

UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIAS DA COMPUTAÇÃO
(Bacharelado)

**PROTÓTIPO DE PADRÃO DE COMUNICAÇÃO E
ARMAZENAMENTO DE DADOS CONTÁBEIS USANDO
XML**

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE
REGIONAL DE BLUMENAU PARA OBTENÇÃO DOS CRÉDITOS NA
DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA
COMPUTAÇÃO – BACHARELADO

JOÃO KRACIK

BLUMENAU, NOVEMBRO/2002

2002/2-38

PROTÓTIPO DE PADRÃO DE COMUNICAÇÃO E ARMAZENAMENTO DE DADOS CONTÁBEIS USANDO XML

JOÃO KRACIK

ESTE TRABALHO DE CONCLUSÃO DE CURSO FOI JULGADO ADEQUADO
PRA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE
CONCLUSÃO DE CURSO OBRIGATÓRIA PARA A OBTENÇÃO DO TÍTULO
DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

Prof. Ricardo Guilherme Radünz – Orientador na FURB

Prof. José Roque Voltolini da Silva – Coordenador do TCC

BANCA EXAMINADORA

Prof. Ricardo Guilherme Radünz

Prof. Maurício Capobianco Lopes

Dr. Oscar Dalfovo

AGRADECIMENTOS

Agradeço a todos aqueles que de uma forma ou de outra contribuíram e incetivaram e conclusão desse trabalho. Um agradecimento especial para meu orientador Ricardo Guilherme Radünz, Marcos Machado Soares Cabral, grande amigo e colaborador e Fabrício Bento, que compartilham comigo a paixão pela informática. Agradeço também a todo o pessoal de Guru Sistemas, pela paciência e a disposição que tiveram comigo.

SUMÁRIO

LISTA DE FIGURAS.....	vi
LISTA DE QUADROS.....	vii
RESUMO.....	ix
ABSTRACT.....	x
1 INTRODUÇÃO.....	1
1.1 OBJETIVO DO TRABALHO.....	2
1.2 ESTRUTURA DO TRABALHO.....	2
2 FUNDAMENTAÇÃO TEÓRICA.....	4
2.1 EXTENSIVE MARKUP LANGUAGE (XML).....	4
2.1.1 ORIGEM DA XML.....	4
2.1.2 USO DA XML.....	5
2.1.3 A ESTRUTURA DA XML.....	6
2.1.3.1 ATRIBUTOS.....	8
2.1.4 XML SCHEMA.....	9
2.1.5 WEB SERVICES E SOAP.....	15
2.2 A CONTABILIDADE.....	17
2.2.1 O PLANO DE CONTAS.....	18
2.2.2 OS LANÇAMENTOS CONTÁBEIS.....	20
2.2.3 ENTIDADES CONTÁBEIS.....	21
3 DESENVOLVIMENTO DO TRABALHO.....	23
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO.....	23
3.2 ESPECIFICAÇÃO.....	24

3.2.1	O PADRÃO E A CONTABILIDADE.....	25
3.2.2	DIAGRAMA DE CASOS DE USO.....	25
3.2.3	DIAGRAMA DE CLASSES.....	29
3.2.4	O PADRÃO EM XML SCHEMA.....	31
3.2.4.1	A DEFINIÇÃO DOS ELEMENTOS.....	31
3.2.4.2	AS MENSAGENS.....	36
3.3	IMPLEMENTAÇÃO.....	40
3.3.1	TÉCNICAS E FERRAMENTAS UTILIZADAS.....	40
3.3.1.1	OS OBJETOS NO SERVIDOR.....	40
3.3.1.2	OS OBJETOS NOS CLIENTES.....	43
3.3.2	OPERACIONALIDADE DA IMPLEMENTAÇÃO.....	44
3.4	RESULTADOS E DISCUSSÃO.....	53
4	CONCLUSÕES.....	55
4.1	EXTENSÕES.....	56
	REFERÊNCIAS BIBLIOGRÁFICAS.....	57
	ANEXOS.....	58

LISTA DE FIGURAS

Figura 1 – Tipos de dados no XML Schema.....	14
Figura 2 – Entidades/relacionamentos da contabilidade.....	22
Figura 3 – Diagrama de casos de uso.....	26
Figura 4 – Diagrama de classes.....	29
Figura 5 – Tela principal do aplicativo cliente.....	44
Figura 6 – Tela de manutenção e seleção de contas.....	47
Figura 7 – Tela de cadastro de conta.....	48
Figura 8 – Cadastro de lançamentos. Partidas simples.....	49
Figura 9 – Cadastro de lançamentos. Partidas compostas.....	50

LISTA DE QUADROS

Quadro 1 – Cabeçalho de documento XML.....	6
Quadro 2 – Cabeçalho de documento XML, detalhado.....	7
Quadro 3 – Exemplo de documento XML.....	7
Quadro 4 – Exemplo de documento XML.....	9
Quadro 5 – Exemplo de XML Schema.....	10
Quadro 6 – Exemplo de mensagem de requisição.....	16
Quadro 7 – Exemplo de mensagem de resposta.....	17
Quadro 8 – Lançamento com partida simples.....	20
Quadro 9 – Lançamento com duas partidas de crédito.....	21
Quadro 10 – Lançamento com duas partidas de débito.....	21
Quadro 11 – Lançamento com dias partidas de débito e duas de crédito.....	21
Quadro 12 – Estrutura de TConta.....	31
Quadro 13 – Estrutura de TContaArray.....	33
Quadro 14 – Estrutura de TPartida.....	33
Quadro 15 – Estrutura de TPartidaArray.....	34
Quadro 16 – Estrutura de TLancamento.....	35
Quadro 17 – Estrutura de TLancamentoArray.....	35
Quadro 18 – Estrutura de GetContaRequest e GetContaResponse.....	36
Quadro 19 – Estrutura de GetChildsRequest e GetChildsResponse.....	37
Quadro 20 – Estrutura de NewContaRequest e NewContaResponse.....	37
Quadro 21 – Estrutura de GetPartidaRequest e GetPartidaResponse.....	38
Quadro 22 – Estrutura de GetLancamentoRequest e GetLancamentoResponse.....	38
Quadro 23 – Estrutura de GetLancamentosRequest e GetLancamentosResponse.....	39

Quadro 24 – Estrutura de NewLancamentoRequest.....	39
Quadro 25 – Classe TConta.....	41
Quadro 26 – Classe TPartida.....	41
Quadro 27 – Classe TLancamento.....	42
Quadro 28 – Definições de TContaArray, TPartidaArray e TLancamentoArray.....	42
Quadro 29 – Definição de interface de Icontabilidade.....	43
Quadro 30 – Código fonte da obtenção de lançamentos.....	45
Quadro 31 – Código fonte da obtenção de contas filhas.....	47
Quadro 32 – Código fonte do cadastro de um novo lançamento.....	50

RESUMO

O presente trabalho trata da elaboração de um padrão de comunicação de dados contábeis que permite a comunicação entre aplicativos independente de sua arquitetura ou plataforma. O padrão é desenvolvido para funcionar em rede usando o conceito de multi-camadas. Por isso utiliza tecnologias como XML Schema, SOAP e Web Services que são baseadas na linguagem XML.

ABSTRACT

The present work deals with a accounting data communication standard that admit the communication between applications independent of architecture or platform. The standard is developed to be used in network using the concept of multi-tier. Hence use technology like XML Schema, SOAP and Web Services that be based in XML language.

1 INTRODUÇÃO

Já há algum tempo a informática tornou-se indispensável em quase todos os ramos da indústria, do comércio e na vida particular de cada pessoa. A comunicação entre pessoas e empresas tem transformado-se todos os dias e as tecnologias de informação têm adaptado-se a essa nova realidade que é a Internet. E neste universo de tecnologias que envolvem a Internet, uma coisa parece ser fundamental: elas devem falar a mesma língua.

Segundo Furgeri (2001), a Internet possibilitou o surgimento de uma atividade muito importante para o mundo de computadores – a troca de dados entre máquinas espalhadas pelo globo terrestre. Essa característica é muito importante pelo fato de os computadores não necessitarem ser do mesmo tipo ou mesmo fabricante. Ela envolve qualquer tipo de computador, seja um supercomputador, ou um microcomputador doméstico.

Não é de hoje que aplicativos compartilham informações entre si. Existem exemplos de comunicação de dados que são largamente usados hoje em dia e que não são em espécie alguma na forma on-line. Um bom exemplo são os dados que devem ser transmitidos por um aplicativo comercial para a Receita Federal, e que são enviados via disquete após a exportação desses dados para um arquivo no formato fornecido pela Receita Federal. Outro exemplo, e nesse muito mais visível a subtração de esforço, são em casos em que uma empresa emite uma nota fiscal, em seu sistema comercial, que envia ao comprador que tem de digitá-la também em seu sistema comercial, sendo que muitas vezes são os mesmos sistemas. Depois disso o comprador ainda tem de enviar essas informações para seu contador.

Uma solução para esse problema seria a padronização dos dados e da forma de armazenamento e a construção de ferramentas que trabalhem dentro deste padrão. Poupar-se assim, trabalho de estudo, definição e implementação.

Em função desses acontecimentos, surgiu a idéia de pesquisar e desenvolver uma linguagem que abrangesse as necessidades básicas de um sistema contábil. Um

padrão de comunicação que permitisse sistemas distintos comunicarem-se e ainda fazer ampliações para que o padrão possa adaptar-se a cada necessidade específica sem perder suas características básicas.

Além disso, o padrão ou a estrutura dos dados deveria ser direcionada à funcionar através da internet e em múltiplas camadas o que fez com que escolhesse a *Extensive Markup Language* (XML) como tecnologia de base para o desenvolvimento do padrão.

1.1 OBJETIVO DO TRABALHO

O objetivo principal que caracteriza este trabalho é a definição de uma forma padrão de comunicação de dados contábeis usando a XML e suas extensões como base dessa estrutura.

Os objetivos secundários deste trabalho são:

- a) desenvolver um servidor, que deverá armazenar e comunicar os dados contábeis, usando o padrão estabelecido;
- b) desenvolver um aplicativo cliente, que deverá comunicar-se com o servidor, fornecendo e extraindo-lhe informações;

1.2 ESTRUTURA DO TRABALHO

O primeiro capítulo apresenta o assunto do trabalho seus objetivos e a estrutura na qual o trabalho será desenvolvido.

O segundo capítulo apresenta as tecnologias que foram utilizadas e os assuntos mais relevantes do trabalho como:

- a) XML;
- b) XML Schema;
- c) *Web Services* e SOAP;

d) a contabilidade;

O terceiro capítulo apresenta a especificação do trabalho, mostrando os diagramas que foram gerados e o detalhamento de sua implementação.

O quarto capítulo apresenta as conclusões e as sugestões para trabalhos futuros.

Os quadros contendo textos escritos em XML são formados em cor azul e os textos que representam código fonte escritos em Delphi 6 são formados em cor preta.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 EXTENSIVE MARKUP LANGUAGE (XML)

A seguir serão descritas as principais características da linguagem XML que foi utilizada para criar a estrutura e também faz a transmissão dos dados entre os aplicativos desenvolvidos neste trabalho.

2.1.1 ORIGEM DA XML

Segundo Furgeri (2001), em 1996, especialistas em SGML (*Standard Generalized Markup Language*), a principal linguagem de marcação da qual surgiu a HTML, sob a chefia de Jon Bosak, da Sun Microsystems, se uniram para definição de um novo padrão de marcação que pudesse ser utilizado na Internet, constituindo-se em uma versão simplificada da SGML, cujo objetivo principal era fornecer aos desenvolvedores da Web maneiras de definir e criar seus próprios marcadores e atributos quando necessário, em vez de estarem restritos ao esquema de marcação da HTML. No final de 1996, o comitê de trabalho anunciou a primeira versão preliminar da XML em uma conferência da SGML, realizada em Boston, nos Estados Unidos. Novos recursos foram consolidados no primeiro semestre de 1997.

A meta principal do comitê foi desenvolver uma linguagem de marcação que tivesse a capacidade e a generalidade da SGML, e fosse fácil de ser implementada na Web. Resumidamente, as características desejadas inicialmente para a XML se referiam a três partes: a definição da linguagem em si (XML-LANG), a definição da ligação entre os documentos (XML-LINK) e a forma de apresentação dos documentos (XS).

As regras básicas para criação dessa linguagem de marcação, isto é, as principais características desejáveis para a implementação na Web eram as seguintes:

- a) criar uma linguagem simples, que possibilitasse a rápida construção de documentos para implementação na Web;
- b) fornecer suporte à criação de aplicações compatíveis com a abordagem HTML;

- c) possibilitar o desenvolvimento de uma grande variedade de aplicativos, aproveitando-se de seus recursos;
- d) fornecer um mecanismo de apresentação genérico e poderoso, permitindo ao desenvolvedor criar a forma de apresentação que mais se adapte às suas necessidades;
- e) fornecer suporte para a criação de marcadores personalizados, definidos pelo desenvolvedor do documento Web;
- f) permitir a criação de documentos que pudessem ser validados, isto é que existisse uma forma de verificar a estrutura do documento, verificando se seus elementos eram válidos, da mesma forma que ocorria com a SGML;
- g) fornecer suporte para a criação de hiperlinks que fossem compatíveis com a especificação de endereços URL (Uniform Resource Locator), de modo a criar ligações entre documentos;
- h) fornecer um mecanismo de folha de estilo genérico e poderoso, que possibilitasse não apenas a formatação do documento, como também sua manipulação.

Uma vez contempladas essas características, a XML passa a fornecer um meio completo para a elaboração e distribuição de documentos por toda a Web, sendo independente de plataformas e de sistemas. O objetivo era transformar o conceito da HTML, fornecendo à XML recursos adicionais para a distribuição de documentos.

2.1.2 O USO DA XML

Entender a utilidade da XML, além do uso como a HTML em um browser, é uma questão trabalhosa. É surpreendente como a XML pode ser muito útil em qualquer sistema que transfira ou armazene dados. De modo prático, a XML não passa de um arquivo tipo texto que armazena os dados em forma de árvore e que faz isso usando marcadores entre cada informação. Isso faz da XML uma forma de comunicação extremamente aberta e de fácil leitura.

A XML não é uma grande novidade ou grande descoberta. A XML é o resultado do avanço tecnológico, que fornece recursos ou retiram limitações e a necessidade de padronização da comunicação de informações.

Segundo Furgeri (2001), a XML é a evolução da linguagem HTML. Ela contém características especiais que permitem descrever o documento de forma inteligente, tornando o significado de seu conteúdo mais compreensível tanto para os seres humanos como para os computadores. Enquanto a HTML indica como algo deve ser exibido, a XML indica o que a informação significa.

Isso faz com que a XML vá além da capacidade da HTML de mostrar páginas na Web. Dois servidores podem se comunicar entre si através de um formato aberto, de uma tecnologia amplamente conhecida, facilitando e aumentando a velocidade de integração entre sistemas distintos.

2.1.3 A ESTRUTURA DA XML

A XML possui várias estruturas que servem para armazenar dados e que tornam distinto cada tipo de informação e como ela deve ser usada. As informações são identificadas em um documento XML, independentes do uso que se fará delas, através da intercalação com símbolos de marcação. Os símbolos de menor (<) e maior (>) são usados para identificar essas marcações, que são chamadas de *tags* (do inglês, caracteres, expressão, pôr etiquetas, ligar ou unir) e o texto entre as *tags* é o conteúdo do documento. E é dessa forma que começa a nascer a XML.

O topo de um documento XML sempre contém informações especiais chamadas de prólogo do documento. O prólogo, em sua versão mais simples, é usado para identificar que tipo de documento está sendo tratado e a versão usada para formatar o documento, (Ray, 2001). No quadro 1 é apresentado um exemplo de um prólogo em sua forma mais comum.

Quadro 1 – Cabeçalho de documento XML

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


Outras informações podem fazer parte do prólogo de um documento, acrescentando links para outros documentos de definição de tipos e formatos do documento. No quadro 2 tem-se um exemplo, encontrado em Ray (2001), que mostra um prólogo mais completo:

Quadro 2 – Cabeçalho de documento XML, detalhado.

```
<?xml version="1.0"?>
<!DOCTYPE time-o-gram
    PUBLIC "-//LordsOfTime//DTD TimeOGRAM 1.8//EN"
    "http://www.lordsoftime.ogr/DTDs/timeogram.dtd"
[
    <!ENTITY sj "Sarah Jane">
    <!ENTITY me "Doctor Who">
]>
```

Logo após o prólogo vem a entidade raiz, que contém o restante do documento.

Em um documento XML só há um único elemento raiz, todos os outros elementos do documento estão sempre contidos no elemento raiz.

No quadro 3, um exemplo de um documento XML que tem por raiz o elemento *contabilidade*:

Quadro 3 – Exemplo de documento XML

```
01 <?xml version="1.0"?>
02 <contabilidade>
03     <lancamento>
04         <data>20020101</data>
05         <historico>Vendas de mercadorias a prazo
06         </historico>
07         <conta id="1234" acao="D"/>
08         <conta id="4321" acao="C"/>
09         <valor>1550,00<valor>
10     </lancamento>
11 </contabilidade>
```

A numeração na frente do exemplo é somente para a identificação das linhas, ela não faz parte do documento.

Na linha 02 está a entidade raiz, que contém todo o documento. Nesse caso, o documento representa as informações de um único lançamento contábil que começa na

linha 03 e termina na linha 10 com a tag <lancamento>. Contidos na tag <lancamentos> estão outros elementos que representam: a data do lançamento na linha 04, o histórico do lançamento nas linhas 05 e 06, a conta de débito na linha 07, a conta de crédito na linha 08 e o valor do lançamento na linha 09.

Há muito que se falar sobre esse exemplo. Pode-se começar com o fato da XML distinguir textos em maiúsculas e minúsculas. Isso quer dizer que <Valor> e <valor> não são o mesmo elemento para o XML.

Nota-se que os elementos no XML podem conter um valor simples, como no caso do elemento <data> ou podem conter outros elementos. Podem também não conter valor algum, como é o caso dos elementos <conta>.

As informações de cada elemento são sempre intercaladas por tags, como dito anteriormente, sendo que a tag que finaliza o elemento possui o símbolo “/” logo após o símbolo “<”, simbolizando que essa é a tag que termina o elemento e não o começo e mais um elemento com o mesmo nome, dentro desse elemento. Existe uma exceção à regra, que é quando um elemento não possui um valor ou tem o valor nulo, como é o caso dos elementos conta. Nesses casos a tag que inicia é também a tag que finaliza, sendo que o formato da tag contém o símbolo “/” indicando isso.

No caso dos elementos de conta, estes são elementos complexos e que não necessitam, nesse caso, levar todos os seus sub elementos, visto que provavelmente o destino dessa informação não necessite desses dados. Ao invés disso foram somente passados *atributos* (*id e acao*) que identificam as contas e que definem a ação que essas contas tomam dentro do lançamento em questão. A informação da ação tomada pela conta não é uma ação que faça parte das informações da conta, mas sim um *atributo* que essa conta possui, enquanto dentro desse lançamento.

2.1.3.1 ATRIBUTOS

Segundo Ray (2001), os atributos servem para transmitir mais informações sobre o elemento do que já expresso em seu nome e conteúdo. Os atributos são usados para

dar características únicas ao elemento que facilite a localização da mesma. Também são usados para definir características e o comportamento do elemento em questão. Nos exemplos de contas, encontra-se os atributos *id* e *acao*. O atributo *id* funciona como um identificador único do elemento para o sistema. Já o atributo *acao* serve para identificar o comportamento do elemento *conta* dentro do elemento *lançamento*.

Os atributos são compostos do nome do atributo, seguido pelo sinal de igual (=) e seguido do conteúdo do atributo entre apóstrofes (‘), como: `id="1234"`.

2.1.4 XML SCHEMA

Segundo a W3C (2002), “XML Schema explica o vocabulário distribuído e permite que máquinas expressem regras feitas por pessoas. Ela provê os meios para definir a estrutura, o conteúdo e a semântica de documentos XML”.

No quadro 4, o exemplo extraído da W3C(2002) mostra um documento XML que armazena os dados sobre uma ordem de compra efetuada em 20/10/1999. O documento que armazena efetivamente os dados, como o documento abaixo é definido como “instância de documento”.

