ser classificada quanto a sua natureza, quanto aos seus objetivos, quanto aos seus procedimentos e quanto ao seu local de realização.

A presente pesquisa pode ser classificada: em sua natureza, como tecnológica já que o objetivo é a geração de uma solução de integração alcançada pelo desenvolvimento de produtos de *software*; quanto aos objetivos, trata-se de uma pesquisa de caráter exploratório, pois tem-se o foco no produto e em inovação tecnológica, utilizando-se de soluções existentes e baseando-se em estudos iniciais sobre o tema; quanto aos procedimentos, o trabalho é caracterizado como estudo de caso por lidar com um contexto local, onde realizou-se a integração entre o dispositivo de *hardware* e um *software* orientado a processos de negócio existentes; quanto ao seu local de desenvolvimento, trata-se de uma pesquisa realizada em laboratório, pois são utilizadas ferramentas que possibilitam a manipulação de variáveis que influenciam diretamente na pesquisa. Finalmente, quanto ao tempo de aplicação do estudo, trata-se de uma pesquisa com estudo longitudinal, realizada ao longo do período de outubro de 2009 a outubro de 2010.

3.2 Materiais

A principal ferramenta utilizada para desenvolvimento do projeto foi o Eclipse Java Platform, Enterprise Edition (Java EE) IDE para desenvolvedores Web, em sua versão denominada *Ganymede*. O Eclipse foi utilizado na criação dos arquivos XML, XML *Schemas*, da aplicação *Desktop* e dos serviços no Labora.

O aplicativo *Subversion* foi utilizado para realizar o controle de versão dos documentos, incluindo os códigos-fonte dos projetos, sendo que o Eclipse possui uma boa integração com o *Subversion*.

Para o desenvolvimento da camada de integração no aplicativo desenvolvido em C/C++, que controla o equipamento, a ferramenta *Microsoft Visual Studio 2010 Express* foi utilizada.

Todo o código-fonte referente aos sistemas manipulados no projeto estão armazenados em um servidor Linux, fisicamente localizado no Departamento de Ciência da Computação da Universidade Federal de Lavras e que pode ser acessado de qualquer computador conectado a Internet. Tal característica permitiu ao autor acessar os repositórios do *Subversion* remotamente, o que facilitou a realização de suas atividades.

3.3 Procedimentos Metodológicos

Nesta seção descreve-se os procedimentos metodológicos utilizados para desenvolver a solução de integração proposta por este trabalho. Para tal, demonstra-se como cada objetivo específico foi alcançado e, consequentemente, o objetivo geral. A metodologia proposta neste trabalho está estruturada em cinco etapas, executadas paralelamente: Estudo das tecnologias envolvidas; Criação do protocolo de comunicação; Criação dos serviços no sistema web (Labora) e Criação da aplicação *desktop*. Esta última é composta também pela criação de um *driver* com

a finalidade de ocultar detalhes do equipamento da aplicação *desktop* e pela criação de arquivos JNLP para permitir que a aplicação *desktop* fosse acionada, sob demanda, pelo usuário através de um navegador.

Este trabalho foi desenvolvido no laboratório de pesquisa do Departamento de Ciência da Computação da Universidade Federal de Lavras, sendo que algumas atividades foram realizadas remotamente.

3.3.1 Estudo das tecnologias envolvidas

Inicialmente, foi fundamental adquirir conhecimentos sobre os sistemas existentes, o equipamento e as tecnologias que seriam utilizados, tais como SOA, o Iguassu *Framework*, XML, XML *Schema*, JAXB, *Java Web Start* e JNI.

3.3.2 Criação do protocolo de comunicação

Nessa fase do projeto, objetivou-se a criação de um protocolo de alto nível e genérico o suficiente para possibilitar que a expansão da comunicação entre o Apix e o Labora não impactasse em grandes alterações no protocolo. Dessa forma, foram definidos e criados quatro XML *Schemas*. Um denominado ”apixgeneralconfig” para configurações gerais do equipamento, como por exemplo parâmetros do GPS, modo de operação, timeout, entre outros. A Figura 22 mostra os elementos existentes nesse esquema e suas relações. Outro denominado ”apixnavigation” para troca de informações de navegação do GPS, como data, latitude e longitude, mostrado na Figura 23. Outro denominado ”apixtaskconfig” para o registro de atividades no equipamento, que conforme explicado na seção 2.7, são identificadas por códigos de barra lidos sequencialmente(SAÚDE *et al.*, 2008a). A Figura 24 mostra os elementos existentes nesse esquema e suas relações. Finalmente, um de-

nominado ”apixtasks” para o envio das atividades realizadas em campo, mostrado na Figura 25.

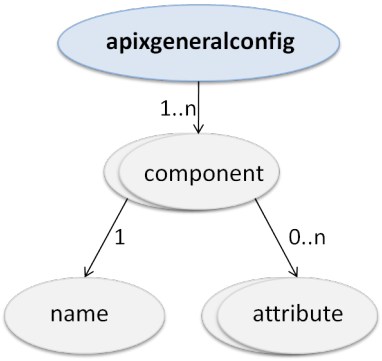

Figura 22: Componentes do XML Schema apixgeneralconfig. Ele foi criado de forma a suportar a adição e remoção de configurações. O Labora envia essas informações para o Apix

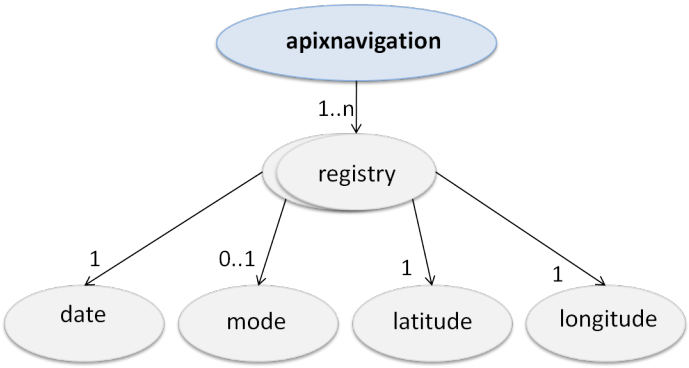

Figura 23: Componentes do XML *Schema* apixnavigation. Vários registros com informações de data, latitude e longitude podem ser inseridos. O Apix envia essas informações ao Labora