Quadro 4 – Exemplo de documento XML

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
  <shipTo country="US">
    <name>Alice Smith</name>
    <street>123 Maple Street</street>
    <city>Mill Valley</city>
    <state>CA</state>
    <zip>90952</zip>
  </shipTo>
  <billTo country="US">
    <name>Robert Smith</name>
    <street>8 Oak Avenue</street>
    <city>Old Town</city>
    <state>PA</state>
    <zip>95819</zip>
  </billTo>
  <comment>Hurry, my lawn is going wild!</comment>
</items>
```

```

    <item partNum="872-AA">
      <productName>Lawnmower</productName>
      <quantity>1</quantity>
      <USPrice>148.95</USPrice>
      <comment>Confirm this is electric</comment>
    </item>
    <item partNum="926-AA">
      <productName>Baby Monitor</productName>
      <quantity>1</quantity>
      <USPrice>39.98</USPrice>
      <shipDate>1999-05-21</shipDate>
    </item>
  </items>
</purchaseOrder>

```

O documento XML mostrado no quadro 4 é constituído pelo elemento principal *purchaseOrder* e pelos elementos *shipto*, *billto*, *comment* e *items*. Elementos que contém subelementos, como *shipto* ou contém atributos são chamados “elementos complexos”. Por sua vez, elementos que possuem um valor, mas não possuem subelementos são chamados elementos simples.

Os elementos complexos e alguns subelementos que constam na instância do documento são declarados no esquema do documento, outros subelementos podem estar diretamente relacionados ao repertório de tipos padrões existente para o XML Schema.

O quadro 5 demonstra o documento XML que representa o XML Schema para a instância de documento *purchaseOrder*.

Quadro 5 – Exemplo de XML Schema

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      Purchase order schema for Example.com.
      Copyright 2000 Example.com. All rights reserved.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>
  <xsd:element name="comment" type="xsd:string"/>

```

```

<xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
    <xsd:element name="shipTo" type="USAddress"/>
    <xsd:element name="billTo" type="USAddress"/>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="items" type="Items"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>

<xsd:complexType name="USAddress">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="street" type="xsd:string"/>
    <xsd:element name="city" type="xsd:string"/>
    <xsd:element name="state" type="xsd:string"/>
    <xsd:element name="zip" type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:NMTOKEN"
    fixed="US"/>
</xsd:complexType>

<xsd:complexType name="Items">
  <xsd:sequence>
    <xsd:element name="item" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="productName" type="xsd:string"/>
          <xsd:element name="quantity">
            <xsd:simpleType>
              <xsd:restriction base="xsd:positiveInteger">
                <xsd:maxExclusive value="100"/>
              </xsd:restriction>
            </xsd:simpleType>
          </xsd:element>
          <xsd:element name="USPrice" type="xsd:decimal"/>
          <xsd:element ref="comment" minOccurs="0"/>
          <xsd:element name="shipDate" type="xsd:date"
minOccurs="0"/>
        </xsd:sequence>
        <xsd:attribute name="partNum" type="SKU" use="required"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<!-- Stock Keeping Unit, a code for identifying products -->
<xsd:simpleType name="SKU">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
</xsd:simpleType>

</xsd:schema>

```

O XML Schema do quadro 5 consiste do elemento *schema* e sub elementos que variam entre elementos complexos (*complextype*) e elementos simples (*simpletype*).

A declaração `<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">` serve para associar o prefixo *xsd:* que aparece em todos os elementos do XML Schema. O objetivo do prefixo *xsd:* é vincular a declaração do elemento como sendo pertencente ao vocabulário padrão do XML Schema. O prefixo *xsd:* foi convencionado para a identificação, embora qualquer outro prefixo possa ser usado desde que transmissor e receptor estejam cientes disso.

Os elementos complexos são definidos usando o termo *complextype* e o atributo *name* especifica os nomes das tags dos elementos que devem seguir essa definição. Um elemento complexo normalmente contém definições de elementos simples que são definidos pelo elemento *element* enquanto atributos são definidos usando-se o elemento *attribute*. Em ambos os casos aparecem os atributos *name* e *type* que definem o nome da regra que define o elemento e o tipo do elemento.

O uso dos atributos *minOccurs* e *maxOccurs* destinam-se a definir quantas vezes um elemento deve ou pode aparecer em um elemento complexo. Se o atributo *minOccurs* aparecer com o valor 0 significa que nenhuma ocorrência do elemento precisa existir, mas se estiver com o valor 1, 2 ou 3, significa que o elemento deve aparecer no mínimo uma ou duas ou três vezes respectivamente. Se o atributo *maxOccurs* aparecer, deverão ser respeitadas o número máximo de ocorrências para esse elemento.

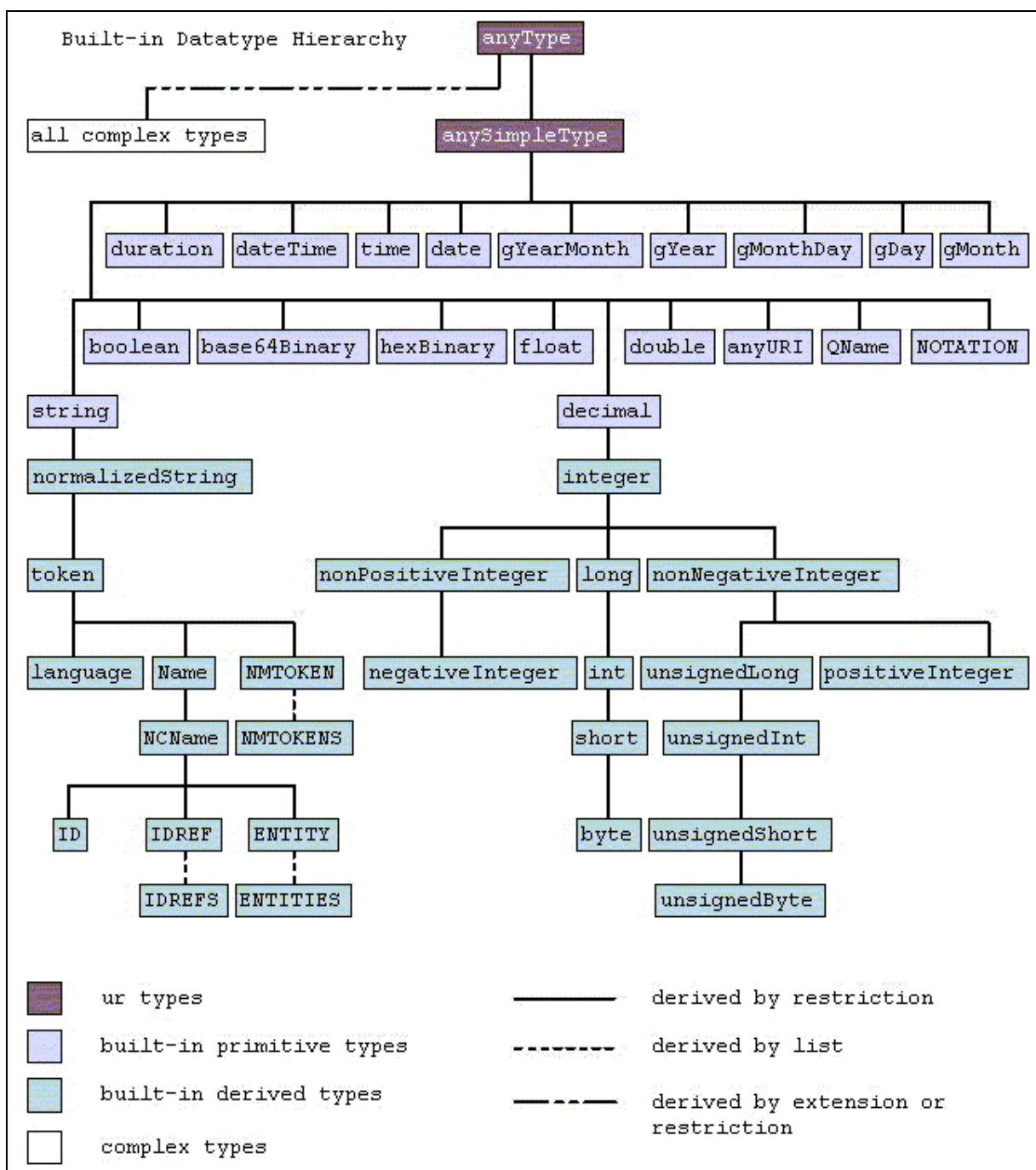
O atributo *use* é usado para definir atributos como sendo requeridos (*required*), opcionais (*optional*) ou proibido (*prohibited*).

O atributo *default* define o valor que deverá ser atribuído ao elemento, caso nenhum valor seja indicado. Valores default podem ser definidos tanto para atributos quanto para elementos. O processador XML irá atribuir o valor default a um atributo, quando este não existir e atribuirá o valor default de um elemento quando este elemento não tiver valor.

O atributo *fixed* define um valor fixo para o elemento ou atributo. Isso permite que tanto o elemento e o atributo sejam opcionais em sua ocorrência, embora nos casos de não ocorrência, o processador XML irá usar o valor declarado no atributo *fixed* para preencher o valor do elemento ou atributo faltante.

O XML Schema pré define alguns tipos de informações que podem ser usados para a definição de elementos e atributos. Há muito que se falar sobre as várias categorias de tipos que podem ser usadas, incluindo tipos em lista e tipos de união. Na figura 1 mostra-se um diagrama hierárquico dos tipos simples disponíveis no XML Schema. Para mais detalhes sobre tipos de dados no XML Schema veja (W3C, 2002).

Figura 1 – Tipos de dados no XML Schema



Fonte: W3C (2002)

2.1.5 WEBSERVICES E SOAP

Segundo Fisher (2002), “*Web services*, na idéia geral do termo, são serviços oferecidos via web. Em um típico cenário *Web services*, uma aplicação de negócios envia uma requisição de serviço para uma determinada URL usando protocolo *Simple Object Access Protocol* (SOAP) sobre *Hyper Text Markup Language* (HTML).”

Um bom exemplo do funcionamento dos *Web Services* são as requisições, por parte de investidores, dos preços das ações na bolsa de valores. O software cliente envia uma mensagem contendo o código das ações para o servidor *Web services* e este devolve uma resposta contendo o valor das ações naquele determinado momento. Os serviços *Web services* sempre trabalham assim: uma mensagem enviada requer sempre uma outra mensagem de resposta.

No Delphi, os *Web services* são representados por duas classes abstratas básicas: *TInvokable* e *TRemotable*, que juntos formam a base da herança dos objetos que compõem a comunicação entre aplicativos.

TInvokable é uma classe básica que agrupa os métodos que representam as mensagens entre cliente e servidor no protocolo SOAP.

TRemotable é a classe que dá origem as classes que representam os elementos que podem ser enviados através das mensagens.

Segundo a W3C(2002), SOAP provê um simples e leve mecanismo para troca estruturada de informação tipada entre colegas e um descentralizado e distribuído ambiente usando XML. SOAP não define, ele mesmo, qualquer semântica na aplicação como um modelo de programação ou especifica uma implementação específica. Preferivelmente ele define um mecanismo simples para expressar a semântica de aplicativos provendo um modelo de empacotamento modular e mecanismos codificados para codificar informações dentro de módulos. Isto permite ao SOAP ser usado em uma grande variedade de sistemas.

O Soap consiste de três partes:

- a) O construtor de envelope SOAP define um completo sistema para expressar o que é uma mensagem; quem pode tratar com ele, e se isso é opcional ou obrigatório;
- b) O codificador de regras SOAP define uma série de mecanismos que podem ser usados para trocar instâncias de dados definidos pela aplicação;
- c) O RPC SOAP define uma convenção que pode ser usada para representar procedimentos de chamada e resposta remotos.

Os quadros 6 e 7 demonstram o conteúdo de mensagens de requisição e resposta em formato SOAP embutido em *Hipertext transfer protocol* (HTTP).

No quadro 6 tem-se uma requisição SOAP ao servidor chamada *GetLastTradePrice*, requerindo o último preço de venda de um determinado produto que é identificado pelo elemento *symbol*, que contém o valor *DIS*.

Quadro 6 – Exemplo de mensagem de requisição

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI "

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePrice xmlns:m="Some-URI">
      <symbol>DIS</symbol>
    </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Fonte: W3C (2002)

Em resposta à requisição mostrada no quadro 6, o servidor envia uma resposta chamada *GetLastTradePriceResponse* que contém o valor requisitado como conteúdo do elemento *Price*