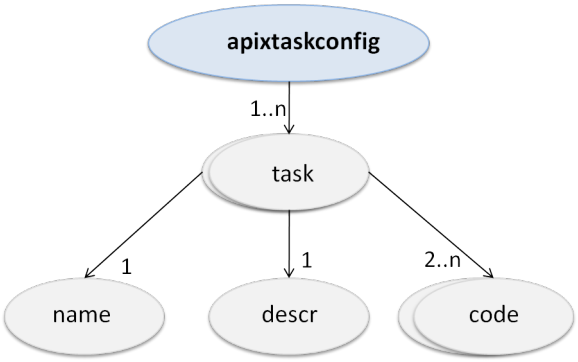

Figura 24: Componentes do XML *Schema* apixgtaskconfig, utilizado para cadastrar atividades no equipamento. O Labora envia essas informações para o Apix

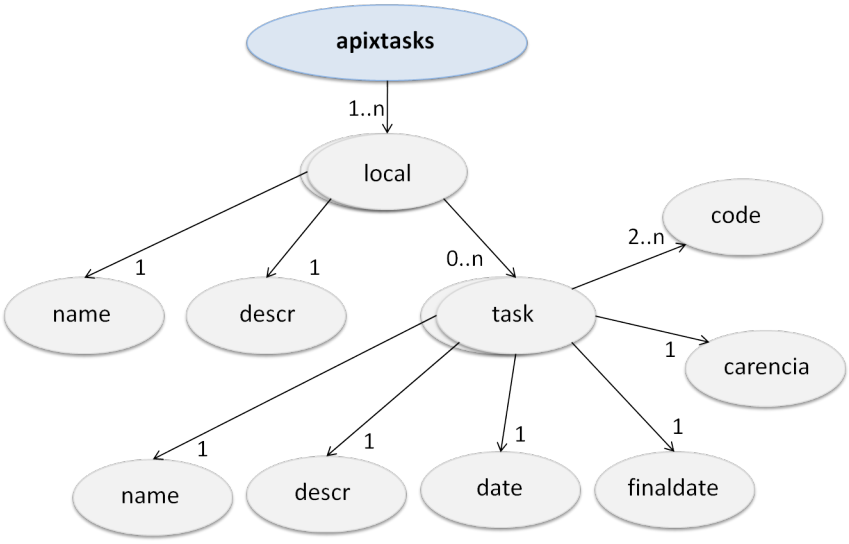

Figura 25: Componentes do XML *Schema* apixtasks, utilizado para armazenar informações de registro de atividades realizadas pelos apicultores. Tais informações são enviadas do Apix para o Labora

Existem duas maneiras de utilizar esse protocolo. Uma abordagem consiste em utilizá-lo na aplicação *desktop* e no Labora, de tal forma que os dados trafegados entre elas fiquem no formato XML e sejam gerados pela API JAXB. A segunda abordagem é fazer com que a aplicação *desktop* apenas repasse os pacotes recebidos do Apix para o Labora, onde ocorreria a tradução dos dados para o protocolo, utilizando-se um *driver* que sabe como realizar a tradução dos *bytes* recebidos do equipamento para os objetos do protocolo criado. Nessa última abordagem, a aplicação *desktop* possui apenas o papel de uma ponte entre o equipamento e o sistema web.

Neste projeto utilizou-se a primeira abordagem. As vantagens e desvantagens de cada abordagem serão discutidas no Capítulo 4.

Dando sequência na abordagem escolhida, já com o protocolo bem definido e com os XML *Schemas* criados, o próximo passo foi utilizar a API JAXB para ”traduzir” dados representados em XML para Java e vice-versa, garantindo que as restrições impostas pelo protocolo fossem atendidas no processo de validação do *marshal* e *unmarshal*, explicados no referencial teórico. Assim, nenhuma das aplicações manipula documentos XML diretamente. Elas importam um arquivo ”.jar” com as classes correspondentes, geradas pelo JAXB a partir dos XML *Schemas* (que definem o protocolo), e utilizam a API para criar os documentos XML e vice-versa, como mostrado na Figura 26.

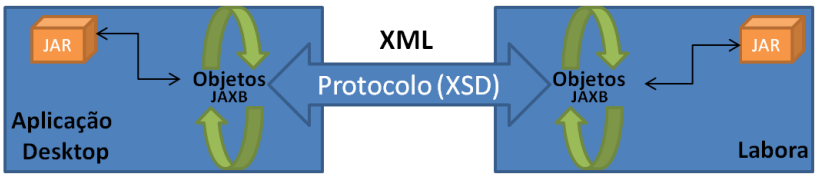

Figura 26: Comunicação entre aplicação desktop e Labora utilizando a primeira abordagem citada pelo trabalho. Elas utilizam a API JAXB para realizar *marshal* e *unmarshal* dos objetos e, portanto, nenhuma das aplicações lida diretamente com documentos XML

A Figura 27 demonstra a comunicação entre a aplicação *desktop* e o Labora na segunda abordagem citada, facilitando seu entendimento. Nela, o ".jar" utilizado pelo Labora é diferente do da primeira por possuir detalhes do equipamento e ser responsável por traduzir os *bytes* recebidos em objetos do protocolo. Ressalta-se ainda que na primeira abordagem, o Labora desconhece detalhes do equipamento pois os dados recebidos já estão no formato do protocolo criado, pois foram traduzidos pelo ".jar" utilizado na aplicação *desktop*.

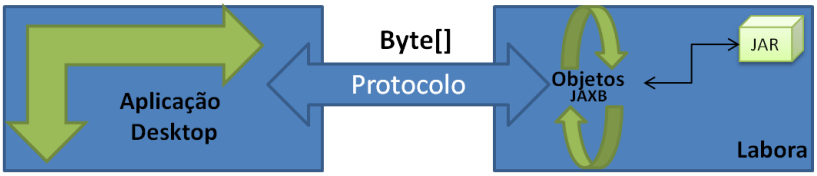

Figura 27: Comunicação entre aplicação desktop e Labora utilizando a segunda abordagem citada pelo trabalho. A aplicação desktop tem o papel de ponte entre o Apix e o Labora, sendo que o processo de tradução dos bytes vindos do equipamento para os objetos definidos no protocolo acontece no Labora, utilizando-se um driver

3.3.3 Criação dos serviços no sistema web

Conforme visto, o Iguassu é um *framework* que implementa SOA e entender partes desse tipo de arquitetura se fez necessário nessa etapa do projeto, já que a comunicação entre o Labora e aplicação *desktop* só é possível mediante a criação de serviços no sistema web.

Foram criados dois serviços: o "ApixUploadAppService" e o "ApixDownloadAppService". O primeiro é reponsável por receber e processar requisições feitas pela aplicação *desktop*, que envia os dados lidos do Apix. O serviço extrai as informações necessárias, cria instâncias de classes do domínio do Labora, inserindo em seus atributos os dados das informações extraídas correspondentes. Finalmente, o serviço invoca um outro serviço que salva ou atualiza as atividades

no banco de dados da aplicação Labora. No momento da persistência dos dados, gatilhos são disparados e tramitam processos de negócios associados através da execução de um outro serviço, que se comunica com uma aplicação BPMS externa, conforme visto na seção 2.8 do referencial teórico. A Figura 28 demonstra a maneira como os objetos envolvidos colaboram e se comunicam ao longo do tempo na execução do serviço ”ApixAppUploadService”.

Viu-se que uma atividade é identificada por uma sequência de códigos de barra e um nome. O objeto ”InstanciaAtividade” representa uma instância de uma atividade realizada pelo apicultor. Ela contém, além do nome e da sequência de códigos, informações sobre o recurso humano realizador, data e hora da execução, ferramentas utilizadas, etc. A Figura 29 mostra o diagrama de estados do objeto ”InstanciaAtividade”. Nesse diagrama é possível ver que há uma ação de entrada no estado ”finished”. Trata-se do serviço ”ActionExecuteInstanciaAtividadeAppService”, citado na Figura 28, responsável por se comunicar com uma ferramenta BPMS externa e tramitar os processos de negócio associados. O diagrama de estado do ”InstanciaAtividade” é representado por um arquivo XML na aplicação Labora, onde é possível indicar quais serviços serão executados na entrada/saída de cada estado. A execução desse serviço é feita de forma transparente e dentro de uma transação aberta pelo Iguassu, que garante que os processos de negócios não serão alterados em caso de erro na persistência dos dados da entidade. Dessa forma, o Iguassu integra a aplicação Labora com uma ferramenta BPMS externa e garante um baixo acoplamento entre elas.

O segundo serviço criado, ”ApixDownloadAppService”, é mais simples e tem como objetivo enviar informações de configurações e de atividades a serem realizadas (cadastradas na agenda) ao equipamento. Ou seja, o serviço busca informações no domínio do Labora e retorna os dados referentes ao XML *Schema*