Quadro 7 – Exemplo de mensagem de resposta

```

HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Fonte: W3C (2002)

Os exemplos demonstram requisições simples e respostas do protocolo SOAP. Eles não parecem muito econômicos no sentido de que carregam muita informação para quem quer transmitir somente 3 ou 4 caracteres. Isso é por causa da generalização do protocolo que o permite ser abrangente e facilmente compreendido por máquinas ou seres humanos. Em alguns casos pode ser preferível abandonar o SOAP por um formato proprietário que reduza a quantidade de informação poupando assim tempo de transmissão sem abandonar o XML.

2.2 A CONTABILIDADE

Segundo Oliveira (1997), nos últimos 30 anos aconteceram mudanças significativas na legislação tributária e nos procedimentos contábeis. O processo artesanal de escrituração foi substituído pelo mecânico e, logo em seguida, pelo automático. As melhorias na forma de fazer a contabilidade de uma empresa, utilizando-se a tecnologia da computação, trouxeram enormes benefícios para os profissionais da área.

O lançamento no diário e no razão tornou-se muito prático, sendo feito de forma simultânea nos sistemas informatizados.

Sem dúvida a contabilidade foi uma das áreas que mais se modificaram com o advento da informática e foi também uma das primeiras. Isso faz com que a informática e a contabilidade sejam hoje parceiras inseparáveis.

2.2.10 PLANO DE CONTAS

A estrutura mais interessante da contabilidade é o plano de contas, que constitui uma árvore com vários níveis em profundidade e possuem quatro contas raízes, que são:

- e) 1-Ativo;
- b) 2-Passivo;
- c) 3-Receita;
- d) 4-Despesa;

Cada empresa pode adotar o plano de contas que melhor convir às suas necessidades, mas em geral o primeiro e o segundo nível do plano de contas é igual na maioria dos casos. O número usado acima, para identificar cada conta raiz, não foram usados ao acaso. Eles costumam ser usados para identificar numericamente cada uma das contas.

As contas estão classificadas em quatro grandes grupos, na estrutura do elenco de contas. Existem planos com um quinto grupo de apuração de resultado, para receber os lançamentos das contas de receitas e despesas antes de os saldos serem transferidos para o Patrimônio Líquido. Isto serve para controle do balancete, sendo que a maioria das empresas usa a própria conta de resultado do exercício, dentro do subgrupo do Patrimônio Líquido, para receber esses lançamentos de encerramento das contas de receitas e despesas (Oliveira,1997).

É possível que uma empresa apresente um plano de contas diferente de outras, porque elas podem ter atividades distintas. Dessa forma, o plano de contas de uma indústria é diferente do plano de contas de uma empresa comercial, que por sua vez é diferente do plano de contas de uma empresa de prestação de serviços.

Existe, contudo, uma estrutura básica, comum à maioria das empresas, que serve de parâmetro para sua organização e implantação.

A correta estruturação do plano de contas é fundamental para que os auxiliares da contabilidade possam desenvolver suas funções, porque evita as constantes consultas aos contadores, o que termina por prejudicar o bom andamento dos trabalhos. (Oliveira,1997).

A estrutura do plano de contas deve conter os seguintes elementos de identificação:

- a) Grupos: é o conjunto de contas que apresentam funções semelhantes, dentro da estrutura do plano, levando-se em consideração sua natureza, finalidade e características. Exemplo: grupo do ativo;
- b) Títulos: nos sistemas manuscritos, era o título ou designação da conta que dava início ao lançamento, identificando as partidas e as contra-partidas. Assim, para se efetuar um lançamento era necessário escrever ou digitar o título da conta devedora e da conta credora, para em seguida completar com data, histórico e valor. A conta credora era antecedida da preposição *a*. Exemplo: *Estoque de mercadorias próprias a Fornecedores*. Os sistemas informatizados trabalham com código, sendo dispensado informar os títulos das contas que, contudo, são necessários para a leitura do balancete e dos demais relatórios, quando impressos ou apresentados no vídeo.
- c) Códigos: os códigos são definidos em função do nível e da natureza da conta. Os sistemas utilizam códigos simplificados ou reduzidos para as contas de movimentação, que são aquelas que aceitam lançamentos e que são chamadas de contas analíticas. As contas sintéticas ou de agrupamento não aceitam lançamentos e os saldos que apresentam são processados pelo sistema. Exemplo: *1.01.00.00.000 – Ativo Circulante (subgrupo de nível 2)*
- d) Níveis: o nível de uma conta pode ser sintético ou analítico, dependendo de seu grau de subordinação. As contas sintéticas não aceitam lançamentos e, portanto não podem ser movimentadas pelos usuários, sendo seu saldo gerado

automaticamente pelo sistema, à medida que são feitos os lançamentos nas contas analíticas do grupo.

- e) Natureza do saldo: os sistemas devem considerar a natureza do saldo das contas em função do grupo a que elas pertencam, atribuindo os sinais de D ou (+) para as devedoras e C ou (-) para as credoras. Assim, ao se cadastrar uma conta de ativo ou despesa deve ser informada no campo próprio, que aquela é uma conta de natureza devedora, indicando o sinal D ou (+). Alternativamente, o balancete pode não precisar apresentar o sinal de débito ou crédito quando relacionar as contas. Esta indicação só se fará necessária se o saldo for diferente do considerado normal para a natureza da conta.

2.2.2 OS LANÇAMENTOS CONTÁBEIS

Os lançamentos contábeis servem para registrar entradas e saídas de valores nas contas contábeis. Por isso, segundo Oliveira(1997) os lançamentos devem ser feitos em partidas dobradas, cada lançamento a débito em uma ou mais contas deve ter o mesmo valor lançado a crédito em uma ou mais contas, onde o valor das partidas de débito deve ser igual ao valor das partidas de crédito.

Nos quadros 8, 9, 10 e 11 vê-se exemplos de lançamentos contábeis com partidas simples e compostas:

Quadro 8 – Lançamento com partidas simples

1ª. Formula: Uma conta devedora e uma credora	
Operação: 01 [01-Incluir 02-Consultar 03-Alterar 04-Excluir]	
Conta Devedora: Fornecedores de materiais	Código: x.xx.xx.xx.xxx
Conta Credora: Bancos conta movimento	Código: x.xx.xx.xx.xxx
Histórico: Paço através do ch.002/Banco AXZ, ref. nf. 00101	
Valor: R\$ 50.000,00	

Quadro 9 – Lançamento com duas partidas de crédito

2ª. Formula: Uma conta devedora e mais de uma credora		
Operação: 01 [01-Incluir 02-Consultar 03-Alterar 04-Excluir]		
Conta Devedora: Despesas de Salário	Cód.: x.xx.xx.xx.xxx	Valor: R\$ 370.000,00
Conta Credora.: Bancos conta movimento	Cód.: x.xx.xx.xx.xxx	Valor: R\$ 345.000,00
Consignações a pagar	Cód.: x.xx.xx.xx.xxx	Valor: R\$ 25.000,00
Histórico: Valor da provisão da folha do mês 10/xx		

Quadro 10 – Lançamento com duas partidas de débito

3ª. Formula: Mais de uma conta devedora e somente uma credora		
Operação: 01 [01-Incluir 02-Consultar 03-Alterar 04-Excluir]		
Conta Credora.: Previdência social a pg.	Cód.: x.xx.xx.xx.xxx	Valor: R\$ 200.000,00
Multa por atraso	Cód.: x.xx.xx.xx.xxx	Valor: R\$ 20.000,00
Conta Devedora: Banco AXZ conta movim.	Cód.: x.xx.xx.xx.xxx	Valor: R\$ 220.000,00
Histórico: Pago a previdência social c/multa por atraso, ch. 102022 banco AXZ		

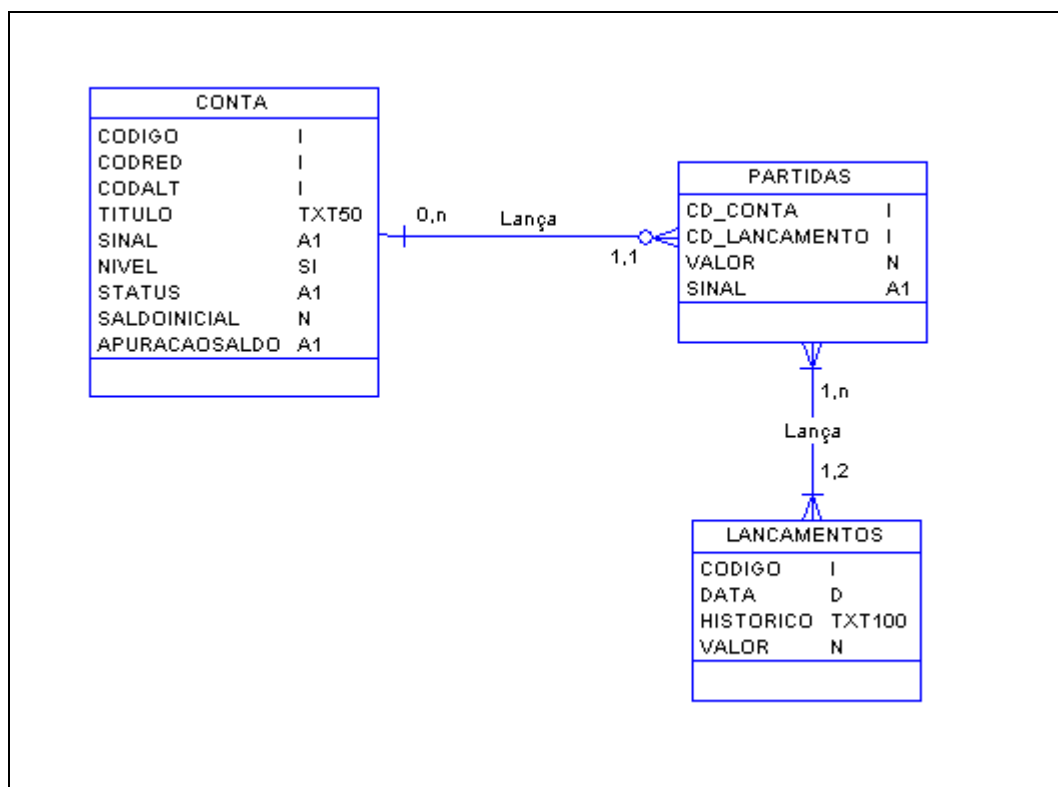
Quadro 11 – Lançamento com duas contas de débito e duas de crédito

4ª. Formula: Mais de uma conta devedora e credora		
Operação: 01 [01-Incluir 02-Consultar 03-Alterar 04-Excluir]		
Conta Credora.: Banco conta movimento	Cód.: x.xx.xx.xx.xxx	Valor: R\$ 25.500,00
Despesas bancárias	Cód.: x.xx.xx.xx.xxx	Valor: R\$ 1.000,00
Conta Credora.: Créditos a rec. cartão A	Cód.: x.xx.xx.xx.xxx	Valor: R\$ 15.000,00
Créditos a rec. cartão B	Cód.: x.xx.xx.xx.xxx	Valor: R\$ 11.500,00
Histórico: Valor creditado em c/c, ref. Cartões A e B, deduzido das despesas bancárias		

2.2.3 ENTIDADES CONTÁBEIS

Finalmente em um sistema contábil tem-se no mínimo três entidades que por si só já formam a base de um sistema contábil. Estas entidades são: as Contas, os Lançamentos e as Partidas. Na figura 2, essas três entidades são mostradas em um modelo de entidade/relacionamento:

Figura 2 – Entidades/relacionamento da contabilidade



Em um sistema contábil informatizado existem dezenas de entidades a mais que as três mostradas no diagrama, mas o enfoque do trabalho é justamente a essência do sistema contábil. Se fosse realmente construir um sistema contábil teria-se que acrescentar entidades como “Notas fiscais” e “Taxas de Impostos”, para que o sistema pudesse fazer cálculos de impostos e os respectivos lançamentos. Assim, estaria-se também restringindo esse modelo contábil a um modelo brasileiro e é justamente isso que não se quer. Em um modelo orientado a objetos poder-se-ia dizer que essas entidades são classes abstratas e que são o topo da hierarquia de um sistema contábil. As peculiaridades de cada tipo de sistema contábil podem agora ser desenvolvidas com base nessa estrutura básica.

3 DESENVOLVIMENTO DO TRABALHO

Foram explanadas no capítulo anterior as tecnologias utilizadas para o desenvolvimento do aplicativo.

Neste capítulo abordar-se o resultado do uso das técnicas bem como as fases da especificação e da implementação dos aplicativos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O objetivo deste trabalho, como já mencionado, é a construção de um padrão de comunicação de dados contábeis usando XML. Dentre as tecnologias disponíveis na XML foram escolhidas o XML Schema para trabalhar com Web Services.

O desenvolvimento do padrão pode parecer muito com a definição de entidades em um banco de dados, mas o objetivo do trabalho é somente abstrair os elementos fundamentais para fazer funcionar um sistema contábil extremamente simples e colocá-los em funcionamento na tecnologia da web.

É sim um objetivo do trabalho, mostrar que os sistemas contábeis atuais não fazem uma real análise de sistemas, mas sim constróem aplicativos que se adaptam aos sistemas já existentes nos escritórios contábeis. Deve-se observar que a palavra “sistema” aqui mencionada não se refere a um aplicativo ou programa de computador como é geralmente utilizada para mencionar. Na expressão “sistema contábil” a palavra sistema simboliza o software desenvolvido para a contabilidade, quando na verdade, “sistema contabil” designa algo mais amplo que somente o software, incluindo ai toda a rotina de um escritório contábil com ou sem um software.

Por uma questão de simplificação e por não se tratar de um aplicativo comercial de verdade, a questão da segurança de acesso aos dados não será levada em consideração nesse trabalho, apesar de ser algo profundamente importante caso o padrão seja aplicado comercialmente na web.

Outro ponto que não entrará em questão é a multiplicidade de entidades contabilizadas visto que essa questão é resolvida instanciando-se vários servidores para representar cada entidade que se desejar. Embora, a visão de que um servidor deva conter várias entidades contábeis não seja descartada, visto a economia de recursos e também a facilidade em relacionar uma entidade com outra.

Quanto à funcionalidade do padrão, ele deverá ser capaz de:

- a) cadastrar todas as entidades envolvidas no padrão;
- b) recuperar as entidades já cadastradas;
- c) as entidades devem ter identificadores únicos que permitam que os sistemas que agreguem-se ao padrão possam referenciar cada entidade sem conflitos.

3.2 ESPECIFICAÇÃO

Para especificar o padrão foi usado o próprio XML Schema que além de funcionar como especificador entre máquinas, é legível, e pode ser usado para visualização tão bem como se fosse escrito em arquivo texto.

Para auxiliar a especificação do problema utilizou-se UML (*Unified Modeling Language*), visto que a XML Schema não conseguirá mostrar graficamente os relacionamentos e as funcionalidades do padrão.

Foram utilizados os seguintes diagramas baseados na UML:

- a) diagrama de casos de uso;
- b) diagrama de classes;

O diagrama de casos de uso foi utilizado para demonstrar as possibilidades de uso sobre o padrão desenvolvido.

3.2.1 O PADRÃO, A CONTABILIDADE

A contabilidade, conforme descrita na fundamentação teórica possui duas entidades, *contas* e *lançamentos*. Essas duas entidades básicas desdobram-se em três entidades, visto que uma conta possui diversos lançamentos e um lançamento possui diversas contas formando um relacionamento n para m. A terceira entidade, chamada aqui de *partidas*, transforma-se em uma entidade de relacionamento, quando esse deve carregar outros campos além dos códigos das duas entidades relacionadas.

Um lançamento possui no mínimo duas contas, sendo uma conta de débito e uma conta de crédito. O valor de um lançamento deve ser igual a soma dos valores das partidas de débito ou a soma do valor das partidas de crédito, ou seja, a soma das partidas de débito menos as somas das partidas de crédito devem sempre ser zero.

Uma partida, além de conter um valor, pode conter uma descrição - apesar de que não faria diferença no funcionamento da contabilidade - que é algo fundamental para a interface com o usuário e visto isso a descrição entra no padrão e em todas as entidades.

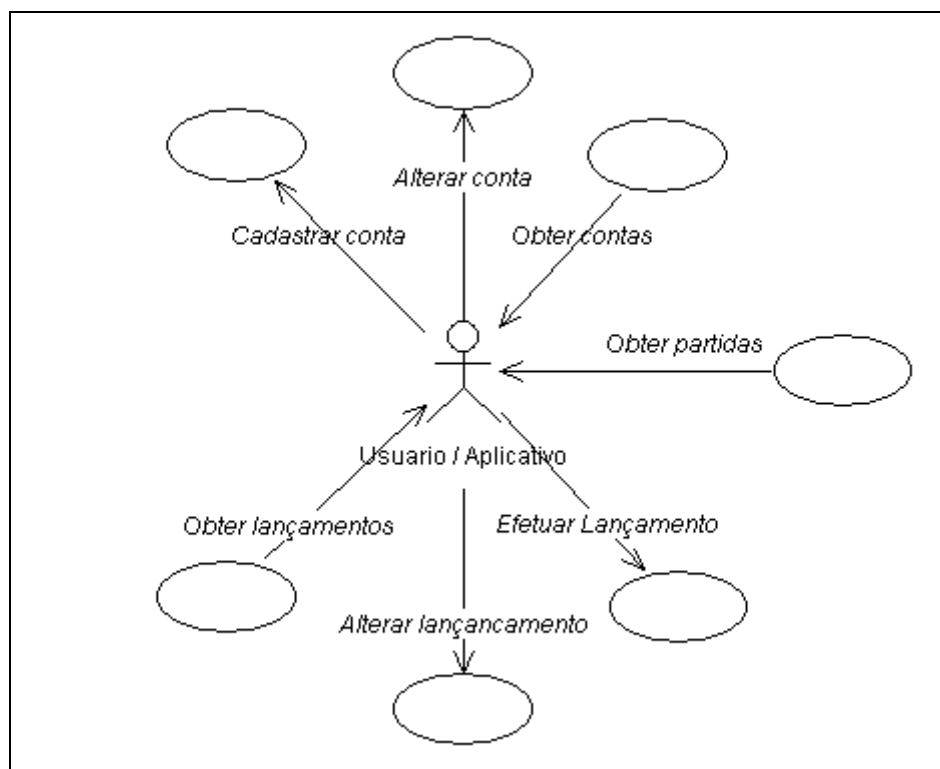
Uma das propriedades mais importantes de todos os elementos são os identificadores únicos, pois eles são a capacidade do padrão de evoluir. Cada instância de um elemento ganha um único identificador que é único também entre todos os outros elementos dentro do sistema.

3.2.2 DIAGRAMA DE CASOS DE USO

O diagrama de casos de uso mostra a funcionalidade ou o comportamento de um aplicativo ou sistema interagindo com um ou mais atores externos.

A figura 3 mostra a interação de um usuário ou aplicativo com o servidor contábil. Há somente as iterações de *cadastro*, *alteração* e *obtenção* para cada elemento, que neste caso são as contas e os lançamentos e as partidas, sendo as partidas as únicas que não sofrem cadastro e alteração direta do ator.

Figura 3 – Diagrama de casos de uso



Mesmo que pareça bastante simples, o diagrama demonstra as únicas tarefas que o servidor deve executar. Na construção de relatórios personalizados essas funções formam a base da obtenção de valores.

A intenção ao descrever o ator como Usuário/Aplicativo é simplesmente demonstrar que as ações podem partir tanto de um usuário humano como de outro aplicativo, que pode tanto ser um servidor ou qualquer espécie de evento gerado automaticamente.

O diagrama de casos também não aborda os eventos que podem ocorrer com o cliente e o servidor, visto que não é objetivo deste trabalho discutir o funcionamento dos clientes e dos servidores, mesmo porque esses aplicativos somente foram desenvolvidos para demonstrar o funcionamento do padrão e porque segundo a idéia do trabalho o objetivo é criar um padrão de comunicação que seja independente de qualquer plataforma. Por isso atem-se somente aos eventos próprios da comunicação contábil.

Os procedimentos que processam as requisições, citadas no diagrama de casos de usos, pelo servidor podem ser vistos no anexo 3 e a seguir são descritas detalhadamente:

- a) cadastrar conta – esta requisição inclui uma conta no registro de contas do servidor. Para isso são passados os parâmetros, IDAnt, Código, Descrição, Sinal, e Status. Como resposta o servidor envia uma mensagem informando o identificador único da conta que também pode acusar um erro se esse for menor que zero.
- b) alterar conta – esta requisição altera uma conta no registro de contas do servidor. Isso é feito passando-se como parâmetro um objeto do tipo TConta, contendo todos os dados inclusive os não modificadores. A resposta a essa requisição é somente uma mensagem confirmando o recebimento do pacote.
- c) obter contas – esta requisição faz com que o servidor retorne uma ou mais contas. Existem duas formas de obter contas para esse caso. A primeira é através da mensagem *GetContaRequest* que enviando-se como parâmetro o identificador único da conta, devolve-se ou não a conta encontrada. O segundo é através da mensagem *GetChildRequest* que envia como parâmetro o identificador único do pai das contas que se esta requisitando. Essa mensagem pode devolver uma lista de contas contendo nenhuma conta ou várias contas.
- d) efetuar lançamento – esta requisição funciona como cadastrar conta. São passados como parâmetros todas as propriedades de um lançamento que são: a data, a descrição, o valor e as partidas. Como retorno da requisição é enviado o identificador único do lançamento ou então um valor menor que zero em caso de erro.
- e) alterar lançamento – funciona como alterar conta. Passam-se como parâmetro uma instância da classe *TLancamento* contendo as

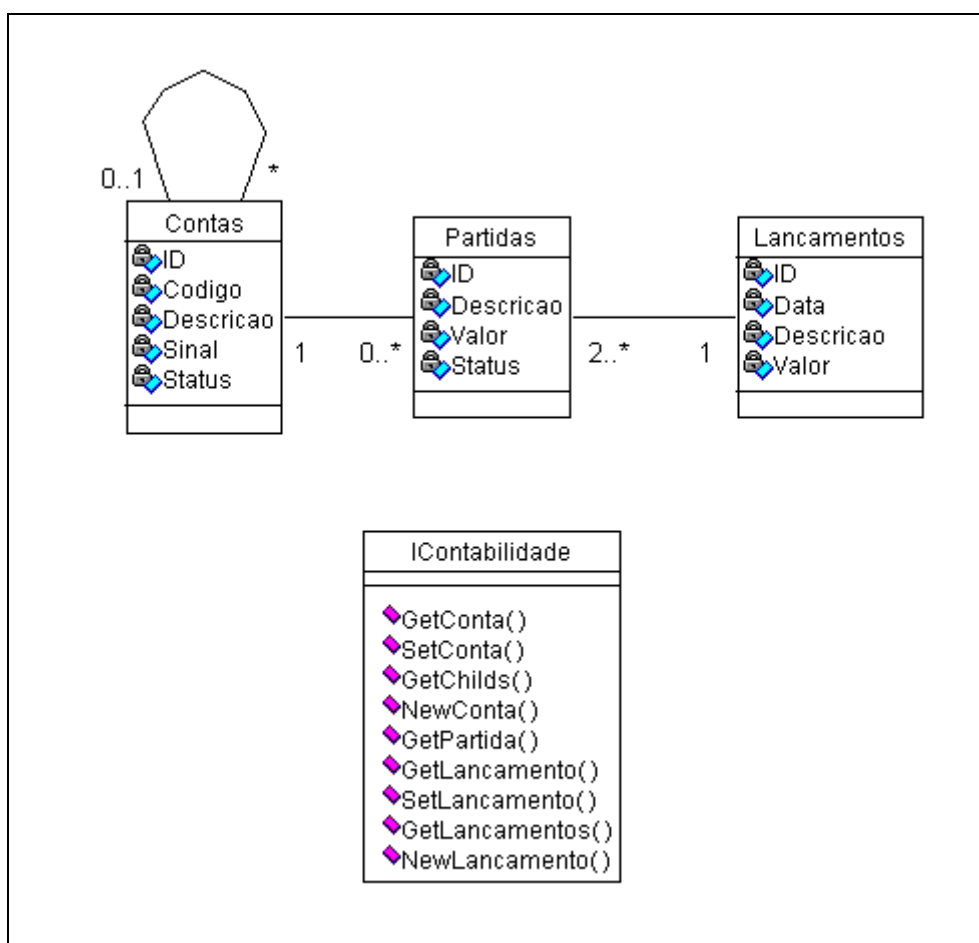
modificações requeridas. Tem-se como resposta uma mensagem confirmando o recebimento da requisição.

- f) obter lançamento – funciona de duas formas. Uma para obter um lançamento específico, através de seu identificador único e a outra, que devolve múltiplos lançamentos, através da posição relativa a lista de lançamentos no servidor e a quantidade de lançamentos requerida.
- g) obter partida – funciona de uma única forma, onde é passado como parâmetro o identificador único de um lançamento e tem-se como resposta uma lista de partidas que compõem aquele lançamento.

3.2.3 DIAGRAMA DE CLASSES

O diagrama de classes mostra a estrutura das classes com seus atributos, métodos, heranças, ligações e agregações. No diagrama de classes da figura 4 vê-se um típico diagrama na linguagem UML. Este diagrama somente mostra as classes com seus atributos e ligações com suas respectivas cardinalidades.

Figura 4 – Diagrama de classes



A classe *Contas* contém as contas do plano de contas contábil. Existe uma ligação de 1 para m dela para ela mesmo. Essa ligação é feita para que possa ser montada uma estrutura em árvore, sendo que as contas que não são referenciadas por outra conta são chamadas “contas raízes”. As contas que não referenciam outras contas são chamadas - segundo a contabilidade - *analíticas* e as contas que se referenciam a

outras contas são consideradas *sintéticas*. A classe *contas* também possui ligação de um para n com a classe *partidas* significando que uma conta pode possuir muitas partidas.

A classe *Lancamentos* armazena cada ocorrência onde haja débitos e créditos entre as contas. O atributo *Data* guarda a data em que ocorreu a movimentação, o atributo *Descricao* guarda a descrição ou a explicação do lançamento e o atributo *Valor* armazena, ou não, o valor que foi transferido. É dito que o atributo *Valor* “pode ou não armazenar”, pois o campo valor é a soma dos valores das partidas de débito ou a soma das partidas de crédito. A opção de guardar o valor junto com os outros atributos de um lançamento é uma questão de performance no servidor, visto que possa ser extremamente lento a tarefa de somar as partidas quando um ou mais lançamentos forem requisitados. No caso em que o valor deva ser gravado junto ao lançamento, isso pode ser feito quando for cadastrado o lançamento e suas partidas.

A classe *Partida* vem para complementar os lançamentos e ainda fazer a ligação entre os lançamentos e as contas. Toda partida está ligada a uma conta e a um lançamento. Um Lançamento por sua vez está sempre ligado a duas partidas: uma partida de débito e uma partida de crédito. Na contabilidade é comum que um lançamento possa ter uma conta de crédito e várias contas de débito e vice-versa.

A classe *partida* possui os atributos: *descricao* que descreve mais precisamente a ocorrência desta partida; *valor* que armazena o valor movimentado da conta; e *status*, que armazena os caracteres “D” ou “C”, indicando se o valor da partida está sendo debitado ou creditado na conta.

A classe *IContabilidade* funciona como o elo de ligação entre um cliente e um servidor. Ela carrega os métodos que fazem a comunicação através de mensagens entre cliente e servidor. Os métodos nela descritos estão citados no diagrama de casos de uso e a implementação pode ser vista no anexo 3.

3.2.4 O PADRÃO EM XML SCHEMA

O XML Schema usado nesse sistema foi gerado pelas ferramentas de web services do programa Borland Delphi 6 com base em objetos construídos no servidor para a comunicação, baseados no mesmo esquema.

O esquema é sempre fornecido quando se acessa o servidor, seguido do texto “/wsdl”. Um exemplo pode ser: “http://localhost/tcc/contabserv.dll/wsdl”

Assim, qualquer aplicativo que queira interagir com o servidor pode simplesmente ler o esquema e fabricar as ferramentas necessárias para se comunicar.

O esquema gerado para o padrão foi dividido em partes e colocados nos capítulos seguintes.

3.2.4.1 A DEFINIÇÃO DOS ELEMENTOS

Como explicado na fundamentação teórica, as ferramentas Web Services normalmente geram um texto XML que segue o formato XML Schema que descrevem a estrutura de comunicação do servidor.

Analisar esta estrutura em XML Schema pode ser a melhor forma de identificar a estrutura de cada elemento.

A seguir são passados todos os elementos do projeto, com suas respectivas definições:

A estrutura no quadro 12 demonstra a estrutura de um elemento complexo chamado *TConta*.

Quadro 12 – Estrutura de TConta

```
<xs:complexType name="TConta">
  <xs:sequence>
    <xs:element name="ID" type="xs:long"/>
    <xs:element name="IDAnt" type="xs:long"/>
    <xs:element name="Codigo" type="xs:string"/>
    <xs:element name="Descricao" type="xs:string"/>
    <xs:element name="Sinal" type="xs:string" length="1"/>
    <xs:element name="Status" type="xs:string" length="1"/>
  </xs:sequence>
</xs:complexType>
```

```
</xs:sequence>
</xs:complexType>
```

O primeiro elemento chamado *complextype* explica que se trata de um elemento complexo ou um elemento que possui outros elementos. Contido neste elemento está o elemento *sequence* que indica que os elementos nele contidos devem seguir a mesma seqüência em que estão apresentados no esquema.

Os elementos de tag *element* representam elementos simples. Os elementos simples podem possuir diversos atributos entre nome, tipo, referência e atributos de restrição. Esses elementos representam os valores atômicos do elemento citado.

O elemento de nome *ID* representa o identificador único da conta e do elemento contábil. *ID* é do tipo *long*, que é um tipo derivado de *integer* e permite conter um número inteiro de 64 bits e permite uso do sinal negativo.

O elemento de nome *IDAnt* representa o identificador da conta pai do elemento citado. Se *IDAnt* for nulo indicará que esta conta não tem pai e será uma conta raiz. Como *ID*, *IDAnt* também é tipo *long*.

O elemento *Código* representa um campo do tipo *string* de tamanho livre que armazena um código qualquer para uma identificação mnemônica da conta ou então algum código qualquer escolhido pelo usuário. O campo de código representa uma identificação pública da conta visto que o campo *ID* não deveria ser visível ao usuário, sendo usado somente para referência interna.

O elemento *Descricao* funciona como um descritor ou nomeador da conta do tipo *string*, também de tamanho livre.

O elemento *Sinal* identifica se a conta é uma conta de devedora “D” ou uma conta credora “C”. *Sinal* é do tipo *string* de tamanho 1 e somente pode conter os caracteres “D” ou “C”. O tamanho do valor do elemento é regulamentado pelo atributo *length*.

O próximo elemento complexo é uma estrutura para armazenar uma lista de contas. *TContaArray* fornece suporte para funções que queiram obter mais de uma conta como retorno de uma requisição.

Quadro 13 – Estrutura de TContaArray

```
<xs:complexType name="TContaArray">
  <xs:complexContent>
    <xs:restriction base="soapenc:Array">
      <xs:sequence/>
      <xs:attribute ref="soapenc:arrayType"
n1:arrayType="nsl:TConta[]"
xmlns:n1="http://schemas.xmlsoap.org/wsdl/" />
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

TContaArray também é definido como um elemento complexo e o elemento *complexContent* contido em *complexType* informa que o conteúdo do elemento será estendido. O elemento *restriction* deriva este elemento do elemento de base *soapenc:Array*. O elemento *attribute* define um atributo para o elemento que neste caso se referencia ao tipo *soapenc:arraytype*.

O elemento *TPartida* (quadro 14) representa a interface para a comunicação de registros de partida entre objetos web services.

Quadro 14 – Estrutura de TPartida

```
<xs:complexType name="TPartida">
  <xs:sequence>
    <xs:element name="ID" type="xs:long"/>
    <xs:element name="IDConta" type="xs:long"/>
    <xs:element name="IDLancamento" type="xs:long"/>
    <xs:element name="Descricao" type="xs:string"/>
    <xs:element name="Valor" type="xs:double"/>
    <xs:element name="Status" type="xs:string" length="1"/>
  </xs:sequence>
</xs:complexType>
```

O elemento nomeado *ID* representa, como em *TConta*, o identificador único da partida e é do mesmo tipo e tem o mesmo funcionamento que no elemento *TConta*.

O elemento nomeado *IDConta* referência a conta a quem essa partida pertence sendo do mesmo tipo que os elementos *ID*.

O elemento nomeado *IDLancamento* referência o lançamento que originou essa partia. Tem o mesmo tipo e funcionamento que *IDConta*.

O elemento nomeado *Descricao*, como os demais elementos de descrição, é usado para descrever os detalhes dessa partida em particular.

O elemento nomeado *Valor* armazena o valor que a partida movimentou na conta. Esse valor é do tipo *double*, que pode armazenar números com casas decimais utilizando uma estrutura de 64 bits.

O elemento nomeado *Status* armazena uma *string* que contém os caracteres “D” ou “C”, informando se os valor da partida ocasionou um débito ou um crédito na conta da partida.

O elemento nomeado *TPartidaArray* (quadro 15), como *TContaArray*, é uma estrutura montada para carregar uma lista de elementos *TPartida*.

Quadro 15 – Estrutura de TPartidaArray

```
<xs:complexType name="TPartidaArray">
  <xs:complexContent>
    <xs:restriction base="soapenc:Array">
      <xs:sequence/>
      <xs:attribute ref="soapenc:arrayType"
n1:arrayType="nsl:TPartida[]"
xmlns:n1="http://schemas.xmlsoap.org/wsdl/" />
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>
```

Seu funcionamento é idêntico ao funcionamento de *TContaArray* sendo que o atributo *n1:arraytype* passa a referenciar-se a *nsl:TPartida[]*.

O elemento nomeado *Tlancamento* (quadro 16) representa a interface para a comunicação de registro de lançamentos.

Quadro 16 – Estrutura de TLancamento

```

<xs:complexType name="TLancamento">
  <xs:sequence>
    <xs:element name="ID" type="xs:long"/>
    <xs:element name="Data" type="xs:TDateTime"/>
    <xs:element name="Descricao" type="xs:string"/>
    <xs:element name="Valor" type="xs:double"/>
    <xs:element name="Partidas" type="nsl:TPartidaArray"/>
  </xs:sequence>
</xs:complexType>

```

O elemento nomeado *ID*, como visto em outros elementos, é o identificador único de um lançamento.

O elemento nomeado *Data* armazena a data da ocorrência do lançamento. *Data* é do tipo *TDateTime*, que não pertence ao conjunto de tipos definidos pelo XML Schema. *TDateTime* é um tipo definido pelas ferramentas web services existentes no Borland Delphi.

O elemento nomeado *Descricao* armazena a descrição às vezes chamada histórico do lançamento.

O elemento nomeado *Valor*, do tipo *double*, fornece o total do valor do lançamento.

O elemento nomeado *Partidas*, do tipo *TPartidasArray*, é a parte interessante do elemento de lançamento. Esse elemento carrega consigo a lista de partidas que compõem o lançamento. Tanto para efeito de cadastro como para efeito de consulta as partidas devem sempre acompanhar o lançamento.

O elemento nomeado *TLancamentoArray* (quadro 17) possui o mesmo funcionamento dos outros elemento de array. Sua função é transmitir uma lista de elementos *TLancamento*.

Quadro 17 – Estrutura de TLancamentoArray

```

<xs:complexType name="TLancamentoArray">
  <xs:complexContent>
    <xs:restriction base="soapenc:Array">

```

```

        <xs:sequence/>
        <xs:attribute ref="soapenc:arrayType"
n1:arrayType="nsl:TLancamento[]"
xmlns:n1="http://schemas.xmlsoap.org/wsdl/" />
    </xs:restriction>
</xs:complexContent>
</xs:complexType>

```

3.2.4.2 AS MENSAGENS

As mensagens funcionam como requisições de serviços para os servidores e respostas do servidor ao serviço requisitado. Toda a estrutura de mensagens foi mostrada em pares, sendo uma a requisição e outra a resposta, quando existir. Nota-se que as mensagens não fazem parte do conjunto de elemento pré-definidos pelo XML Schema. As mensagens, como a estrutura de tipos, fazem parte da estrutura do SOAP, que roda sobre o XML Schema.

As duas mensagens do quadro 18 referenciam o método *GetConta* do servidor que devolve uma conta baseada no identificador enviada pela requisição.

Quadro 18 – Estrutura de *GetContaRequest* e *GetContaResponse*

```

<message name="GetContaRequest">
  <part name="ID" type="xs:long" />
</message>

<message name="GetContaResponse">
  <part name="return" type="nsl:TConta" />
</message>

```

O elemento de mensagem com o atributo *name* igual a *GetContaRequest* é uma mensagem enviada para o servidor contendo um o parâmetro *id* que identifica a conta que está sendo requisitada. O servidor responde a essa requisição enviando a mensagem *GetContaResponse* com um parâmetro chamado *return*, contendo um elemento do tipo *TConta*

A mensagem *GetChildsRequest* (quadro 19) é uma requisição de sub-contas de uma conta. Seu funcionamento é muito parecido com a mensagem *GetContaRequest*

com a diferença de que o parâmetro *ID* enviado na mensagem referencia o identificador do pai das contas que deveram retornar, ou seja, *GetContaRequest* envia um *ID* e retorna a conta referente a esse *ID*. Em *GetChildsRequest* envia-se um *ID* e retorna as contas que referenciam esse *ID* no campo *IDAnt*. O retorno desta mensagem é feito através da mensagem *GetChildsResponse* que não devolve somente uma conta, mas uma lista de contas através do da estrutura *TContaArray*..

Quadro 19 - Estrutura de *GetChildsRequest* e *GetChildsResponse*

```
<message name="GetChildsRequest">
  <part name="ID" type="xs:long"/>
</message>

<message name="GetChildsResponse">
  <part name="return" type="nsl:TContaArray"/>
</message>
```

A mensagem *NewContaRequest* (quadro 20) é usada para criar uma nova conta. Nesta mensagem são passados parâmetros que correspondem aos campos de uma conta com exceção dos campos de não são de domínio do usuário ou cliente como é o caso do *ID*. Em resposta a essa mensagem é enviada a mensagem *NewContaResponse* que retorna o número do *ID* gerado pelo servidor e que identifica a nova conta gerada. Em casos de erro no cadastramento é retornado o valor -1, que em todos os casos envolvendo *IDs* e o mesmo que nulo.

Quadro 20 – Estrutura de *NewContaRequest* e *NewContaResponse*

```
<message name="NewContaRequest">
  <part name="IDAnt" type="xs:long"/>
  <part name="Codigo" type="xs:string"/>
  <part name="Descricao" type="xs:string"/>
  <part name="Sinal" type="xs:string"/>
  <part name="Status" type="xs:string"/>
</message>

<message name="NewContaResponse">
  <part name="return" type="xs:int"/>
</message>
```

A mensagem *GetPartidaRequest* (quadro 21) requisita uma partida com base em seu identificador, que é enviado por parâmetro. O retorno em *GetPartidaResponse* devolve uma única partida, se ela existir.

Quadro 21 – Estrutura de *GetPartidaRequest* e *GetPartidaResponse*

```
<message name="GetPartidaRequest">
  <part name="ID" type="xs:long"/>
</message>

<message name="GetPartidaResponse">
  <part name="return" type="nsl:TPartida"/>
</message>
```

GetLancamentoRequest (quadro 22) requisita um lançamento com base em seu indicador, passado como parâmetro. Em resposta é enviada a mensagem *GetLancamentoResponse* com um único lançamento existente.

Quadro 22 – Estrutura de *GetLancamentoRequest* e *GetLancamentoResponse*

```
<message name="GetLancamentoRequest">
  <part name="ID" type="xs:long"/>
</message>

<message name="GetLancamentoResponse">
  <part name="return" type="nsl:TLancamento"/>
</message>
```

GetLancamentosRequest (quadro 23) retorna uma lista de lançamentos com base em sua ordem de cadastramento. Essa mensagem possui dois parâmetros que filtram os lançamentos que devem ser enviados. O primeiro parâmetro, *Inicio*, indica o ponto em que começa a lista que esta ordenada decrescentemente pela data de lançamento e *MaxResult* indica a quantidade de lançamento que devem ser devolvidos.

A mensagem *GetLancamentosResponse* devolve uma lista de lançamentos que foram requisitados em *GetLancamentosRequest*.

Quadro 23 – Estrutura de GetLancamentosRequest e GetLancamentosResponse

```

<message name="GetLancamentosRequest">
  <part name="Inicio" type="xs:int"/>
  <part name="MaxResult" type="xs:int"/>
</message>

<message name="GetLancamentosResponse">
  <part name="return" type="nsl:T LancamentoArray"/>
</message>

```

Essas mensagens funcionam como em um site de busca onde o usuário faz uma busca e tem como retorno uma lista de resultados com uma certa quantidade de itens. Se o usuário desejar ver mais resultados ele deverá requisitar os “próximos” daquela pesquisa.

Nos casos em que o usuário desejar voltar na lista, o parâmetro *MaxResult* deverá ser negativo, o que fará com que o servidor inverta a seqüência de busca.

A mensagem *NewLancamentoRequest* (quadro 24) serve para adicionar um novo registro. Nela são enviadas as informações do novo lançamento que incluem: a data do lançamento, a descrição do lançamento, o valor total do lançamento e uma lista de partidas que compõem o lançamento. O valor total do lançamento serve somente para conferência já que o servidor atribuirá o valor do lançamento com base na soma do valor das partidas enviadas. O retorno desta requisição é feito pela mensagem *NewLancamentoResponse*, que não devolve parâmetros.

Quadro 24 – Estrutura de NewLancamentoRequest

```

<message name="NewLancamentoRequest">
  <part name="Data" type="xs:string"/>
  <part name="Descricao" type="xs:string"/>
  <part name="Valor" type="xs:double"/>
  <part name="Partidas" type="nsl:TPartidaArray"/>
</message>

<message name="NewLancamentoResponse"/>

```

3.3 IMPLEMENTAÇÃO

A implementação divide-se em duas fases, a implementação do servidor e a implementação de clientes. Serão demonstradas as ferramentas e técnicas utilizadas para a construção do servidor e a construção de um único cliente que servirá para demonstrar a funcionalidade do servidor.

3.3.1 TÉCNICAS E FERRAMENTAS UTILIZADAS

As Técnicas utilizadas foram:

- a) Análise orientada a objetos usando UML;
- b) O XML Schema, para a estruturação do padrão;
- c) Objetos Web Services;

As ferramentas utilizadas foram:

- a) Rational Rose para a especificação dos objetos;
- b) Interbase/Firebird como ferramenta de banco de dados que armazena os dados;
- c) Microsoft Internet Information Services que é utilizado para rodar os CGIs do servidor;
- d) Borland Delphi 6 como ambiente visual de desenvolvimento;

3.3.1.1 OS OBJETOS NO SERVIDOR

A primeira fase da implementação constituiu-se da construção dos objetos de interface de acordo com o diagrama de classes. Esses objetos formam a base para a comunicação com o servidor.

Além dos objetos citados no diagrama de classes, outras estruturas foram criadas para dar funcionalidade ao sistema como: *TContaArray*, *TPartidaArray* e *TLancamentoArray*. Que servem para transportar listas de cada classe de objeto.

Todos os objetos são descendentes do objeto *TRemotable* que tem a capacidade de ser referenciado ou passado como parâmetro e retornar valores nas ferramentas Web Services.

Abaixo estão as descrições dos objetos que foram implementados no servidor.

O objeto *TConta* (quadro 25) provê interface para todos os “negócios” realizados com contas contábeis.

Quadro 25 – Classe TConta

```
TConta = Class( TRemotable )
private
    fID          : Int64;
    fIDAnt       : Int64;
    fCodigo      : WideString;
    fDescricao   : WideString;
    fSinal       : Char;
    fStatus      : Char;
published
    property ID          : Int64          Read fID          Write fID;
    property IDAnt      : Int64          Read fIDAnt       Write fIDAnt;
    property Codigo     : WideString     Read fCodigo      Write fCodigo;
    property Descricao  : WideString     Read fTitulo      Write fTitulo;
    property Sinal      : Char           Read fSinal       Write fSinal;
    property Status     : Char           Read fStatus      Write fStatus;
end;
```

O objeto *TPartida* (quadro 26) provê interface para todos os negócios realizados com as partidas entre contas contábeis e lançamentos.

Quadro 26 – Classe TPartida

```
TPartida = Class( TRemotable )
private
    fID          : Int64;
    fIDConta     : Int64;
    fIDLanca     : Int64;
    fDesc        : WideString;
    fValor       : Double;
    fStatus      : WideString;
published
    property ID          : Int64          Read fID          Write fID;
```

```

property IDConta      : Int64      Read fIDConta Write fIDConta;
property IDLancamento: Int64      Read fIDLanca Write fIDLanca;
property Descricao    : WideString Read fDesc    Write fDesc;
property Valor        : Double      Read fValor   Write fValor;
property Status       : WideString  Read fStatus  Write fStatus;
end;

```

O objeto *TLancamento* (quadro 27) provê interface para todos os negócios realizados com lançamentos contábeis.

Quadro 27 – Classe TLancamento

```

TLancamento = Class( TRemotable )
private
  fID      : Int64;
  fData    : TDateTime;
  fDesc    : WideString;
  fValor   : Double;
  fPartidas : TPartidaArray;
public
  procedure AddPartida( Value : TPartida );
published
  property ID      : Int64      Read fID      Write fID;
  property Data    : TDateTime  Read fData    Write fData;
  property Descricao: WideString Read fDesc    Write fDesc;
  property Valor   : Double      Read fValor   Write fValor;
  property Partidas: TPartidaArray Read fPartidas Write fPartidas;
end;

```

TContaArray, *TPartidaArray* e *TLancamentoArray* foram implementados para capacitar o servidor a trabalhar com listas de objetos. Essa estrutura é fundamental para que o retorno de uma requisição possa devolver uma série de valores de mesmo tipo ou no caso de cadastro de um novo lançamento possa ser enviada uma lista de partidas que irão compor esse lançamento.

Quadro 28 – Definições de TContaArray, TPartidaArray e TLancamentoArray

```

TContaArray = Array of TConta;

TPartidaArray = Array of TPartida;

TLancamentoArray = Array of TLancamento;

```

A classe *Icontabilidade* (quadro 27) é uma classe muito importante para o funcionamento de um servidor Web Services. É ela quem provê a funcionalidade do servidor. Os métodos declarados nessa classe são disponibilizados pelo servidor para uso dos clientes. Cada método declarado em *IContabilidade* corresponde às mensagens declaradas na especificação do projeto.

Quadro 29 – Definição de interface de IContabilidade

```
IContabilidade = interface(IInvokable)
[ '{1AF5F692-49F7-492C-A27F-65C13CF00324}' ]
function GetConta( ID : Int64 ): TConta; stdcall;
function GetChilds( ID : Int64 ): TContaArray; stdcall;
function NewConta( IDAnt : Int64; Codigo, Titulo :
WideString; Sinal, Status : Char; SaldoIni :
Double ): Integer; stdcall;
function GetPartida( ID : Int64 ): TPartida; stdcall;
function GetLancamento( ID : Int64 ): TLancamento; stdcall;
function GetLancamentos( Inicio : Integer; MaxResult :
Integer ): TLancamentoArray; stdcall;
procedure NewLancamento( Data, Descricao : String; Valor :
Double; Partidas : TPartidaArray ); stdcall;
end;
```

3.3.1.2 OS OBJETOS NOS CLIENTES

A tarefa de criar um cliente que se comunique com o servidor é uma tarefa bem simples usando os *wizards* do Borland Delphi 6.

Todo servidor Web Services construído em Delphi 6 fornece, acessando o diretório `/wsdl` do CGI do servidor, um documento XML com as estruturas disponibilizadas pelo servidor. O *wizard* do Delphi lê esta estrutura e monta as classes de interface necessárias para o funcionamento. Daí em diante basta utilizar as classes criadas para interagir com o servidor. As classes geradas podem ser vistas no anexo 1.

3.3.2 OPERACIONALIDADE DA IMPLEMENTAÇÃO

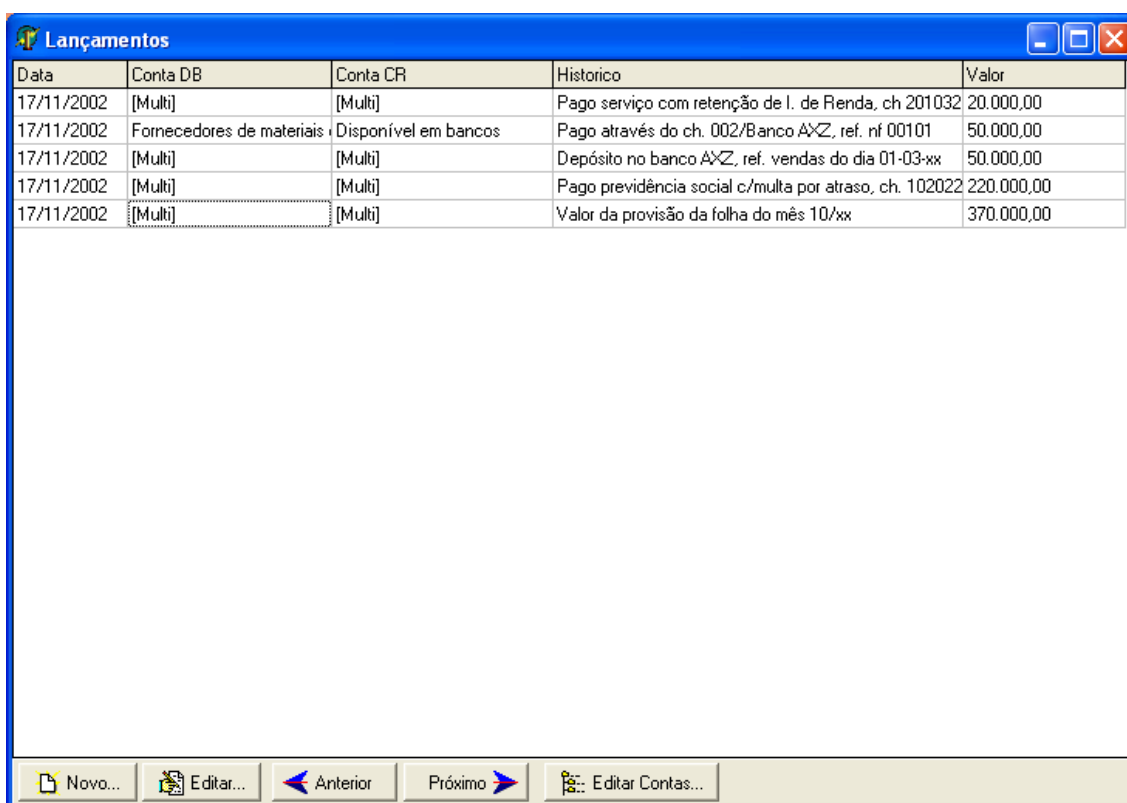
A fim de demonstrar o funcionamento da implementação elaborou-se um plano de contas, sugerido em Oliveira (1997) e simulou-se vários lançamentos os mais corriqueiros que uma empresa comum possa ter.

Não é possível demonstrar o funcionamento do servidor a não ser através de um cliente, mas vale mencionar que o servidor foi colocado a rodar em: <http://localhost/stimba/webserv.dll/>.

Por detrás de cada formulário apresentado existe uma iteração com o servidor, que é demonstrado e explicado com códigos fonte junto com cada tela.

A figura 5 mostra o formulário principal do aplicativo, que contém os lançamentos feitos pelo usuário.

Figura 5 – Tela principal do aplicativo cliente.



The screenshot shows a window titled "Lançamentos" with a table of financial entries. The table has five columns: Data, Conta DB, Conta CR, Historico, and Valor. The data is as follows:

Data	Conta DB	Conta CR	Historico	Valor
17/11/2002	[Multi]	[Multi]	Pago serviço com retenção de I. de Renda, ch 201032	20.000,00
17/11/2002	Fornecedores de materiais	Disponível em bancos	Pago através do ch. 002/Banco A×Z, ref. nf 00101	50.000,00
17/11/2002	[Multi]	[Multi]	Depósito no banco A×Z, ref. vendas do dia 01-03-xx	50.000,00
17/11/2002	[Multi]	[Multi]	Pago previdência social c/multa por atraso, ch. 102022	220.000,00
17/11/2002	[Multi]	[Multi]	Valor da provisão da folha do mês 10/xx	370.000,00

At the bottom of the window, there is a toolbar with the following buttons: Novo..., Editar..., Anterior, Próximo, and Editar Contas...

O quadro 30 demonstra a rotina de obtenção de lançamentos junto ao servidor. No código, grifado com letras vermelhas esta o exato momento em que a requisição é efetuada e armazenada em uma lista de lançamentos do tipo *TLancamentoArray*.