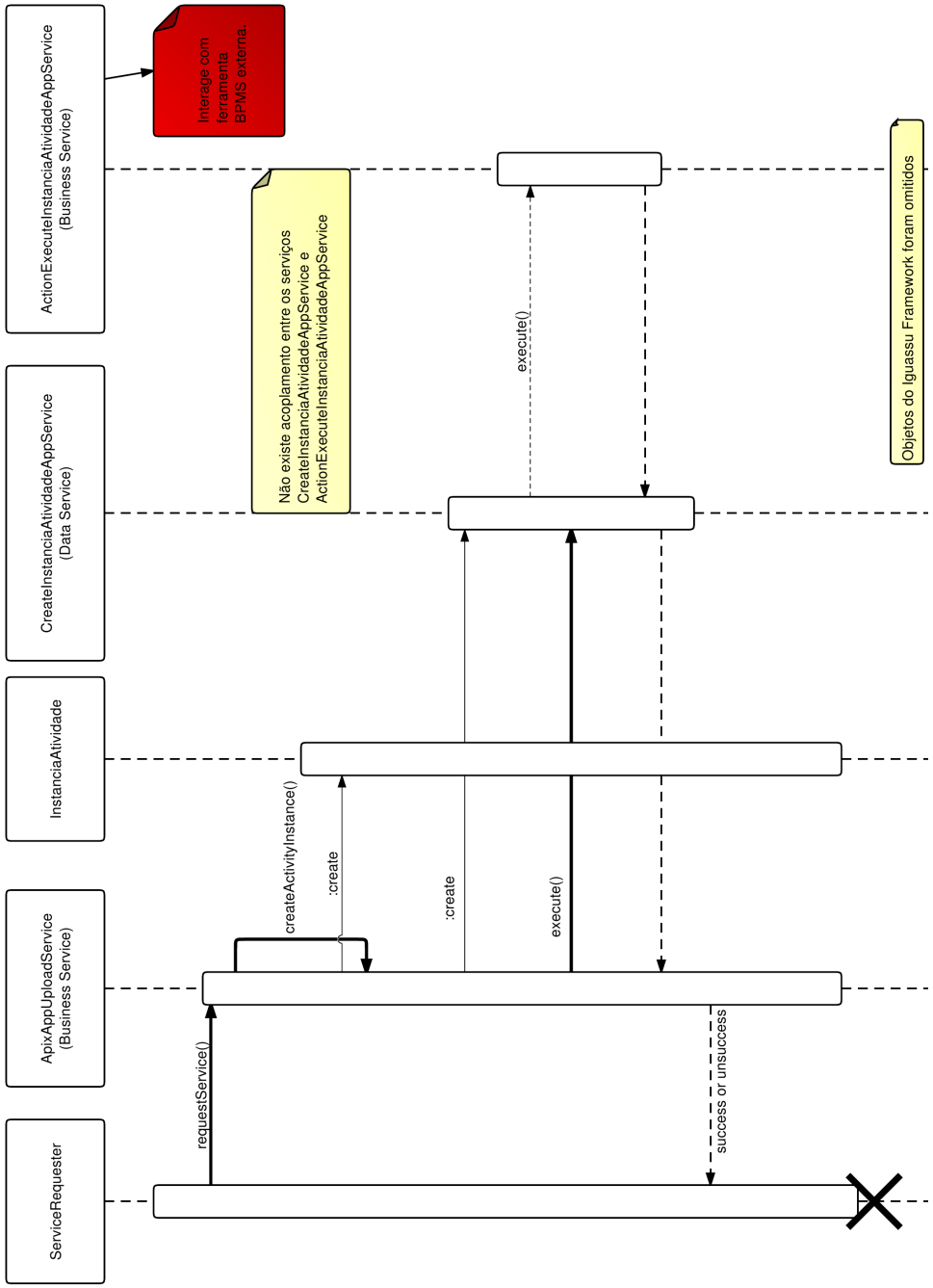

Figura 28: Diagrama de sequência, sem detalhes, da execução do serviço ApixAppUploadService

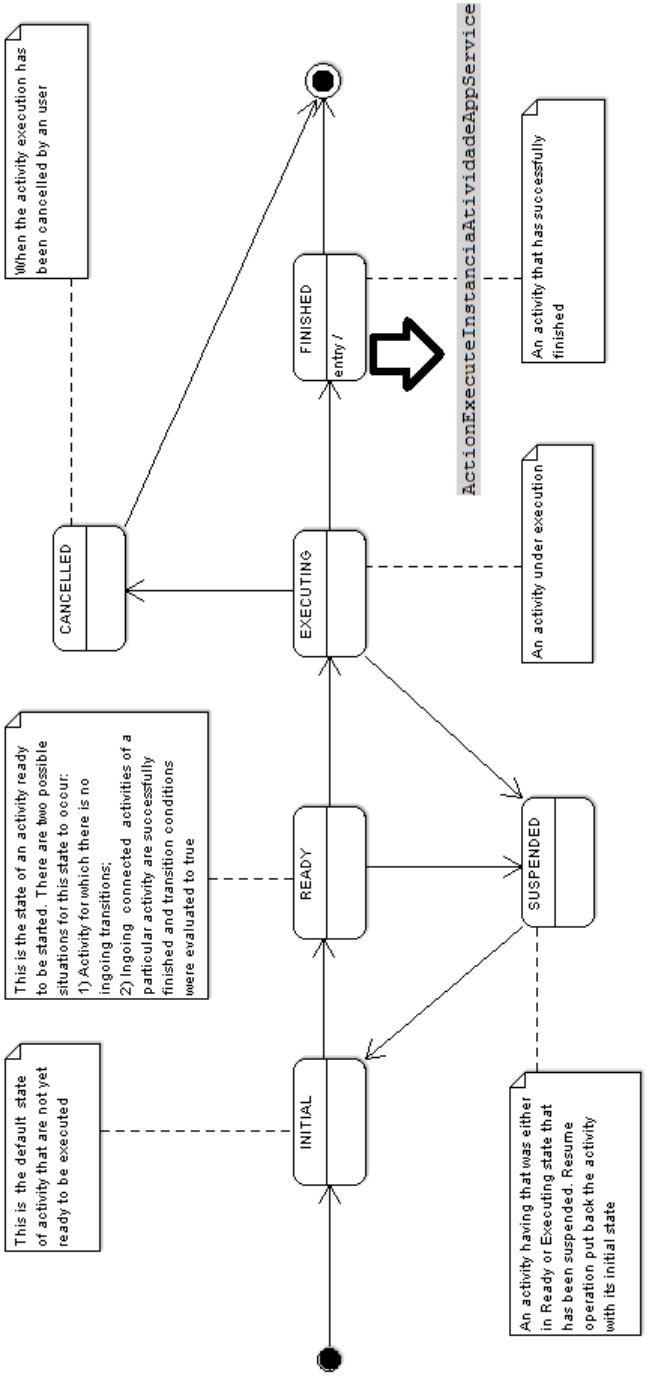

Figura 29: Diagrama de estados da entidade InstanciaAtividade do Labora

”apixgeneralconfig” e ao ”apixtasksconfig” do protocolo que foi descrito anteriormente.

3.3.4 A aplicação desktop

Diante da impossibilidade de comunicação direta entre o sistema web e um dispositivo de *hardware*, uma aplicação *desktop* foi criada para realizar a comunicação com o dispositivo através da porta serial do computador. A tecnologia *Java Web Start* foi utilizada para invocar sob demanda a aplicação *desktop*, que recebe ou envia os dados para o Apix e para o Labora.

Entretanto, uma aplicação desenvolvida em C/C++ que se comunicava com o equipamento já existia. Diante disso, nessa etapa do projeto, optou-se por utilizar a tecnologia JNI para invocar métodos nativos na aplicação legada que já funcionava.