Quadro 30 – Código fonte da obtenção de lançamentos

```
function TfLanca.Moveto(Indice, Step : Integer): Integer;
Var ICon : IContabilidade;
    lanca : TLancamentoArray;
    Conta : TConta;
    X,Y : Integer;
    IDCD,IDCC, QTCD, QTCC : Integer;
begin
    Result := 0;
    fLastIndex := Indice;

    ICon := HRContab as IContabilidade;
    lanca := ICon.GetLancamentos( Indice, Step );

    Result := Length( Lanca );

    If Result = 0 then exit;

    grid.RowCount := Length( lanca ) + 1;

    for x := low( lanca ) to high( lanca ) do begin
        grid.Objects[0,x+1] := Pointer( lanca[x].ID );
        grid.Cells[0,x+1] := Datetimetostr( lanca[x].Data );
        grid.Cells[3,x+1] := lanca[x].Descricao;
        grid.Cells[4,x+1] := formatfloat( '###,###,##0.00',lanca[x].Valor );

        QTCD := 0; QTCC := 0;
        for Y := Low( lanca[x].Partidas ) to High( lanca[x].Partidas ) do
        begin
            If lanca[x].Partidas[y].Status = 'D' then begin
                Inc( QTCD );
                IDCD := lanca[x].Partidas[y].IDConta;
            end;

            If lanca[x].Partidas[y].Status = 'C' then begin
                Inc( QTCC );
                IDCC := lanca[x].Partidas[y].IDConta;
            end;
        end;

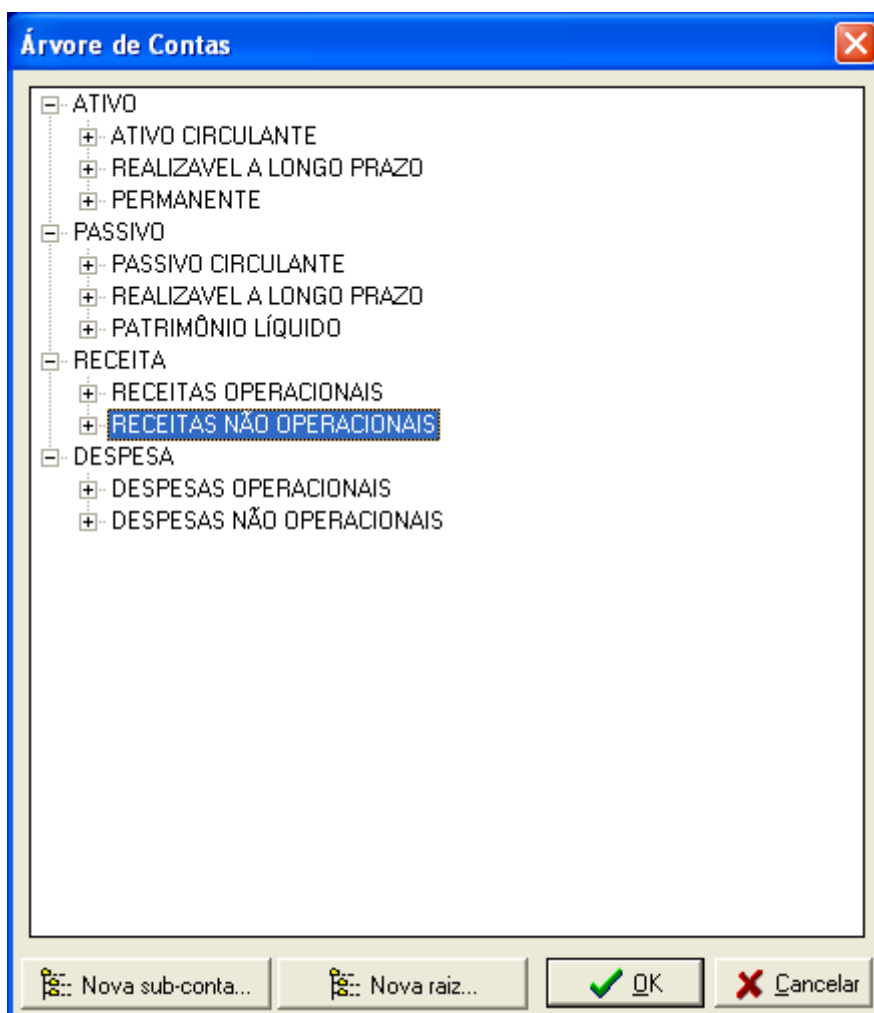
        If Qtcd = 1 then begin
            Conta := ICon.GetConta( Idcd );
            If conta <> nil then
                Grid.Cells[1,x+1] := Conta.Descricao;
        end else
            If Qtcd > 1 then Grid.Cells[1,x+1] := '[Multi]';

        If Qtcc = 1 then begin
            Conta := ICon.GetConta( Idcc );
        end;
    end;
end;
```

```
Grid.Cells[2,x+1] := Conta.Descricao;  
end else  
  If Qtcc > 1 then Grid.Cells[2,x+1] := '[Multi]';  
end;  
end;  
end;
```

Na figura 6 vê-se o diálogo que funciona tanto para a seleção quanto para o cadastro e a manutenção de contas contábeis.

Figura 6 – Tela de manutenção e seleção de contas



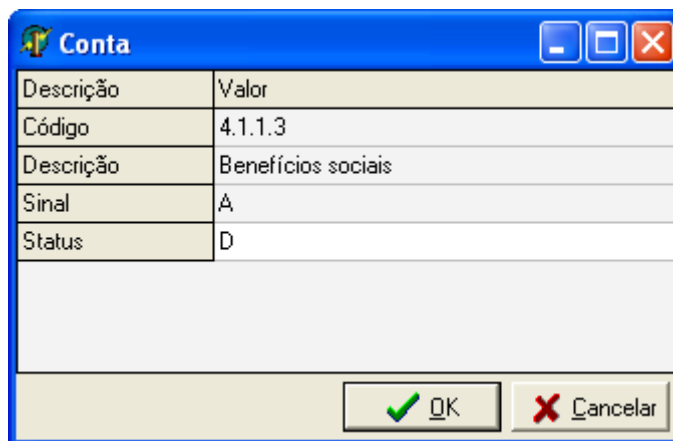
No quadro trinta é demonstrado uma das funções de requisição ao servidor. Esta função é ativada toda vez que um nó é aberto e precisa mostrar seus filhos. Na parte grifada em vermelho, o momento em que a requisição é feita.

Quadro 31 – Código fonte da obtenção de contas filhas.

```
procedure TfContas.ResetChild( lNode: TTreeNode );
Var ICon : IContabilidade;
    Conta : TContaArray;
    X : Integer;
    Node : TTreeNode;
begin
    ICon := HRContab as IContabilidade;
    Conta := ICon.GetChilds( Integer( lNode.Data ) );
    lNode.DeleteChildren;
    for x := Low( Conta ) To High( Conta ) do begin
        Node := Tree.Items.AddChild( lNode, Conta[x].Descricao );
        Node.Data := Pointer( Conta[x].ID );
    end;
end;
```

Para a adição ou manutenção de uma conta específica é usado o formulário da figura 7. Esta tela não possui eventos ligados ao servidor. Quem faz o cadastro efetivamente é a tela da figura 6.

Figura 7 – Tela de cadastro de conta



The image shows a Windows-style dialog box titled "Conta". It contains a table with the following data:

Descrição	Valor
Código	4.1.1.3
Descrição	Benefícios sociais
Sinal	A
Status	D

At the bottom of the dialog box, there are two buttons: "OK" (with a green checkmark icon) and "Cancelar" (with a red X icon).

Nas figuras 8 e 9 são mostradas duas formas do formulário de cadastro e manutenção de lançamentos.

A figura 8 mostra o cadastramento de um lançamento que possui uma única conta de débito e uma única conta de crédito.

Figura 8 – Cadastro de lançamentos - Partidas simples.

Editar Lançamento

Data: 17/11/2002

Histórico: Pago através do ch. 002/Banco AXZ, ref. nf 00101

Valor: 50000

Conta única | Várias contas

Conta débito:
Fornecedores de materiais de uso

Conta única | Várias contas

Conta crédito:
Disponível em bancos

OK Cancelar

A figura 9 mostra o cadastramento de um lançamento que pode possuir várias contas de débito e várias contas de crédito.

Figura 9 – Cadastro de Lançamentos. Partidas compostas.

Editar Lançamento

Data: 17/11/2002

Histórico: Valor da provisão da folha do mês 10/xx

Valor: 370000

Conta única Várias contas

conta	descricao	valor
<input checked="" type="checkbox"/>	Disponível em caixa na matriz	30000
<input type="checkbox"/>	Disponível em bancos na matriz	7000

Conta única Várias contas

conta	descricao	valor
<input checked="" type="checkbox"/>	Salários a pagar	345000
<input type="checkbox"/>	Consignações a pagar	25000

OK Cancelar

Ambas as telas das figuras 8 e 9 compartilham o mesmo código de edição de lançamentos do quadro 33. Neste mesmo quadro pode-se ver múltiplas iterações com o servidor, para a obtenção de contas e para o cadastro de um novo lançamento. As iterações com o servidor estão grifadas em vermelho.

Quadro 32 – Código fonte do cadastro de um novo lançamento.

```
function EditLanca( Lanca : TLancamento ): Boolean; Overload;
Var ICont : IContabilidade;
    Conta : TConta;
    IVetPartidas : TPartidaArray;
    res, x, D,C : integer;
begin
  If lanca = nil then exit;
  If not assigned(fEditLanca) then fEditLanca :=
    TfEditLanca.Create(application);
  with fEditLanca do begin
    ICont := HRContab as IContabilidade;
    edData.Date := lanca.Data;
```

```

edHistorico.Text := lanca.Descricao;
edValor.text     := floattostr( lanca.Valor );
D := 0; C:= 0;
for x := Low( lanca.Partidas ) to High( Lanca.Partidas ) do
  If Lanca.Partidas[x].Status = 'D' then
    inc( D )
  else
    Inc( C );
multidb.Close;
multidb.CreateDataSet;
multicr.Close;
multicr.CreateDataSet;

If D > 1 then begin
  pagedb.ActivePage := tabmultidb;
  for x := Low( lanca.Partidas ) to High( Lanca.Partidas ) do
    If Lanca.Partidas[x].Status = 'D' then begin
      multidb.Append;
      multidb.FieldName( 'IDCONTA' ).AsInteger :=
        Lanca.Partidas[x].IDConta;
      multidb.FieldName( 'DESCRICAO' ).AsString :=
        Lanca.Partidas[x].Descricao;
      multidb.FieldName( 'VALOR' ).AsFloat :=
        Lanca.Partidas[x].Valor;
      conta := ICont.GetConta( Lanca.Partidas[x].IDConta );
      If conta <> nil then
        multidb.FieldName( 'CONTA' ).AsString :=
          Conta.Descricao;
    end;
  end else
    for x := Low( lanca.Partidas ) to High( Lanca.Partidas ) do
      If Lanca.Partidas[x].Status = 'D' then begin
        conta := ICont.GetConta( Lanca.Partidas[x].IDConta );
        If conta <> nil then begin
          cbDebito.Tag := Conta.ID;
          cbDebito.Text := Conta.Descricao;
        end;
      end;
    end;

If C > 1 then begin
  pagecr.ActivePage := tabmulticr;
  for x := Low( lanca.Partidas ) to High( Lanca.Partidas ) do
    If Lanca.Partidas[x].Status = 'C' then begin
      multicr.Append;
      multicr.FieldName( 'IDCONTA' ).AsInteger :=
        Lanca.Partidas[x].IDConta;
      multicr.FieldName( 'DESCRICAO' ).AsString :=
        Lanca.Partidas[x].Descricao;
      multicr.FieldName( 'VALOR' ).AsFloat :=
        Lanca.Partidas[x].Valor;
      conta := ICont.GetConta( Lanca.Partidas[x].IDConta );
      If conta <> nil then
        multicr.FieldName( 'CONTA' ).AsString :=
          Conta.Descricao;
    end;
  end else begin
    for x := Low( lanca.Partidas ) to High( lanca.Partidas ) do
      If Lanca.Partidas[x].Status = 'C' then begin

```

```

        conta := ICont.GetConta( Lanca.Partidas[x].IDConta );
        If conta <> nil then begin
            cbCredito.Tag := Conta.ID;
            cbCredito.Text := Conta.Descricao;
        end;
    end;

end;

If ShowModal = mrOk then begin
    IVetPartidas := nil;
    If pagedb.ActivePage = TabMonoDb then begin
        SetLength( IVetPartidas, Length( IVetPartidas )+1 );
        IVetPartidas[High(IVetPartidas)] := TPartida.Create;
        IVetPartidas[High(IVetPartidas)].IDConta := cbDebito.Tag;
        IVetPartidas[High(IVetPartidas)].Valor :=
            strtofloat( edValor.Text );
        IVetPartidas[High(IVetPartidas)].Status := 'D';
    end else begin
        multidb.DisableControls;
        try
            multidb.First;
            while not multidb.Eof do begin
                SetLength( IVetPartidas, Length( IVetPartidas )+1 );
                IVetPartidas[High(IVetPartidas)] :=
                    TPartida.Create;
                IVetPartidas[High(IVetPartidas)].IDConta :=
                    multidb.fieldbyname( 'IDConta' ).AsInteger;
                IVetPartidas[High(IVetPartidas)].Descricao :=
                    multidb.fieldbyname( 'DESCRICA0' ).AsString;
                IVetPartidas[High(IVetPartidas)].Valor :=
                    multidb.fieldbyname( 'VALOR' ).AsFloat;
                IVetPartidas[High(IVetPartidas)].Status := 'D';
            multidb.Next;
        end;
    finally
        multidb.EnableControls;
    end;
end;

If pagecr.ActivePage = TabMonoCr then begin
    SetLength( IVetPartidas, Length( IVetPartidas )+1 );
    IVetPartidas[High(IVetPartidas)] := TPartida.Create;
    IVetPartidas[High(IVetPartidas)].IDConta := cbCredito.Tag;
    IVetPartidas[High(IVetPartidas)].Valor :=
        strtofloat( edValor.Text );
    IVetPartidas[High(IVetPartidas)].Status := 'C';
end else begin
    multicr.DisableControls;
    try
        multicr.First;
        while not multicr.Eof do begin
            SetLength( IVetPartidas, Length( IVetPartidas )+1 );
            IVetPartidas[High(IVetPartidas)] :=
                TPartida.Create;
            IVetPartidas[High(IVetPartidas)].IDConta :=
                multicr.fieldbyname( 'IDConta' ).AsInteger;
            IVetPartidas[High(IVetPartidas)].Descricao :=
                multicr.fieldbyname( 'DESCRICA0' ).AsString;

```

```

        IVetPartidas[High(IVetPartidas)].Valor      :=
            multicr.fieldbyname('VALOR').AsFloat;
        IVetPartidas[High(IVetPartidas)].Status    := 'C';
        multicr.Next;
    end;
finally
    multicr.EnableControls;
end;
end;

ICont := HRContab as IContabilidade;
ICont.NewLancamento( Datetimetostr( edData.Date ),
edHistorico.Text,
                    StrToFloat( edValor.Text ), IVetPartidas
);
    end;
end;
end;

```

3.4 RESULTADOS E DISCUSSÃO

As telas demonstradas na operacionalidade da implementação, como já ditas anteriormente, representam a telas do software cliente desenvolvido já que o servidor não pode ser demonstrado por não possuir telas.

Apesar disso foram incluídos códigos fontes que representam as rotinas no cliente que fazem as requisições de dados junto ao servidor. Mesmo assim é difícil mostrar o funcionamento do servidor isoladamente, além de que o software cliente assemelha-se muito com um software que acessa seus dados através de um gerenciador de banco de dados comum. Isso pode até confundir mas na verdade o gerenciador de banco de dados também trabalha como um servidor. Quando acessa-se dados em um banco de dados também não se vê o servidor de dados.

A XML também é outra parte invisível nos protótipos. Ela esta por trás da comunicação entre o cliente e o servidor e a única forma de averiguar as mensagens de requisição e resposta seria monitorando a comunicação entre os *sockets* de cliente e servidor.

O diferencial entre o servidor desenvolvido e um servidor de banco de dados esta na independência de comunicação e na especialização do conteúdo, permitindo que as informações possam ser tratadas independentes da plataforma usada pelo servidor, já que a forma de comunicação é padronizada, diferentemente dos servidores de banco de dados.

4 CONCLUSÕES

O padrão de comunicação desenvolvido funciona como esperado. Permite a transmissão e obtenção de dados contábeis entre um cliente e um servidor.

Sem dúvida este trabalho não é o precursor de trabalhos voltados a internet e nem sobre estruturas padrões. A estrutura desenvolvida não pretende impor-se aos desenvolvedores, mas talvez dar uma idéia de quanto uma padronização é importante para a diminuição do esforço de desenvolvimento.

A estrutura desenvolvida apresenta limitações e que deveriam ser resolvidas com mais estudos sobre a contabilidade e mais testes de usabilidade. Muitos atributos dos elementos construídos como sinal e status da conta e valor nos lançamentos, necessitam ser revistos e discutidos para saber a sua verdadeira relevância e como eles devem comportar-se em cada caso.

O servidor construído como base na estrutura desenvolvida funcionou embora apresente uma performance muito baixa, que pode se tornar um problema em situações de maior volume de comunicação. Muitos testes deveriam ser feitos para assegurar a funcionalidade e a velocidade de resposta e otimizar as funções que apresentam deficiência. Testes em situações reais e com múltiplos acessos simultâneos deveriam ser feitos para avaliar a sua funcionalidade. Outra solução também poderia ser a mudança no modelo da comunicação, abandonando os *Web Service* e o SOAP, que mostraram-se no capítulo 2, um pouco pesados e partir para uma estrutura em nível mais baixo, sem abandonar a XML.

O aplicativo cliente desenvolvido mostrou a eficiência das ferramentas Web Services em criar com facilidade rotinas que se comuniquem com o servidor. Seria interessante testar o cliente em uma situação de internet em baixa velocidade para ver se funciona adequadamente, apesar de que a maior responsabilidade quanto a performance seja atribuída ao servidor.

4.1 EXTENSÕES

Como sugestões para trabalhos futuros:

- a) A pesquisa sobre técnicas de análise que permitam retirar de um problema um modelo orientado a objetos, pois as técnicas existentes, como a UML, funcionam somente para diagramar o problema quando este já está entendido. O maior desafio nesses casos não é diagramar, mas sim entender o problema de uma forma orientada a objetos. Um exemplo pode ser o costume de se projetar o problema junto com a interface do software o que acaba tirando a estrutura do conceito de “o mundo como o mundo é”;
- b) O estudo de outras formas de representar a estrutura em formato XML, visto que o XML Schema é somente uma das possibilidades da XML;
- c) O desenvolvimento de uma extensão para o padrão, adaptando-o a alguma situação específica da contabilidade. Pode-se por exemplo adaptar o padrão para o mercado brasileiro, ou algum outro uso específico da contabilidade;

REFERÊNCIAS BIBLIOGRÁFICAS

FURGERI, Sergio. **Ensino didático da linguagem XML**. São Paulo: Érica, 2001.

FISHER, Maydene. **Introduction to Web Services**. United States, 2002. Disponível em: <<http://java.sun.com>>, Acesso em: 30 out de 2002.

OLIVEIRA, Edson. **Contabilidade informatizada: teoria e prática**. São Paulo: Atlas, 1997.

RAY, Eric T. **Aprendendo XML**. Rio de Janeiro: Campus, 2001.

W3C. **World wide web consortium**, United States, 2002. Disponível em: <<http://www.w3.org>>, Acesso em: 30 out de 2002.