Visando a alta manutenibilidade e extensibilidade da solução, optou-se por criar um *driver* que é responsável pela tradução dos dados vindos da aplicação C/C++ em objetos do JAXB (do protocolo criado anteriormente) e vice-versa. Para isso, foi necessário entender partes do funcionamento do *hardware* e o protocolo de comunicação entre ele e a aplicação C/C++. Uma das vantagens dessa abordagem é ocultar detalhes do hardware da aplicação *desktop* Java e, consequentemente, do Labora, garantindo que em caso de mudanças no equipamento e na aplicação C/C++, nada mudaria na aplicação *desktop*. Basta alterar a implementação do *driver* que ela utiliza.

Para que a aplicação *desktop* criada possa ser acionada pelo usuário através do navegador, foram criados dois arquivos JNLP. Um chamado ”apixDownload.jnlp” e o outro ”apixUpload.jnlp”. Ambos contém instruções sobre como o *Java Web Start* fará o *download* e utilizará a aplicação *desktop* (armazenada

no servidor). O que diferencia os arquivos é o argumento passado para a aplicação desktop no momento de sua execução. O primeiro passa a string ”laborapix.common.apix_ApixAppDownloadService” e o segundo a string ”laborapix. common. apix_ ApixAppUploadService”, que informam quais são os nomes dos serviços que serão invocados no Labora. As funções desses serviços foram explicadas anteriormente. Uma vez que a aplicação desktop foi iniciada, o usuário confirma a operação e a transferência dos dados é realizada.

Portanto, a aplicação desktop criada neste projeto é composta por uma camada que se comunica com o Labora através da invocação de serviços e por outra camada que implementa o protocolo criado por este projeto. Utilizando JNI, a aplicação desktop se comunica com a aplicação C/C++ legada para enviar e receber dados do equipamento. A utilização da JNI demandou a criação de código C/C++ para implementar os métodos nativos declarados no driver. Essa implementação consistiu apenas da invocação de funções na aplicação C/C++ previamente construída. A Figura 30 mostra essas camadas e a comunicação entre elas para facilitar o entendimento da estratégia adotada.

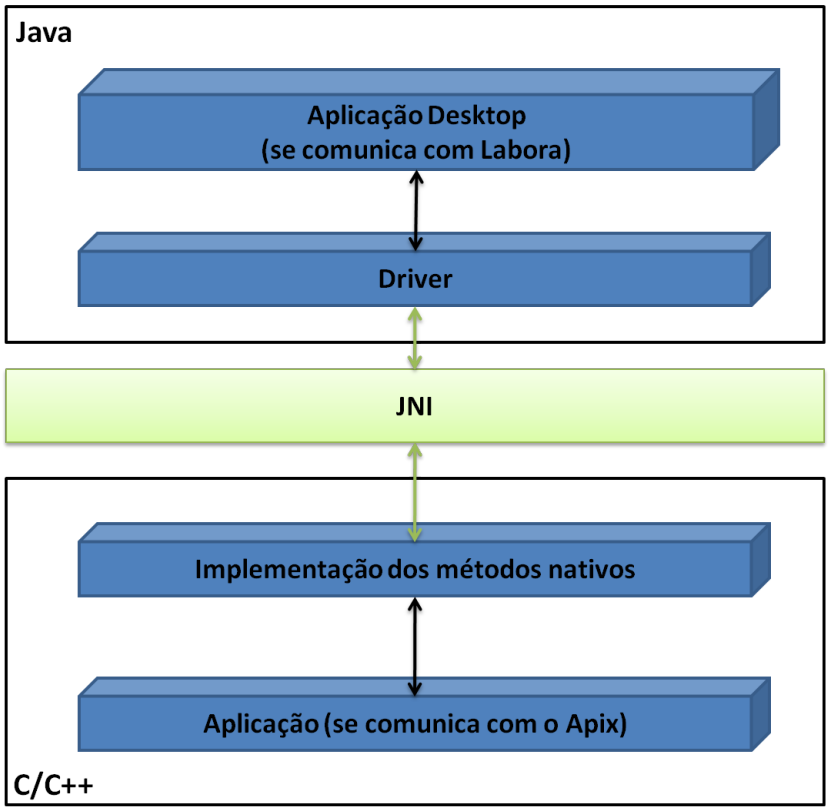

Figura 30: Utilização da JNI na comunicação entre a aplicação desktop desenvolvida em Java e a aplicação já existente

4 Discussão

Neste capítulo, todos os artefatos criados com o desenvolvimento do projeto e a forma como foram utilizados são descritos de forma detalhada, propiciando ao leitor a percepção completa dos resultados obtidos. Além disso, os resultados são discutidos com base no referencial teórico do projeto.

4.1 A Integração

Utilizando-se a metodologia apresentada anteriormente, verificou-se que os objetivos esperados foram alcançados e, como resultado, a solução de integração apresentada na Figura 31 foi implementada com sucesso.

A Figura 32 e a Figura 33 mostram o fluxo da comunicação quando deseja-se enviar dados do Labora para o Apix e quando deseja-se enviar dados do Apix para o Labora, respectivamente.