Anexo 1 – A estrutura padrão de comunicação em XML

```

<?xml version="1.0"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  name="IContabilidadeservice"
  targetNamespace="http://tempuri.org/"
  xmlns:tns="http://tempuri.org/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:ns1="urn:IAccountIntf">
  <types>
    <xs:schema targetNamespace="urn:IAccountIntf"
      xmlns="urn:IAccountIntf">
      <xs:complexType name="TConta">
        <xs:sequence>
          <xs:element name="ID" type="xs:long"/>
          <xs:element name="IDAnt" type="xs:long"/>
          <xs:element name="Codigo" type="xs:string"/>
          <xs:element name="Descricao" type="xs:string"/>
          <xs:element name="Sinal" type="xs:string"/>
          <xs:element name="Status" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="TContaArray">
        <xs:complexContent>
          <xs:restriction base="soapenc:Array">
            <xs:sequence/>
            <xs:attribute ref="soapenc:arrayType"
nl:arrayType="ns1:TConta[]"
xmlns:ns1="http://schemas.xmlsoap.org/wsdl/" />
          </xs:restriction>
        </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="TPartida">
        <xs:sequence>
          <xs:element name="ID" type="xs:long"/>
          <xs:element name="Conta" type="ns1:TConta"/>
          <xs:element name="IDConta" type="xs:long"/>
          <xs:element name="IDLancamento" type="xs:long"/>
          <xs:element name="Descricao" type="xs:string"/>
          <xs:element name="Valor" type="xs:double"/>
          <xs:element name="Status" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="TLancamento">
        <xs:sequence>
          <xs:element name="ID" type="xs:long"/>
          <xs:element name="Data" type="xs:TDateTime"/>
          <xs:element name="Descricao" type="xs:string"/>
          <xs:element name="Valor" type="xs:double"/>
          <xs:element name="Partidas" type="ns1:TPartidaArray"/>
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="TPartidaArray">

```

```

        <xs:complexContent>
          <xs:restriction base="soapenc:Array">
            <xs:sequence/>
            <xs:attribute ref="soapenc:arrayType"
n1:arrayType="nsl:TPartida[]"
xmlns:n1="http://schemas.xmlsoap.org/wsdl/"/>
          </xs:restriction>
        </xs:complexContent>
      </xs:complexType>
      <xs:complexType name="TLancamentoArray">
        <xs:complexContent>
          <xs:restriction base="soapenc:Array">
            <xs:sequence/>
            <xs:attribute ref="soapenc:arrayType"
n1:arrayType="nsl:TLancamento[]"
xmlns:n1="http://schemas.xmlsoap.org/wsdl/"/>
          </xs:restriction>
        </xs:complexContent>
      </xs:complexType>
    </xs:schema>
  </types>
  <message name="GetContaRequest">
    <part name="ID" type="xs:long"/>
  </message>
  <message name="GetContaResponse">
    <part name="return" type="nsl:TConta"/>
  </message>
  <message name="GetChildsRequest">
    <part name="ID" type="xs:long"/>
  </message>
  <message name="GetChildsResponse">
    <part name="return" type="nsl:TContaArray"/>
  </message>
  <message name="NewContaRequest">
    <part name="IDAnt" type="xs:long"/>
    <part name="Codigo" type="xs:string"/>
    <part name="Titulo" type="xs:string"/>
    <part name="Sinal" type="xs:string"/>
    <part name="Status" type="xs:string"/>
    <part name="SaldoIni" type="xs:double"/>
  </message>
  <message name="NewContaResponse">
    <part name="return" type="xs:int"/>
  </message>
  <message name="GetPartidaRequest">
    <part name="ID" type="xs:long"/>
  </message>
  <message name="GetPartidaResponse">
    <part name="return" type="nsl:TPartida"/>
  </message>
  <message name="GetLancamentoRequest">
    <part name="ID" type="xs:long"/>
  </message>
  <message name="GetLancamentoResponse">
    <part name="return" type="nsl:TLancamento"/>
  </message>
  <message name="GetLancamentosRequest">
    <part name="Inicio" type="xs:int"/>
    <part name="MaxResult" type="xs:int"/>

```

```

</message>
<message name="GetLancamentosResponse">
  <part name="return" type="nsl:T LancamentoArray"/>
</message>
<message name="NewLancamentoRequest">
  <part name="Data" type="xs:string"/>
  <part name="Descricao" type="xs:string"/>
  <part name="Valor" type="xs:double"/>
  <part name="Partidas" type="nsl:TPartidaArray"/>
</message>
<message name="NewLancamentoResponse"/>
<portType name="IContabilidade">
  <operation name="GetConta">
    <input message="tns:GetContaRequest"/>
    <output message="tns:GetContaResponse"/>
  </operation>
  <operation name="GetChilds">
    <input message="tns:GetChildsRequest"/>
    <output message="tns:GetChildsResponse"/>
  </operation>
  <operation name="NewConta">
    <input message="tns:NewContaRequest"/>
    <output message="tns:NewContaResponse"/>
  </operation>
  <operation name="GetPartida">
    <input message="tns:GetPartidaRequest"/>
    <output message="tns:GetPartidaResponse"/>
  </operation>
  <operation name="GetLancamento">
    <input message="tns:GetLancamentoRequest"/>
    <output message="tns:GetLancamentoResponse"/>
  </operation>
  <operation name="GetLancamentos">
    <input message="tns:GetLancamentosRequest"/>
    <output message="tns:GetLancamentosResponse"/>
  </operation>
  <operation name="NewLancamento">
    <input message="tns:NewLancamentoRequest"/>
    <output message="tns:NewLancamentoResponse"/>
  </operation>
</portType>
<binding name="IContabilidadebinding" type="tns:IContabilidade">
  <soap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetConta">
    <soap:operation soapAction="urn:IAccountIntf-
IContabilidade#GetConta" style="rpc"/>
    <input>
      <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:IAccountIntf-IContabilidade"/>
    </input>
    <output>
      <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:IAccountIntf-IContabilidade"/>
    </output>
  </operation>
  <operation name="GetChilds">

```

```

        <soap:operation soapAction="urn:IAccountIntf-
IContabilidade#GetChilds" style="rpc"/>
        <input>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:IAccountIntf-IContabilidade"/>
        </input>
        <output>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:IAccountIntf-IContabilidade"/>
        </output>
    </operation>
    <operation name="NewConta">
        <soap:operation soapAction="urn:IAccountIntf-
IContabilidade#NewConta" style="rpc"/>
        <input>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:IAccountIntf-IContabilidade"/>
        </input>
        <output>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:IAccountIntf-IContabilidade"/>
        </output>
    </operation>
    <operation name="GetPartida">
        <soap:operation soapAction="urn:IAccountIntf-
IContabilidade#GetPartida" style="rpc"/>
        <input>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:IAccountIntf-IContabilidade"/>
        </input>
        <output>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:IAccountIntf-IContabilidade"/>
        </output>
    </operation>
    <operation name="GetLancamento">
        <soap:operation soapAction="urn:IAccountIntf-
IContabilidade#GetLancamento" style="rpc"/>
        <input>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:IAccountIntf-IContabilidade"/>
        </input>
        <output>
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:IAccountIntf-IContabilidade"/>
        </output>
    </operation>
    <operation name="GetLancamentos">
        <soap:operation soapAction="urn:IAccountIntf-
IContabilidade#GetLancamentos" style="rpc"/>
        <input>

```

```

        <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:IAccountIntf-IContabilidade"/>
    </input>
    <output>
        <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:IAccountIntf-IContabilidade"/>
    </output>
</operation>
<operation name="NewLancamento">
    <soap:operation soapAction="urn:IAccountIntf-
IContabilidade#NewLancamento" style="rpc"/>
    <input>
        <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:IAccountIntf-IContabilidade"/>
    </input>
    <output>
        <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:IAccountIntf-IContabilidade"/>
    </output>
</operation>
</binding>
<service name="IContabilidadeservice">
    <port name="IContabilidadePort"
binding="tns:IContabilidadebinding">
        <soap:address
location="http://localhost/stimba/webserv.dll/soap/IContabilidade"/>
    </port>
</service>
</definitions>

```

Anexos 2 – Código fonte da definição dos objetos no servidor

```

unit IAccountIntf;

interface

uses InvokeRegistry;

type
  TConta = Class( TRemotable )
  private
    fID      : Int64;
    fIDAnt   : Int64;
    fCodigo  : WideString;
    fDescricao: WideString;
    fSinal   : Char;
    fStatus  : Char;
  published
    property ID      : Int64      Read fID      Write fID;
    property IDAnt   : Int64      Read fIDAnt   Write fIDAnt;
    property Codigo  : WideString Read fCodigo  Write fCodigo;
    property Descricao: WideString Read fTitulo  Write fTitulo;
    property Sinal   : Char       Read fSinal   Write fSinal;
    property Status  : Char       Read fStatus  Write fStatus;
  end;

  TContaArray = Array of TConta;

  TPartida = Class( TRemotable )
  private
    fID      : Int64;
    fIDConta: Int64;
    fIDLanca: Int64;
    fDesc    : WideString;
    fValor   : Double;
    fStatus  : WideString;
  published
    property ID      : Int64 Read fID      Write fID;
    property IDConta : Int64 Read fIDConta Write fIDConta;
    property IDLancamento : Int64 Read fIDLanca Write fIDLanca;
    property Descricao : WideString Read fDesc Write fDesc;
    property Valor     : Double Read fValor Write fValor;
    property Status    : WideString Read fStatus Write fStatus;
  end;

  TPartidaArray = Array of TPartida;

  TLancamento = Class( TRemotable )
  private
    fID      : Int64;
    fData    : TDateTime;
    fDesc    : WideString;
    fValor   : Double;
    fPartidas : TPartidaArray;
  public
    procedure AddPartida( Value : TPartida );

```



```

published
  property ID      : Int64      Read fID      Write fID;
  property Data    : TDateTime  Read fData   Write fData;
  property Descricao: WideString Read fDesc   Write fDesc;
  property Valor   : Double     Read fValor   Write fValor;
  property Partidas : TPartidaArray Read fPartidas Write fPartidas;
end;

TLancamentoArray = Array of TLancamento;

IContabilidade = interface(IInvokable)
['{1AF5F692-49F7-492C-A27F-65C13CF00324}']
  function GetConta( ID : Int64 ): TConta; stdcall;
  function GetChilds( ID : Int64 ): TContaArray; stdcall;
  function NewConta( IDAnt : Int64; Codigo, Titulo : WideString;
                    Sinal, Status : Char; SaldoIni : Double ):
                    Integer; stdcall;

  function GetPartida( ID : Int64 ): TPartida; stdcall;

  function GetLancamento( ID : Int64 ): TLancamento; stdcall;
  function GetLancamentos( Inicio : Integer; MaxResult : Integer
                          ): TLancamentoArray; stdcall;
  procedure NewLancamento( Data, Descricao : String; Valor :
                          Double; Partidas : TPartidaArray );
                          stdcall;
end;

implementation

{ TLancamento }

procedure TLancamento.AddPartida(Value: TPartida);
begin
  SetLength( fPartidas, Length( fPartidas ) + 1 );
  fPartidas[High( fPartidas )] := Value;
end;

initialization
  InvRegistry.RegisterInterface(TypeInfo(IContabilidade));
  RemTypeRegistry.RegisterXSClass(TConta);
  RemTypeRegistry.RegisterXSClass(TPartida);
  RemTypeRegistry.RegisterXSClass(TLancamento);
  RemTypeRegistry.RegisterXSInfo(TypeInfo(TContaArray));
  RemTypeRegistry.RegisterXSInfo(TypeInfo(TPartidaArray));
  RemTypeRegistry.RegisterXSInfo(TypeInfo(TLancamentoArray));
end.

```

Anexo 3 – Código fonte da implementação dos objetos

```

unit IAccountImpl;

interface

uses InvokeRegistry, IAccountIntf, sysutils, windows;

type
  TContabilidade = class(TInvokableClass, IContabilidade )
  public
    function GetConta( ID : Int64 ): TConta; stdcall;
    function GetChilds( ID : Int64 ): TContaArray; stdcall;
    function NewConta( IDAnt : Int64; Codigo, Descricao :
      WideString; Sinal, Status : Char ): Integer;
      stdcall;
    function GetPartida( ID : Int64 ): TPartida; stdcall;
    function GetLancamento( ID : Int64 ): TLancamento; stdcall;
    function GetLancamentos( Inicio : Integer; MaxResult :
      Integer ): TLancamentoArray; stdcall;
    procedure NewLancamento( Data, Descricao : String; Valor :
      Double; Partidas : TPartidaArray );
      stdcall;

  end;

implementation

uses udbserv;

{ TContabilidade }

function TContabilidade.GetChilds( ID: Int64): TContaArray;
var P : Integer;
    Si, St : String;
begin
  with dbserv do begin
    Tabaux.close;
    If ID = 0 then
      Tabaux.CommandText := 'SELECT * FROM CONTAS WHERE ID_CONTAS'+
        'IS NULL'
    else
      Tabaux.CommandText :=
        'SELECT * FROM CONTAS WHERE ID_CONTAS = '+ Inttostr( ID );
    Tabaux.Open;
    while not tabaux.eof do begin
      SetLength( result, Length( result ) + 1 );
      P := high( result );
      result[P] := TConta.Create;
      result[P].ID := TabAux.Fieldbyname('ID').AsInteger;
      result[P].IDAnt :=
        TabAux.Fieldbyname('ID_CONTAS').AsInteger;
      result[P].Codigo := TabAux.Fieldbyname('CODIGO').AsString;
      result[P].Descricao := TabAux.Fieldbyname('TITULO').AsString;
    end;
  end;
end;

```

```

        Si := TabAux.Fieldbyname('SINAL').AsString;
        St := TabAux.Fieldbyname('STATUS').AsString;
        If Length( Si ) > 0 then result[P].Sinal := Si[1];
        If Length( St ) > 0 then result[P].Status := St[1];
        tabaux.next;
    end;
end;
end;

function TContabilidade.GetConta( ID: Int64): TConta;
begin
    Result := nil;
    with dbserv do begin
        TabAux.Close;
        TabAux.CommandText := 'SELECT * FROM CONTAS WHERE ID =
'+Inttostr( ID );
        TabAux.Open;
        If not TabAux.FieldByName('ID').IsNull then begin
            result := TConta.Create;
            result.ID := TabAux.Fieldbyname('ID').AsInteger;
            result.IDAnt :=
TabAux.Fieldbyname('ID_CONTAS').AsInteger;
            result.Descricao := TabAux.Fieldbyname('TITULO').AsString;
        end;
    end;
end;

function TContabilidade.GetLancamento( ID: Int64): TLancamento;
begin
    result := TLancamento.Create;
    result.ID := 10001;
    result.Descricao := 'Lancamento de Teste';
    result.Valor := 100.02;
end;

function TContabilidade.GetPartida( ID: Int64): TPartida;
begin
    result := TPartida.Create;
    result.ID := 100002;
    result.Descricao := 'Partida de Teste';
    result.Valor := 100.50;
end;

function TContabilidade.NewConta( IDAnt: Int64; Codigo, Titulo:
WideString;
        Sinal, Status: Char; SaldoIni: Double): Integer;
begin
    with dbserv do begin
        Result := MaxId( 'CONTAS' ) + 1;
        tabaux.Append;
        tabaux.FieldByName('ID').AsInteger := Result;
        If idAnt > 0 then tabaux.FieldByName('ID_CONTAS').AsInteger :=
IDAnt;
        tabaux.FieldByName('TITULO').AsString := Titulo;
        tabaux.FieldByName('SINAL').AsString := Sinal;
        tabaux.FieldByName('STATUS').AsString := Status;
        tabaux.FieldByName('SALDOINICIAL').AsFloat := SaldoIni;
        tabaux.Post;
        tabaux.ApplyUpdates(0);
    end;
end;

```

```

    end;
end;

function TContabilidade.GetLancamentos(Inicio, MaxResult: Integer):
    TLancamentoArray;
var count : integer;
    part : TPartida;
begin
    with dbserv do begin
        tablanca.Close;
        tablanca.CommandText := 'SELECT * FROM LANCAMENTOS ORDER BY '+'
                                'DATA DESC';

        tablanca.Open;
        count := 0;
        tablanca.first;
        while (not tablanca.Eof) and (Count < Inicio) do begin
            tablanca.Next;
            inc( count );
        end;
        while not tablanca.Eof do begin
            If length( Result ) >= MaxResult then break;
            tabpart.Close;
            tabpart.CommandText := 'SELECT * FROM PARTIDAS WHERE '+'
                                    'ID_LANCAMENTOS = '+'
                                    tablanca.fieldbyname('ID').AsString;

            tabpart.Open;

            SetLength( Result, Length( Result ) + 1 );
            Result[ High( Result ) ] := TLancamento.Create;
            Result[ High( Result ) ].ID :=
                tablanca.fieldbyname('ID').AsInteger;
            Result[ High( Result ) ].Data :=
                tablanca.fieldbyname('DATA').AsDatetime;
            Result[ High( Result ) ].Descricao :=
                tablanca.fieldbyname('HISTORICO').AsString;
            Result[ High( Result ) ].Valor :=
                tablanca.fieldbyname('VALOR').AsFloat;

            while not tabpart.Eof do begin
                part := TPartida.Create;
                part.ID :=
                    tabpart.fieldbyname('ID').AsInteger;
                part.IDConta :=
                    tabpart.fieldbyname('ID_CONTAS').AsInteger;
                part.IDLancamento :=
                    tabpart.fieldbyname('ID_LANCAMENTOS').AsInteger;
                part.Descricao :=
                    tabpart.fieldbyname('DESCRICAO').AsString;
                part.Valor := tabpart.fieldbyname('VALOR').AsFloat;
                part.Status :=
                    tabpart.fieldbyname('STATUS').AsString;

                Result[ High( Result ) ].AddPartida( part );
                tabpart.Next;
            end;
            tablanca.next;
        end;
    end;
end;
end;
end;

```

```

procedure TContabilidade.NewLancamento(Data, Descricao: String; Valor:
Double; Partidas: TPartidaArray);
var idlan, idpart, x : integer;
begin
  with dbserv do begin
    idlan := MaxID( 'LANCAMENTOS' ) + 1;
    tablanca.append;
    tablanca.FieldName( 'ID' ).AsInteger      := idlan;
    tablanca.FieldName( 'DATA' ).AsDateTime  :=
      StrToDatetime( Data );
    tablanca.FieldName( 'HISTORICO' ).AsString := Descricao;
    tablanca.FieldName( 'VALOR' ).AsFloat    := Valor;
    tablanca.Post;
    tablanca.ApplyUpdates(0);

    idpart := MaxID( 'PARTIDAS' );
    for x := low( Partidas ) to high( Partidas ) do begin
      inc( idpart );
      tabpart.append;
      tabpart.FieldName( 'ID' ).AsInteger := idpart;
      tabpart.FieldName( 'ID_CONTAS' ).AsInteger :=
        Partidas[x].IDConta;
      tabpart.FieldName( 'ID_LANCAMENTOS' ).AsInteger := idlan;
      tabpart.FieldName( 'DESCRICAO' ).AsString :=
        Partidas[x].Descricao;
      tabpart.FieldName( 'VALOR' ).AsFloat :=
        Partidas[x].Valor;
      tabpart.FieldName( 'STATUS' ).AsString :=
        Partidas[x].Status;
      tabpart.Post;
    end;
    tabpart.ApplyUpdates(0);
  end;
end;

initialization
  InvRegistry.RegisterInvokableClass(TContabilidade);
end.

```

Anexo 4 – Código fonte do modulo de acesso a dados

```

unit uDBServ;

interface

uses
  SysUtils, Classes, DBXpress, FMTBcd, DB, SqlExpr, Provider,
  DBClient, DBLocal, DBLocalS;

type
  Tdm = class(TDataModule)
    con: TSQLConnection;
    tabmaxid: TSQLDataSet;
    tabaux: TSQLClientDataSet;
    tabpart: TSQLClientDataSet;
    tablanca: TSQLClientDataSet;
  private
    { Private declarations }
  public
    function Maxid( TableName : String ): Integer;
  end;

function dbserv : Tdm;

implementation

{$R *.dfm}

{ Tdm }

var dm: Tdm;

function dbserv : Tdm;
begin
  If not assigned( dm ) then dm := Tdm.Create( nil );
  result := dm;
end;

function Tdm.Maxid(TableName: String): Integer;
begin
  TabMaxid.Close;
  TabMaxid.CommandText := 'SELECT MAX(ID) AS MAXID FROM '+TableName;
  TabMaxid.Open;
  Result := TabMaxid.fieldbyname('MAXID').AsInteger;
end;

initialization
finalization
  If assigned( dm ) then dm.Free;
end.