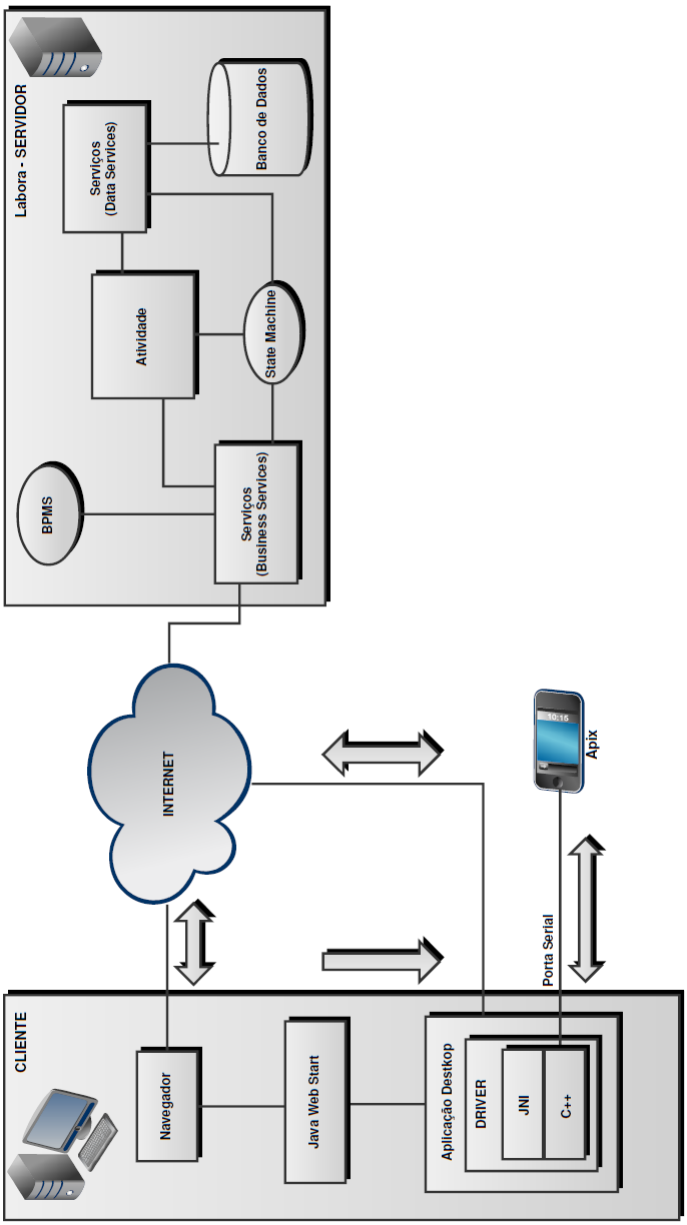

Figura 31: Demonstra como o dispositivo coletor de dados móvel foi integrado ao *software* orientado a processos de negócio

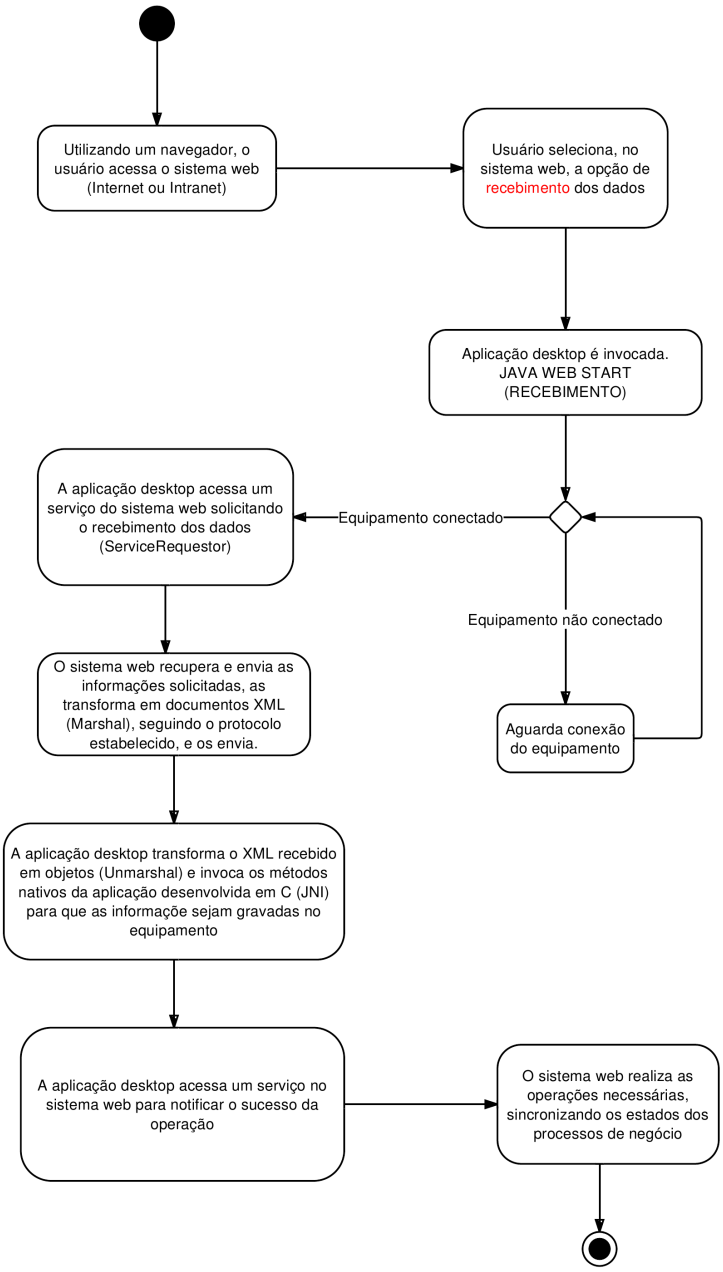

Figura 32: Fluxo da comunicação quando deseja-se enviar dados do Labora para o Apix