```

Anexo 5 – Código fonte do formulário de acesso a dados

```

object dm: Tdm
  OldCreateOrder = False
  Left = 557
  Top = 264
  Height = 196
  Width = 221
  object con: TSQLConnection
    Connected = True
    ConnectionName = 'TCC'
    DriverName = 'Interbase'
    GetDriverFunc = 'getSQLDriverINTERBASE'
    LibraryName = 'dbexpint.dll'
    LoginPrompt = False
    Params.Strings = (
      'DriverName=Interbase'
      'BlobSize=-1'
      'CommitRetain=False'
      'Database=127.0.0.1:c:\TCC.gdb'
      'ErrorResourceFile='
      'LocaleCode=0000'
      'Password=masterkey'
      'RoleName=RoleName'
      'ServerCharSet='
      'SQLDialect=1'
      'Interbase TransIsolation=ReadCommitted'
      'User_Name=SYSDBA'
      'WaitOnLocks=True')
    VendorLib = 'GDS32.DLL'
    Left = 24
    Top = 16
  end
  object tabmaxid: TSQLDataSet
    SQLConnection = con
    Params = <>
    Left = 72
    Top = 16
  end
  object tabaux: TSQLClientDataSet
    Active = True
    CommandText = 'select * from contas'
    Aggregates = <>
    Options = [poAllowCommandText]
    ObjectView = True
    Params = <>
    DBConnection = con
    Left = 24
    Top = 80
  end
  object tabpart: TSQLClientDataSet
    Active = True
    CommandText = 'select * from PARTIDAS'
    Aggregates = <>

```

```
Options = [poAllowCommandText]
ObjectView = True
Params = <>
DBConnection = con
Left = 72
Top = 80
end
object tablanca: TSQLClientDataSet
Active = True
CommandText = 'select * from LANCAMENTOS'
Aggregates = <>
Options = [poAllowCommandText]
ObjectView = True
Params = <>
DBConnection = con
Left = 120
Top = 80
end
end
```


Anexo 6 – Código fonte do modulo web

```
unit uServii;

interface

uses
  SysUtils, Classes, HTTPApp, DBXpress, FMTBcd, DB, SqlExpr, WSDLPub,
  SOAPPasInv, SOAPHTTTPasInv, SOAPHTTTPDisp, WebBrokerSOAP,
  IAccountImpl, IAccountIntf;

type
  Twebmod = class(TWebModule)
    HTTPSoapDispatcher1: THTTPSoapDispatcher;
    HTTPSoapPascalInvoker1: THTTPSoapPascalInvoker;
    WSDLHTMLPublish1: TWSDLHTMLPublish;
    procedure WebModule2DefaultHandlerAction(Sender: TObject; Request:
      TWebRequest; Response: TWebResponse; var Handled: Boolean);
  end;

var webmod: Twebmod;

implementation

uses WebReq;

{$R *.DFM}

procedure Twebmod.WebModule2DefaultHandlerAction(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  WSDLHTMLPublish1.ServiceInfo(Sender, Request, Response, Handled);
end;

initialization
  WebRequestHandler.WebModuleClass := Twebmod;
end.
```

Anexo 7 – Código fonte do formulário do módulo web

```
object webmod: Twebmod
  OldCreateOrder = False
  Actions = <
    item
      Default = True
      Name = 'DefaultHandler'
      PathInfo = '/'
      OnAction = WebModule2DefaultHandlerAction
    end>
  Left = 416
  Top = 365
  Height = 230
  Width = 415
  object HTTPSoapDispatcher1: THTTPSoapDispatcher
    Dispatcher = HTTPSoapPascalInvoker1
    WebDispatch.PathInfo = 'soap*'
    Left = 60
    Top = 11
  end
  object HTTPSoapPascalInvoker1: THTTPSoapPascalInvoker
    Converter.Options = [soSendMultiRefObj, soTryAllSchema]
    Left = 60
    Top = 67
  end
  object WSDLHTMLPublish1: TWSDLHTMLPublish
    WebDispatch.MethodType = mtAny
    WebDispatch.PathInfo = 'wsdl*'
    AdminEnabled = False
    TargetNamespace = 'http://tempuri.org/'
    PublishOptions = []
    Left = 60
    Top = 123
  end
end
```

Anexo 8 – Código fonte dos objetos usados pelo cliente para acesso ao servidor

```

// ***** //
// The types declared in this file were generated from data read from the
// WSDL File described below:
// WSDL      : http://localhost/stimba/webserv.dll/wsdl/IContabilidade
// Version   : 1.0
// (7/11/2002 16:10:21 - $Revision: 1.9.1.0.1.0.1.9 $)
// ***** //

unit IContabilidade;

interface

uses InvokeRegistry, Types, XSBuiltIns;

type

//***** //
// The following types, referred to in the WSDL document are not being represented
// in this file. They are either aliases[@] of other types represented or were referred
// to but never[!] declared in the document. The types from the latter category
// typically map to predefined/known XML or Borland types; however, they could also
// indicate incorrect WSDL documents that failed to declare or import a schema type.
//***** //
// !:long      - "http://www.w3.org/2001/XMLSchema"
// !:string    - "http://www.w3.org/2001/XMLSchema"
// !:double    - "http://www.w3.org/2001/XMLSchema"
// !:TDateTime - "http://www.w3.org/2001/XMLSchema"
// !:int       - "http://www.w3.org/2001/XMLSchema"

    TConta      = class;           { "urn:IAccountIntf" }
    TPartida    = class;           { "urn:IAccountIntf" }
    TLancamento = class;           { "urn:IAccountIntf" }

//***** //
// Namespace : urn:IAccountIntf
//***** //
    TConta = class(TRemotable)
    private
        FID: Int64;
        FIDAnt: Int64;
        FCodigo: String;
        FTitulo: String;
        FSinal: String;
        FStatus: String;
        FSaldoIni: Double;
    published
        property ID: Int64 read FID write FID;
        property IDAnt: Int64 read FIDAnt write FIDAnt;
        property Codigo: String read FCodigo write FCodigo;
        property Descricao: String read FTitulo write FTitulo;
        property Sinal: String read FSinal write FSinal;
        property Status: String read FStatus write FStatus;

```

```

end;

TContaArray = array of TConta;           { "urn:IAccountIntf" }

//***** //
// Namespace : urn:IAccountIntf
// ***** //
TPartida = class(TRemotable)
private
  FID: Int64;
  FConta: TConta;
  FIDConta: Int64;
  FIDLancamento: Int64;
  FDescricao: String;
  FValor: Double;
  FStatus: String;
public
  destructor Destroy; override;
published
  property ID: Int64 read FID write FID;
  property Conta: TConta read FConta write FConta;
  property IDConta: Int64 read FIDConta write FIDConta;
  property IDLancamento: Int64 read FIDLancamento write
FIDLancamento;
  property Descricao: String read FDescricao write FDescricao;
  property Valor: Double read FValor write FValor;
  property Status: String read FStatus write FStatus;
end;

TPartidaArray = array of TPartida;       { "urn:IAccountIntf" }

// ***** //
// Namespace : urn:IAccountIntf
// ***** //
TLancamento = class(TRemotable)
private
  FID: Int64;
  FData: TDateTime;
  FDescricao: String;
  FValor: Double;
  FPartidas: TPartidaArray;
public
  destructor Destroy; override;
published
  property ID: Int64 read FID write FID;
  property Data: TDateTime read FData write FData;
  property Descricao: String read FDescricao write FDescricao;
  property Valor: Double read FValor write FValor;
  property Partidas: TPartidaArray read FPartidas write FPartidas;
end;

TLancamentoArray = array of TLancamento; { "urn:IAccountIntf" }

// ***** //
// Namespace : urn:IAccountIntf-IContabilidade
// soapAction: urn:IAccountIntf-IContabilidade#operationName%
// transport : http://schemas.xmlsoap.org/soap/http

```

```

// style      : rpc
// binding    : IContabilidadebinding
// service    : IContabilidadeservice
// port       : IContabilidadePort
// URL        : http://localhost/stimba/webserv.dll/soap/IContabilidade
// ***** //
IContabilidade = interface(IInvokable)
['{C60D47F8-6635-F389-13F0-F9F0E367BADA}']
function GetConta(const ID: Int64): TConta; stdcall;
function GetChilds(const ID: Int64): TContaArray; stdcall;
function NewConta(const IDAnt: Int64; const Codigo: String;
const Descricao: String; const Sinal: String;
const Status: String ): Integer; stdcall;
function GetPartida(const ID: Int64): TPartida; stdcall;
function GetLancamento(const ID: Int64): TLancamento; stdcall;
function GetLancamentos(const Inicio: Integer;
const MaxResult: Integer): TLancamentoArray; stdcall;
procedure NewLancamento(const Data: String; const Descricao:
String; const Valor: Double; const Partidas:
TPartidaArray); stdcall;

end;

function GetIContabilidade(UseWSDL: Boolean=System.False; Addr:
string=''): IContabilidade;

implementation
uses SOAPHTTPClient;

function GetIContabilidade(UseWSDL: Boolean; Addr: string):
IContabilidade;
const
defWSDL = 'http://localhost/stimba/webserv.dll/wsd/IContabilidade';
defURL  = 'http://localhost/stimba/webserv.dll/soap/IContabilidade';
defSvc  = 'IContabilidadeservice';
defPrt  = 'IContabilidadePort';
var
RIO: THTTTPRIO;
begin
Result := nil;
if (Addr = '') then
begin
if UseWSDL then
Addr := defWSDL
else
Addr := defURL;
end;
RIO := THTTTPRIO.Create(nil);
try
if UseWSDL then
begin
RIO.WSDLLocation := Addr;
RIO.Service := defSvc;
RIO.Port := defPrt;
end else
RIO.URL := Addr;
Result := (RIO as IContabilidade);
finally
if Result = nil then

```

```
        RIO.Free;
    end;
end;

destructor TPartida.Destroy;
begin
    if Assigned(FConta) then
        FConta.Free;
    inherited Destroy;
end;

destructor TLancamento.Destroy;
var
    I: Integer;
begin
    for I := 0 to Length(FPartidas)-1 do
        if Assigned(FPartidas[I]) then
            FPartidas[I].Free;
        SetLength(FPartidas, 0);
    inherited Destroy;
end;

initialization
    InvRegistry.RegisterInterface(TypeInfo(IContabilidade),
'urn:IAccountIntf-IContabilidade', '');
    InvRegistry.RegisterDefaultSOAPAction(TypeInfo(IContabilidade),
'urn:IAccountIntf-IContabilidade#%operationName%');
    RemClassRegistry.RegisterXSClass(TConta, 'urn:IAccountIntf',
'TConta');
    RemClassRegistry.RegisterXSInfo(TypeInfo(TContaArray),
'urn:IAccountIntf', 'TContaArray');
    RemClassRegistry.RegisterXSClass(TPartida, 'urn:IAccountIntf',
'TPartida');
    RemClassRegistry.RegisterXSInfo(TypeInfo(TPartidaArray),
'urn:IAccountIntf', 'TPartidaArray');
    RemClassRegistry.RegisterXSClass(TLancamento, 'urn:IAccountIntf',
'TLancamento');
    RemClassRegistry.RegisterXSInfo(TypeInfo(TLancamentoArray),
'urn:IAccountIntf', 'TLancamentoArray');
end.
```

Anexo 9 – Código fonte do formulário de manutenção de contas

```

unit ufrmContas;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, Rio, SOAPHTTPClient, ComCtrls, StdCtrls, Buttons,
  IContabilidade, ExtCtrls, u_EditConta;

type
  TfContas = class(TForm)
    Tree: TTreeView;
    HRContab: THTTTPRIO;
    pnButtons: TPanel;
    btOK: TBitBtn;
    btCancel: TBitBtn;
    SpeedButton1: TSpeedButton;
    SpeedButton2: TSpeedButton;
    procedure FormCreate(Sender: TObject);
    procedure TreeExpanding(Sender: TObject; Node: TTreeNode;
      var AllowExpansion: Boolean);
    procedure SpeedButton1Click(Sender: TObject);
    procedure SpeedButton2Click(Sender: TObject);
  private
    procedure ResetChild( lNode : TTreeNode );
  public
    { Public declarations }
  end;

function GetConta( var lTitulo : String ) : Integer;
Procedure ShowContas;

var fContas: TfContas;

implementation

{$R *.dfm}

function GetConta( var lTitulo : String ) : Integer;
begin
  Result := -1;
  If not Assigned( fContas ) then fContas :=
    TfContas.Create( Application );

  with fContas do begin
    If ShowModal = mrOk then
      If Tree.Selected <> nil then begin
        Result := Integer( Tree.Selected.Data );
        lTitulo := Tree.Selected.Text;
      end;
    end;
  end;
end;

```

```

Procedure ShowContas;
begin
  If not Assigned( fContas ) then fContas :=
    TfContas.Create( Application );
  with fContas do begin
    btOk.Visible := False;
    btCancel.Visible := False;
    ShowModal;
    btOk.Visible := True;
    btCancel.Visible := True;
  end;
end;

procedure TfContas.ResetChild( lNode: TTreeNode );
Var ICon : IContabilidade;
    Conta : TContaArray;
    X : Integer;
    Node : TTreeNode;
begin
  ICon := HRContab as IContabilidade;
  Conta := ICon.GetChilds( Integer( lNode.Data ) );

  lNode.DeleteChildren;
  for x := Low( Conta ) To High( Conta ) do begin
    Node := Tree.Items.AddChild( lNode, Conta[x].Titulo );
    Node.Data := Pointer( Conta[x].ID );
  end;
end;

procedure TfContas.FormCreate(Sender: TObject);
Var ICon : IContabilidade;
    Conta : TContaArray;
    X : Integer;
    Node : TTreeNode;
begin
  ICon := HRContab as IContabilidade;
  Conta := ICon.GetChilds( 0 );

  for x := Low( Conta ) To High( Conta ) do begin
    Node := Tree.Items.Add( nil, Conta[x].Titulo );
    Node.Data := Pointer( Conta[x].ID );
    ResetChild( Node );
  end;
end;

procedure TfContas.TreeExpanding(Sender: TObject; Node: TTreeNode; var
AllowExpansion: Boolean);
var x : Integer;
begin
  Tree.Items.BeginUpdate;
  try
    for x := 0 to Node.Count - 1 do ResetChild( Node.Item[x] );
  finally
    Tree.Items.EndUpdate;
  end;
end;

procedure TfContas.SpeedButton1Click(Sender: TObject);
Var ICon : IContabilidade;

```



```
Cod, des, Sin, Sta : String;
Sdi : Double;
IdAux : Integer;
Node : TTreeNode;
begin
  If Tree.Selected <> nil then
    If EditConta( cod, des, sin, sta ) then begin
      ICon := HRContab as IContabilidade;
      IDAux := ICon.NewConta( Integer( Tree.Selected.Data ), Cod,
                             des, Sin, Sta );

      If IDAux >= 0 then begin
        Node := Tree.Items.AddChild( Tree.Selected, des );
        Node.Data := Pointer( IDAux );
      end;
    end;
  end;
end;

procedure TfContas.SpeedButton2Click(Sender: TObject);
Var ICon : IContabilidade;
    Cod, des, Sin, Sta : String;
    Sdi : Double;
begin
  If Tree.Selected <> nil then
    If EditConta( cod, des, sin, sta ) then begin
      ICon := HRContab as IContabilidade;
      ICon.NewConta( 0, Cod, des, Sin, Sta );
    end;
  end;
end;
end.
```

Anexo 10 – Estrutura do formulário de manutenção de contas

```

object fContas: TfContas
  Left = 343
  Top = 131
  BorderStyle = bsDialog
  Caption = 'Árvore de Contas'
  ClientHeight = 472
  ClientWidth = 433
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  OnCreate = FormCreate
  PixelsPerInch = 96
  TextHeight = 13
  object Tree: TTreeView
    Left = 8
    Top = 8
    Width = 417
    Height = 425
    Ctl3D = False
    Indent = 19
    ParentCtl3D = False
    TabOrder = 0
    OnExpanding = TreeExpanding
  end
  object pnButtons: TPanel
    Left = 0
    Top = 439
    Width = 433
    Height = 33
    Align = alBottom
    BevelOuter = bvNone
    TabOrder = 1
    DesignSize = (
      433
      33)
    object SpeedButton1: TSpeedButton
      Left = 4
      Top = 4
      Width = 125
      Height = 26
      Caption = 'Nova sub-conta...'
      Glyph.Data = {
        76010000424D760100000000000007600000002800000020000000100000000100
        04000000000000010000120B0000120B000010000000000000000000000000
        8000008000000080800080000000800080008000007F7F7F00BFBFBF000000
        FF0000FF000000FFFF00FF000000FF00FF00FFFF0000FFFFFF00303333333333
        333337F333333333333330333333333333337F3333333333303300000300
        300337FF77777F77377330000BBB0333333337777F337F33333330330BB0333
        33337F373F773333333033300333333337F33733333333033300003003000
        333377777F77377733330BBB03333333337F337333333330BB03333333
        33373F773333333330033333333333377333333333333333333333333333}
      NumGlyphs = 2
      OnClick = SpeedButton1Click
    end
  end
end

```

```

end
object SpeedButton2: TSpeedButton
  Left = 132
  Top = 4
  Width = 125
  Height = 26
  Caption = 'Nova raiz...'
  Glyph.Data = {
    76010000424D76010000000000000760000002800000020000000100000000100
    04000000000000010000120B0000120B00001000000000000000000000000000
    8000008000000008080008000000080008000808000007F7F7F00BFBFBF000000
    FF0000FF000000FFFF00FF000000FF00FF00FFFF0000FFFF00303333333333
    333337F333333333333330333333333333333333337F33FFFF3FF3FF30330000300
    300337FF77777F77377330000BBB03333333337777F337F33333330330BB0333
    333337F373F773333333303330033333333337F3377333333330333333333
    333337F33FFFF3FF3FF30330000300300337FF7777F77377330000BBB0333
    33337777F337F33333330330BB033333337F373F77333333303330033333
    33337F33773333333303333333333337FFFF3FF3FFF33300003003000
    33337777F7737733330BBB0333333333337F337F333333330BB03333333
    333373F7733333333303333333333337733333333333}
  NumGlyphs = 2
  OnClick = SpeedButton2Click
end
object btOK: TBitBtn
  Left = 266
  Top = 4
  Width = 79
  Height = 25
  Anchors = [akTop, akRight]
  Caption = '&OK'
  TabOrder = 0
  Kind = bkOK
end
object btCancel: TBitBtn
  Left = 350
  Top = 4
  Width = 79
  Height = 25
  Anchors = [akTop, akRight]
  Caption = '&Cancelar'
  TabOrder = 1
  Kind = bkCancel
end
end
object HRContab: THTTPIO
  WSDLLocation = 'http://localhost/stimba/webserv.dll/wsdl/IContabilidade'
  Service = 'IContabilidadeservice'
  HTTPWebNode.Agent = 'Borland SOAP 1.1'
  HTTPWebNode.UseUTF8InHeader = False
  HTTPWebNode.InvokeOptions = [soIgnoreInvalidCerts]
  Converter.Options = [soSendMultiRefObj, soTryAllSchema,
soRootRefNodesToBody]
  Left = 32
  Top = 16
end
end
end

```

Anexo 11 – Código fonte do formulário de cadastro de conta

```

unit u_EditConta;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, Grids, ValEdit, StdCtrls, Buttons, ExtCtrls;

type
  TfEditConta = class(TForm)
    pnButtons: TPanel;
    btOK: TBitBtn;
    btCancel: TBitBtn;
    grid: TValueListEditor;
    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

function EditConta( var cod, tit, sin, sta : String ): Boolean;

implementation

{$R *.dfm}
var fEditConta: TfEditConta;

function EditConta( var cod, tit, sin, sta : String ): Boolean;
begin
  result := false;
  If not assigned( fEditConta ) then fEditConta :=
    fEditConta.Create( Application );
  with fEditConta do begin
    grid.cells[1,1] := cod;
    grid.cells[1,2] := tit;
    grid.cells[1,3] := sin;
    grid.cells[1,4] := sta;
    If ShowModal = mrOk then begin
      cod := grid.cells[1,1];
      tit := grid.cells[1,2];
      sin := grid.cells[1,3];
      sta := grid.cells[1,4];
      result := True;
    end;
  end;
end;

procedure TfEditConta.FormShow(Sender: TObject);
begin
  grid.SetFocus;
end;

```

```
end.
```

Anexo 12 – Estrutura do formulário de cadastro de conta

```
object fEditConta: TEditConta
  Left = 437
  Top = 197
  Width = 338
  Height = 219
  Caption = 'Conta'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  Position = poScreenCenter
  OnShow = FormShow
  PixelsPerInch = 96
  TextHeight = 13
  object pnButtons: TPanel
    Left = 0
    Top = 152
    Width = 330
    Height = 33
    Align = alBottom
    BevelOuter = bvNone
    TabOrder = 0
    DesignSize = (
      330
      33)
    object btOK: TBitBtn
      Left = 163
      Top = 4
      Width = 79
      Height = 25
      Anchors = [akTop, akRight]
      Caption = '&OK'
      TabOrder = 0
      Kind = bkOK
    end
    object btCancel: TBitBtn
      Left = 247
      Top = 4
      Width = 79
      Height = 25
      Anchors = [akTop, akRight]
      Caption = '&Cancelar'
      TabOrder = 1
      Kind = bkCancel
    end
  end
  object grid: TValueListEditor
    Left = 0
    Top = 0
    Width = 330
```

```
Height = 152
Align = alClient
Color = 15987699
Ctl3D = False
FixedCols = 1
Options = [goFixedVertLine, goFixedHorzLine, goVertLine,
goHorzLine, goColSizing, goEditing, goTabs, goAlwaysShowEditor,
goThumbTracking]
ParentCtl3D = False
Strings.Strings = (
  'Código='
  'Descrição='
  'Sinal='
  'Status=')
TabOrder = 1
TitleCaptions.Strings = (
  'Descrição'
  'Valor')
ColWidths = (
  97
  229)
RowHeights = (
  18
  18
  18
  18
  18)
end
end
```

Anexo 13 – Código fonte do formulário de manutenção de lançamentos

```

unit u_EditLanca;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Buttons, ExtCtrls, Mask, ToolEdit, Rio,
  SOAPHTTPClient, IContabilidade1, ufrmContas, ComCtrls, DB, DBClient,
  Grids, DBGrids;

type
  TfEditLanca = class(TForm)
    pnButtons: TPanel;
    btOK: TBitBtn;
    btCancel: TBitBtn;
    HRContab: THTTTPRIO;
    Label3: TLabel;
    Label4: TLabel;
    edHistorico: TEdit;
    edValor: TEdit;
    edData: TDateEdit;
    Label5: TLabel;
    pagedb: TPageControl;
    TabMonodb: TTabSheet;
    TabSheet2: TTabSheet;
    cbDebito: TComboEdit;
    Label1: TLabel;
    pagecr: TPageControl;
    tabmonocr: TTabSheet;
    tabmulticr: TTabSheet;
    cbCredito: TComboEdit;
    Label2: TLabel;
    multidb: TClientDataSet;
    multidbidconta: TIntegerField;
    multidbdescricao: TStringField;
    multidbvalor: TFloatField;
    multicr: TClientDataSet;
    IntegerField1: TIntegerField;
    StringField1: TStringField;
    FloatField1: TFloatField;
    dsmultidb: TDataSource;
    dsmulticr: TDataSource;
    DBGrid1: TDBGrid;
    DBGrid2: TDBGrid;
    multicrconta: TStringField;
    multidbconta: TStringField;
    procedure cbDebitoButtonClick(Sender: TObject);
    procedure cbCreditoButtonClick(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure DBGrid1EditButtonClick(Sender: TObject);
    procedure DBGrid2EditButtonClick(Sender: TObject);
  private
    { Private declarations }
  public