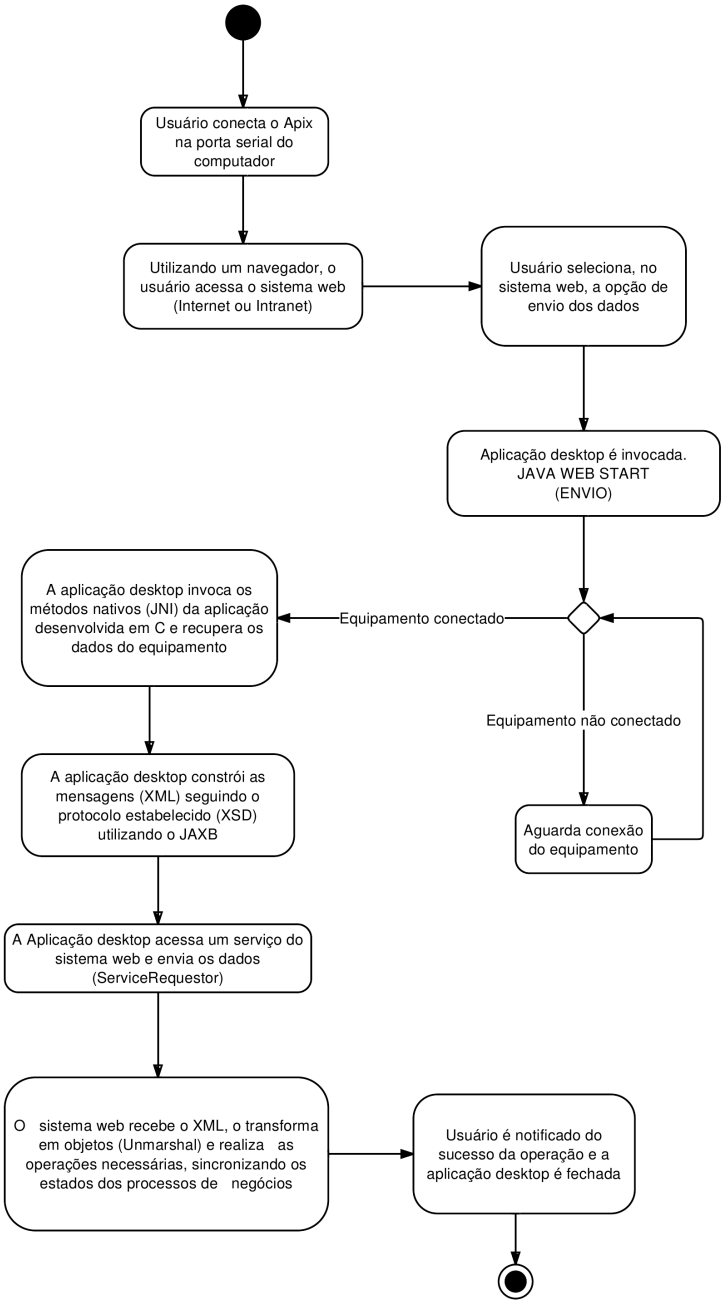

Figura 33: Fluxo da comunicação quando deseja-se enviar dados do Apix para o Labora

A integração foi implementada levando-se em consideração que o Labora não deve depender de uma versão específica do Apix. Dessa forma, diversos fornecedores de *hardware* podem fornecer versões distintas do equipamento, bastando que a implementação da tradução entre os dados do *hardware* e o protocolo criado por este trabalho seja feita. Essa implementação fica no *driver*, um arquivo ".jar" utilizado na aplicação *desktop* ou no Labora, dependendo da abordagem de integração escolhida. Dessa forma, apenas o *driver* é alterado quando quando uma das partes sofre alterações, o que garante um baixo acoplamento entre elas.

Relembrando, duas abordagens de comunicação entre Apix e Labora foram citadas por este trabalho. Uma delas consiste em utilizar o protocolo na aplicação *desktop* e no Labora, de tal forma que os dados trafegados entre elas fiquem no formato XML e sejam gerados pela API JAXB utilizando os XML *Schemas* do protocolo. Tal abordagem tem a vantagem de manter o Labora completamente desacoplado do equipamento, de tal forma que não é necessário alterar nada no servidor em caso de troca ou evolução do equipamento. Essa abordagem facilita o trabalho do fornecedor de *hardware*, que pode escolher em enviar os dados do equipamento diretamente no formato XML, seguindo o protocolo (o que acaba com a necessidade do *driver*), ou um ".jar" que implementa o *driver* e que é utilizado na aplicação desktop. É uma abordagem mais segura, já que implementações de terceiros não ficam no servidor, juntamente com o Labora, e sim na aplicação *desktop*, que só possui acesso aos dados do domínio do Labora através da invocação de serviços. Uma desvantagem dessa abordagem é o *overhead* de informação e de processamento na comunicação entre as partes. A Figura 26 mostra que os dados trafegados são encapsulados em documentos XML, que são transferidos pela rede. Isso aumenta significativamente o tráfego entre a aplicação *desktop* e o Labora. Além disso, esses documentos XML passam por um processo de tradução

em objetos (e vice-versa) em ambos os lados, o que demanda mais memória e processamento das partes.

Na segunda abordagem explicada por este trabalho, a aplicação *desktop* apenas repassa os pacotes recebidos do Apix para o Labora, onde ocorre a tradução dos dados para o protocolo utilizando-se a implementação do *driver* disponibilizada pelo fornecedor do *hardware*. Trata-se de uma melhor abordagem no que diz respeito à performance e é aconselhada quando muitos dados são trafegados entre as partes integradas. Porém, ela exige uma análise da implementação enviada pelo fornecedor de *hardware* a procura, principalmente, de riscos à segurança da informação, já que o código fica implantado e é executado no servidor, juntamente com o Labora.

Segundo (BARR; MASSA, 2006), um denominador comum em quase todo desenvolvimento de sistemas para *hardwares* e *softwares* que se comunicam com *hardwares* é a utilização das linguagens de programação C e C++. Isso implica que desenvolvedores de *hardwares* estão mais acostumados com aplicações *desktops* desenvolvidas em C/C++ para se comunicar com o *hardware*. Entretanto, existem APIs que facilitam a utilização da linguagem Java para se comunicar com a porta serial e, dessa forma, mais facilmente com o *hardware*. Porém, ambas são restritas a aplicações *desktops*, que devem ser instaladas no computador do usuário e são arquiteturalmente diferentes dos sistemas web. Confrontando as limitações expostas, este trabalho demonstrou como utilizar tecnologias existentes de tal forma a permitir que, através de um sistema web e de forma fácil e transparente, o usuário consiga se comunicar com o *hardware* conectado ao seu computador. O caso tratado pelo trabalho possui a peculiaridade de que o sistema que se comunicava com o *hardware* foi escrito em C/C++ e o sistema web escrito em Java, o que é muito comum nos dias atuais.

5 Conclusão

Este trabalho realizou com sucesso a integração entre o dispositivo coletor de dados Apix e o *software* orientado a processos de negócio Labora. Ela foi feita apoiando-se em boas práticas de desenvolvimento e padrões conhecidos, garantindo alta manutenibilidade e extensibilidade à solução desenvolvida. Tais características são importantes, visto que tanto o Apix quanto o Labora estão sofrendo alterações constantemente. Abstraindo o caso específico tratado por este trabalho, ele apresentou uma maneira fácil de utilizar um equipamento de *hardware* através de sistemas web acessados por qualquer computador com o Java e um navegador instalado.

Como forma de contribuição para a área de Ciência da Computação, o presente trabalho acrescentou uma nova fonte de consulta para pesquisadores e profissionais que irão realizar ou estudar uma integração de natureza semelhante a abordada por este trabalho. O trabalho foi realizado dentro de um contexto local e, dessa forma, ajudou a empresa Mitah Technologies (detentora do dispositivo coletor de dados móvel e do *software* orientado a processos de negócio) com a implementação da solução de integração proposta.

Referências

BARR, M.; MASSA, A. *Programming Embedded Systems*. [S.l.]: O'Reilly, 2006. 304 p.

COHEN, F. A cryptographic checksum for integrity protection. *Computers & Security*, v. 6, p. 505–510, 1987.

ERL, T. *Service-Oriented Architecture: Concepts, Technology, and Design*. [S.l.]: Prentice Hall PTR, 2005.

GOSLING, J.; JOY, B.; STEELE, G.; BRACHA, G. *The Java language specification, third edition.* [S.l.]: Addison-Wesley, 2005.

GROUP, S. O. *Whitepaper: Service-Oriented Architecture (SOA)*. 2006. Consultado em 02/05/2010. Disponível em: <http://www.opengroup.org-/projects/soa/doc.tpl? CALLER=documents.tpl\dcat=\gdid=1873>.

GROVES, D. *Successfully planning for SOA*. [S.l.]: BEA Systems Worldwide, 2005.

JOHNSTON, S.; KELLY, K.; BROWN, A. *Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications*. [S.l.]: Ration Software Corporation, 2002.

JUNG, C. F. *Metodologia para Pesquisa e Desenvolvimento: aplicada a novas tecnologias, produtos e processos*. [S.l.]: Axcel Books do Brasil, 2004.

JÚNIOR, J. J. L. D. *A Software Architecture Process for SOA-Based Enterprise Applications*. Disserta (Mestrado) — Universidade Federal de Pernambuco, 2008.

LIANG, S. *The Java Native Interface. Programmers Guide and Specification*. [S.l.]: Addison-Wesley, 1999.

MAHMOUD, Q. H. Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI). 2005.

MARCONI, M. A. *Fundamentos de Metodologia Científica*. [S.l.: s.n.], 2003.

MCGOVERN, S.; TYAGI, J.; STEVENS, S.; MATTHEW, M. *Java Web Services Architecture*. [S.l.]: Morgan Kaufmann, 2003.

MCLAUGHLIN, B. *Java and XML Data Binding*. [S.l.]: O'Reilly Media, 2002.

MCLAUGHLIN, B.; EDELSON, J. *Java and XML*. [S.l.]: O'Reilly Media, 2006.

ROSHEN, W. *SOA-Based Enterprise Integration: A Step-by-Step Guide to Services-Based Application Integration*. [S.l.]: McGraw-Hill Companies, 2009.

SAÚDE, A. V.; VARGAS, G. V.; VICTORIO, R. A.; JR, J. C. R. A data collector specialized on beekeepers activity registering. *TRACE 4th Annual Meeting and Conference*, p. 58–58, 2008.

SAÚDE, A. V.; VARGAS, G. V.; VICTORIO, R. A.; JR, J. C. R. A generic protocol for fully automatic activity registering in traceability. *TRACE 4th Annual Meeting and Conference*, p. 59–59, 2008.

SAÚDE, A. V.; VICTORIO, R. A. da S. S.; COUTINHO, G. C. A.; MARCON, M.; PAIVA, J. L. de; DAMASCENO, R. J. Service oriented framework for lightweight enterprise applications. *INFOCOMP Journal of Computer Science*, Special, n. 2, p. 49–58, 2010.

SCHOOLS, W. *XML Schema Tutorial*. 2010. Acessado em 20/05/2010. Disponível em: <http://www.w3schools.com/schema/default.asp>.

SUN. *Java Web Start 1.4.2 Developer Guide*. 2004. Acessado em 03/05/2010. Disponível em: <http://java.sun.com/j2se/1.4.2/docs% -/guide/jws-/developersguide/overview.htm>.

VERNADAT, F. B. Enterprise integration and interoperability. In: ____. [S.l.]: Springer-Verlag, 2009. p. 1529–1538.

VICTORIO, R. A.; SAÚDE, A. V.; BOMFIM, L. J.; PATACA, C.; RAMOS, A.; JR, C. J. R. Good practices, haccp, traceability and production management for apiculture. *TRACE 4th Annual Meeting and Conference*, p. 57–57, 2008.

VOHRA, A.; VOHRA, D. *Pro XML Development with Java Technology*. [S.l.]: Apress, Inc, 2006.

WESKE, M. *Business Process Management: Concepts, Languages, Architectures*. [S.l.]: Springer Berlin Heidelberg, 2007.

ZHANG, L. J. Business process management and integration. In: ____. [S.l.]: Springer Berlin Heidelberg, 2008. p. 224–242.