```

```

    { Public declarations }
end;

function EditLanca( lID : Integer ): Boolean;

implementation

{$R *.dfm}

var fEditLanca: TfEditLanca;

function EditLanca( lID : Integer ): Boolean;
Var ICont : IContabilidade;
    IVetPartidas : TPartidaArray;
    res : integer;
begin
    If not assigned( fEditLanca ) then fEditLanca :=
        TfEditLanca.Create( application );
    with fEditLanca do begin
        If lID = -1 then begin
            edData.Date := Now;
            cbDebito.Text := '';
            cbCredito.Text := '';
            edHistorico.Text := '';
            edValor.text := '';
        end;
        If ShowModal = mrOk then begin
            IVetPartidas := nil;
            If pagedb.ActivePage = TabMonoDb then begin
                SetLength( IVetPartidas, Length( IVetPartidas )+1 );
                IVetPartidas[High(IVetPartidas)] := TPartida.Create;
                IVetPartidas[High(IVetPartidas)].IDConta := cbDebito.Tag;
                IVetPartidas[High(IVetPartidas)].Valor :=
                    strtofloat( edValor.Text );
                IVetPartidas[High(IVetPartidas)].Status := 'D';
            end else begin
                multidb.DisableControls;
                try
                    multidb.First;
                    while not multidb.Eof do begin
                        SetLength( IVetPartidas, Length( IVetPartidas )+1 );
                        IVetPartidas[High(IVetPartidas)] := TPartida.Create;
                        IVetPartidas[High(IVetPartidas)].IDConta :=
                            multidb.fieldbyname( 'IDConta' ).AsInteger;
                        IVetPartidas[High(IVetPartidas)].Descricao :=
                            multidb.fieldbyname( 'DESCRICAO' ).AsString;
                        IVetPartidas[High(IVetPartidas)].Valor :=
                            multidb.fieldbyname( 'VALOR' ).AsFloat;
                        IVetPartidas[High(IVetPartidas)].Status := 'D';
                    multidb.Next;
                end;
            finally
                multidb.EnableControls;
            end;
        end;
    end;

    If pagecr.ActivePage = TabMonoCr then begin
        SetLength( IVetPartidas, Length( IVetPartidas )+1 );
        IVetPartidas[High(IVetPartidas)] := TPartida.Create;
    end;
end;

```



```

        IVetPartidas[High(IVetPartidas)].IDConta := cbCredito.Tag;
        IVetPartidas[High(IVetPartidas)].Valor :=
            strtofloat( edValor.Text );
        IVetPartidas[High(IVetPartidas)].Status := 'C';
    end else begin
        multicr.DisableControls;
        try
            multicr.First;
            while not multicr.Eof do begin
                SetLength( IVetPartidas, Length( IVetPartidas )+1 );
                IVetPartidas[High(IVetPartidas)] :=
                    TPartida.Create;
                IVetPartidas[High(IVetPartidas)].IDConta :=
                    multicr.fieldbyname( 'IDConta' ).AsInteger;
                IVetPartidas[High(IVetPartidas)].Descricao :=
                    multicr.fieldbyname( 'DESCRICAO' ).AsString;
                IVetPartidas[High(IVetPartidas)].Valor :=
                    multicr.fieldbyname( 'VALOR' ).AsFloat;
                IVetPartidas[High(IVetPartidas)].Status := 'C';
                multicr.Next;
            end;
        finally
            multicr.EnableControls;
        end;
    end;

    ICont := HRContab as IContabilidade;
    ICont.NewLancamento( Datetimetostr( edData.Date ),
        edHistorico.Text, StrToFloat( edValor.Text ), IVetPartidas);
end;
end;

procedure TfEditLanca.cbDebitoButtonClick(Sender: TObject);
var aux : String;
begin
    cbDebito.Tag := getConta( aux );
    cbDebito.Text := aux;
end;

procedure TfEditLanca.cbCreditoButtonClick(Sender: TObject);
var aux : String;
begin
    cbCredito.Tag := getConta( aux );
    cbCredito.Text := aux;
end;

procedure TfEditLanca.FormShow(Sender: TObject);
begin
    If Visible then edData.SetFocus;
end;

procedure TfEditLanca.DBGrid1EditButtonClick(Sender: TObject);
var aux : string;
begin
    If not (multidb.State in [dsEdit,dsInsert]) then
        If multidb.Eof then multidb.Append else multidb.Edit;

    multidb.FieldName( 'IDCONTA' ).AsInteger := GetConta( aux );

```

```
    multidb.FieldName('CONTA').AsString      := Aux;
end;

procedure TfEditLanca.DBGrid2EditButtonClick(Sender: TObject);
var aux : string;
begin
    If not (multicr.State in [dsEdit,dsInsert]) then
        If multicr.Eof then multicr.Append else multicr.Edit;

    multicr.FieldName('IDCONTA').AsInteger := GetConta( aux );
    multicr.FieldName('CONTA').AsString   := Aux;
end;

end.
```

Anexo 14 – Estrutura do formulário de manutenção de lançamentos

```
object fEditLanca: TfEditLanca
  Left = 333
  Top = 272
  BorderStyle = bsDialog
  Caption = 'Editar Lançamento'
  ClientHeight = 375
  ClientWidth = 472
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  OnShow = FormShow
  PixelsPerInch = 96
  TextHeight = 13
  object Label3: TLabel
    Left = 8
    Top = 60
    Width = 27
    Height = 13
    Caption = 'Valor:'
  end
  object Label4: TLabel
    Left = 8
    Top = 36
    Width = 44
    Height = 13
    Caption = 'Histórico:'
  end
  object Label5: TLabel
    Left = 8
    Top = 12
    Width = 26
    Height = 13
    Caption = 'Data:'
  end
  object pnButtons: TPanel
    Left = 0
    Top = 342
    Width = 472
    Height = 33
    Align = alBottom
    BevelOuter = bvNone
    TabOrder = 5
    DesignSize = (
      472
      33)
    object btOK: TBitBtn
      Left = 305
      Top = 4
      Width = 79
      Height = 25
      Anchors = [akTop, akRight]
      Caption = '&OK'
      TabOrder = 0
      Kind = bkOK
    end
    object btCancel: TBitBtn
```

```
    Left = 389
    Top = 4
    Width = 79
    Height = 25
    Anchors = [akTop, akRight]
    Caption = '&Cancelar'
    TabOrder = 1
    Kind = bkCancel
end
end
object edHistorico: TEdit
    Left = 56
    Top = 32
    Width = 409
    Height = 21
    AutoSize = False
    Ctl3D = False
    ParentCtl3D = False
    TabOrder = 1
end
object edValor: TEdit
    Left = 56
    Top = 56
    Width = 145
    Height = 21
    AutoSize = False
    Ctl3D = False
    ParentCtl3D = False
    TabOrder = 2
end
object edData: TDateEdit
    Left = 56
    Top = 8
    Width = 121
    Height = 21
    Ctl3D = False
    NumGlyphs = 2
    ParentCtl3D = False
    TabOrder = 0
end
object pagedb: TPageControl
    Left = 0
    Top = 86
    Width = 472
    Height = 128
    ActivePage = TabMonodb
    Align = alBottom
    TabIndex = 0
    TabOrder = 3
    object TabMonodb: TTabSheet
        Caption = 'Conta única'
        object Labell: TLabel
            Left = 8
            Top = 24
            Width = 63
            Height = 13
            Caption = 'Conta débito:'
        end
        object cbDebito: TComboEdit
            Left = 8
            Top = 40
            Width = 201
            Height = 21
            Ctl3D = False
            GlyphKind = gkEllipsis
```

```

    NumGlyphs = 1
    ParentCtl3D = False
    TabOrder = 0
    OnButtonClick = cbDebitoButtonClick
end
end
object TabSheet2: TTabSheet
  Caption = 'Várias contas'
  ImageIndex = 1
  object DBGrid1: TDBGrid
    Left = 0
    Top = 0
    Width = 464
    Height = 100
    Align = alClient
    BorderStyle = bsNone
    Ctl3D = False
    DataSource = dsmultidb
    ParentCtl3D = False
    TabOrder = 0
    TitleFont.Charset = DEFAULT_CHARSET
    TitleFont.Color = clWindowText
    TitleFont.Height = -11
    TitleFont.Name = 'MS Sans Serif'
    TitleFont.Style = []
    OnEditButtonClick = DBGrid1EditButtonClick
    Columns = <
      item
        ButtonStyle = cbsEllipsis
        Expanded = False
        FieldName = 'conta'
        Width = 120
        Visible = True
      end
      item
        Expanded = False
        FieldName = 'descricao'
        Width = 240
        Visible = True
      end
      item
        Expanded = False
        FieldName = 'valor'
        Visible = True
      end>
    end
  end
end
object pagecr: TPageControl
  Left = 0
  Top = 214
  Width = 472
  Height = 128
  ActivePage = tabmonocr
  Align = alBottom
  TabIndex = 0
  TabOrder = 4
  object tabmonocr: TTabSheet
    Caption = 'Conta única'
    object Label2: TLabel
      Left = 7
      Top = 24
      Width = 66
      Height = 13
      Caption = 'Conta crédito:'
    end
  end
end

```

```

end
object cbCredito: TComboEdit
  Left = 8
  Top = 40
  Width = 201
  Height = 21
  Ctl3D = False
  GlyphKind = gkEllipsis
  NumGlyphs = 1
  ParentCtl3D = False
  TabOrder = 0
  OnButtonClick = cbCreditoButtonClick
end
end
object tabmulticr: TTabSheet
  Caption = 'Várias contas'
  ImageIndex = 1
  object DBGrid2: TDBGrid
    Left = 0
    Top = 0
    Width = 464
    Height = 100
    Align = alClient
    BorderStyle = bsNone
    Ctl3D = False
    DataSource = dsmulticr
    ParentCtl3D = False
    TabOrder = 0
    TitleFont.Charset = DEFAULT_CHARSET
    TitleFont.Color = clWindowText
    TitleFont.Height = -11
    TitleFont.Name = 'MS Sans Serif'
    TitleFont.Style = []
    OnEditButtonClick = DBGrid2EditButtonClick
    Columns = <
      item
        ButtonStyle = cbsEllipsis
        Expanded = False
        FieldName = 'conta'
        Width = 125
        Visible = True
      end
      item
        Expanded = False
        FieldName = 'descricao'
        Width = 240
        Visible = True
      end
      item
        Expanded = False
        FieldName = 'valor'
        Visible = True
      end
    end>
  end
end
end
object HRContab: THTTTPRIO
  WSDLLocation = 'http://localhost/stimba/webserv.dll/wsdl/IContabilidade'
  Service = 'IContabilidadeservice'
  HTTPWebNode.Agent = 'Borland SOAP 1.1'
  HTTPWebNode.UseUTF8InHeader = False
  HTTPWebNode.InvokeOptions = [soIgnoreInvalidCerts]
  Converter.Options = [soSendMultiRefObj, soTryAllSchema,
soRootRefNodesToBody]
  Left = 64

```

```

    Top = 16
end
object multidb: TClientDataSet
  Active = True
  Aggregates = <>
  FieldDefs = <
    item
      Name = 'idconta'
      DataType = ftInteger
    end
    item
      Name = 'descricao'
      DataType = ftString
      Size = 100
    end
    item
      Name = 'valor'
      DataType = ftFloat
    end
    item
      Name = 'conta'
      DataType = ftString
      Size = 60
    end
  end>
  IndexDefs = <>
  Params = <>
  StoreDefs = True
  Left = 192
  Top = 120
  Data = {
    700000009619E0BD01000000180000000400000000003000000700007696463
    6F6E746104000100000000000964657363726963616F01004900000001000557
    494454480200020064000576616C6F72080004000000000005636F6E74610100
    490000000100055749445448020002003C000000}
  object multidbidconta: TIntegerField
    DisplayWidth = 20
    FieldName = 'idconta'
    Visible = False
  end
  object multidbdescricao: TStringField
    DisplayWidth = 54
    FieldName = 'descricao'
    Size = 100
  end
  object multidbvalor: TFloatField
    DisplayWidth = 12
    FieldName = 'valor'
  end
  object multidbconta: TStringField
    FieldName = 'conta'
    Size = 60
  end
end
object multicr: TClientDataSet
  Active = True
  Aggregates = <>
  FieldDefs = <
    item
      Name = 'idconta'
      DataType = ftInteger
    end
    item
      Name = 'descricao'
      DataType = ftString
      Size = 100

```

```

end
item
  Name = 'valor'
  DataType = ftFloat
end
item
  Name = 'conta'
  DataType = ftString
  Size = 60
end>
IndexDefs = <>
Params = <>
StoreDefs = True
Left = 416
Top = 120
Data = {
  700000009619E0BD01000000180000000400000000000300000007000007696463
  6F6E746104000100000000000964657363726963616F01004900000001000557
  494454480200020064000576616C6F720800040000000000005636F6E74610100
  490000000100055749445448020002003C000000}
object IntegerField1: TIntegerField
  DisplayWidth = 20
  FieldName = 'idconta'
  Visible = False
end
object StringField1: TStringField
  DisplayWidth = 50
  FieldName = 'descricao'
  Size = 100
end
object FloatField1: TFloatField
  DisplayWidth = 12
  FieldName = 'valor'
end
object multicrconta: TStringField
  FieldName = 'conta'
  Size = 60
end
end
object dsmultidb: TDataSource
  DataSet = multidb
  Left = 200
  Top = 128
end
object dsmulticr: TDataSource
  DataSet = multicr
  Left = 424
  Top = 128
end
end
end

```


Anexo 15 – Código fonte do formulário principal

```

unit u_MainLanca;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, Grids, ExtCtrls, Rio, SOAPHTTPClient, StdCtrls,
  Buttons, u_EditLanca, IContabilidade, ufrmContas;

type
  TfLanca = class(TForm)
    grid: TStringGrid;
    Panell: TPanel;
    HRContab: THTTTPRIO;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    procedure FormCreate(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
  private
    fCurrentIndex : Integer;
    fCurrentStep  : Integer;
  public
    function MoveTo( Indice, Step : Integer ): Integer;
  end;

var fLanca: TfLanca;

implementation
{$R *.dfm}

procedure TfLanca.FormCreate(Sender: TObject);
begin
  grid.Cells[0,0] := 'Data';
  grid.Cells[1,0] := 'Conta DB';
  grid.Cells[2,0] := 'Conta CR';
  grid.Cells[3,0] := 'Historico';
  grid.Cells[4,0] := 'Valor';
  fCurrentIndex := 0;
  fCurrentStep  := 10;
  fCurrentIndex := fCurrentIndex + Moveto( fCurrentIndex,
fCurrentStep );
end;

procedure TfLanca.BitBtn1Click(Sender: TObject);
begin
  EditLanca( -1 );
end;

```

```

function TfLanca.Moveto(Indice, Step : Integer): Integer;
Var ICon : IContabilidade;
    lanca : TLancamentoArray;
    Conta : TConta;
    X,Y : Integer;
    IDCD,IDCC, QTCD, QTCC : Integer;
begin
    ICon := HRContab as IContabilidade;
    lanca := ICon.GetLancamentos( Indice, Step );
    Result := Length( Lanca );
    grid.RowCount := Length( lanca ) + 1;
    for x := low( lanca ) to high( lanca ) do begin
        grid.Cells[0,x+1] := Datetimetostr( lanca[x].Data );
        grid.Cells[3,x+1] := lanca[x].Descricao;
        grid.Cells[4,x+1] := formatfloat( '###,###,##0.00',
            lanca[x].Valor );
        QTCD := 0; QTCC := 0;
        for Y:=Low(lanca[x].Partidas) to High(lanca[x].Partidas) do begin
            If lanca[x].Partidas[y].Status = 'D' then begin
                Inc( QTCD );
                IDCD := lanca[x].Partidas[y].IDConta;
            end;
            If lanca[x].Partidas[y].Status = 'C' then begin
                Inc( QTCC );
                IDCC := lanca[x].Partidas[y].IDConta;
            end;
        end;
    end;

    If Qtcd = 1 then begin
        Conta := ICon.GetConta( Idcd );
        If conta <> nil then
            Grid.Cells[1,x+1] := Conta.Titulo;
    end else
        If Qtcd > 1 then Grid.Cells[1,x+1] := '[Multi]';
    If Qtcc = 1 then begin
        Conta := ICon.GetConta( Idcc );
        Grid.Cells[2,x+1] := Conta.Titulo;
    end else
        If Qtcc > 1 then Grid.Cells[2,x+1] := '[Multi]';
    end;
end;

procedure TfLanca.BitBtn3Click(Sender: TObject);
begin
    fCurrentIndex :=
        fCurrentIndex + Moveto( fCurrentIndex, fCurrentStep );
end;

procedure TfLanca.BitBtn4Click(Sender: TObject);
begin
    fCurrentIndex :=
        fCurrentIndex - MoveTo( fCurrentIndex, -fCurrentStep );
end;

procedure TfLanca.BitBtn5Click(Sender: TObject);
begin
    ShowContas;
end;

```

end.

Anexo 16 – Estrutura do formulário principal

```

object fLanca: Tflanca
  Left = 189
  Top = 132
  Width = 740
  Height = 530
  Caption = 'Lançamentos'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  OnCreate = FormCreate
  PixelsPerInch = 96
  TextHeight = 13
  object grid: TStringGrid
    Left = 0
    Top = 0
    Width = 732
    Height = 466
    Align = alClient
    Ctl3D = False
    DefaultRowHeight = 17
    FixedCols = 0
    ParentCtl3D = False
    TabOrder = 0
    ColWidths = (
      73
      131
      142
      264
      104)
  end
  object Panel1: TPanel
    Left = 0
    Top = 466
    Width = 732
    Height = 30
    Align = alBottom
    TabOrder = 1
    object BitBtn1: TBitBtn
      Left = 4
      Top = 3
      Width = 77
      Height = 25
      Caption = 'Novo...'
      TabOrder = 0
      OnClick = BitBtn1Click
      Glyph.Data = {
        76010000424D7601000000000000076000000280000002000000010000000100
        040000000000000010000130B0000130B000010000000000000000000000000
        8000008000000080000080000080000080000007F7F7F00BFBFBF000000
        FF0000FF000000FFFF00FF000000FF00FF00FFFF0000FFFFFF00333333333333
        33333333FF33337F33337F333377BB3777BB7777BB3777FFFF77FFFF773333B000000000
        0B333377777777777733330FFFFFFF0733337F3333337F333330FFFFFFF
        0733337F3333337F333330FFFFFFF0733337F3333337F333330FFFFFFF
        07333FF7F3333337FFFBBB0FFFFFFF0BB37777F333333777F33B0FFFFFFF
        0BBB3777F3333FFF77773330FFFF000003333337F3337777333330FFFF0FF0
      }
    end
  end
end

```

```

33333337F3337F37F3333330FFFF0F0B3333337F3337F77FF33330FFFF003B
B3333337FFF77377FF333B000000333BB3333777777F3377FF3BB3333BB333
3BB33773333773333773B333333B333333B733333373333337}
NumGlyphs = 2
end
object BitBtn2: TBitBtn
  Left = 84
  Top = 3
  Width = 77
  Height = 25
  Caption = 'Editor...'
  TabOrder = 1
  Glyph.Data = {
    76010000424D7601000000000000760000002800000020000000100000000100
    04000000000000010000120B0000120B00001000000000000000000000000000
    8000008000000008080008000000080008000808000007F7F7F00BFBFBF000000
    FF0000FF000000FFFF00FF000000FF00FF00FFFF0000FFFFFF00333333000000
    000033333377777777773333330FFFFFFF03FF3FF7FF33F3FF700300000FF0F
    00F077F77773F737737E00BFBFB0FFFFFFF07773333F7F3333F7E0BFBF00FFF
    F0F077F3337773F3F737E0BFBFBF0F00FF077F3333FF7F77F37E0BFBF00000B
    0FF077F33377737337E0BFBFBFB0FFF077F33FFFFFF73337E0BF0000000F
    FFF077FF77777733FF7000BFB00B0FF00F07773FF77373377373330000B0FFF
    FFF0333777373333FF7333330B0FFFF00003333373733FF777733330B0FF00F
    0FF03333737F37737F373330B00FFFF0F033337F77F33337F733309030FFFFF
    00333377737FFF7733330330000000333333733777777333}
NumGlyphs = 2
end
object BitBtn3: TBitBtn
  Left = 253
  Top = 3
  Width = 81
  Height = 25
  Caption = 'Próximo'
  TabOrder = 2
  OnClick = BitBtn3Click
  Glyph.Data = {
    76010000424D7601000000000000760000002800000020000000100000000100
    04000000000000010000120B0000120B00001000000000000000000000000000
    8000008000000008080008000000080008000808000007F7F7F00BFBFBF000000
    FF0000FF000000FFFF00FF000000FF00FF00FFFF0000FFFFFF00333333333333
    333333FF3333333333334473333333333377FFF3333333333744473333333
    333337773FF33333333344444733333333373F773FF333333334444447333
    3333373F3773FF333333374444444733333337F333773FF33333344444444
    73333373F3333773FF33334444444444733FFF7FFFFFF77FF999999999999
    9999777777777733733333333333333333333333333333333333333333333333
    3333337F3333F7733333333333333333333333333333333333333333333333
    33333733F77333333333333333333333333333333333333333333333333333
    333337773333333333333333333333333333333333333333333333333333}
Layout = blGlyphRight
Margin = 5
NumGlyphs = 2
Spacing = 5
end
object BitBtn4: TBitBtn
  Left = 168
  Top = 3
  Width = 81
  Height = 25
  Caption = 'Anterior'
  TabOrder = 3
  OnClick = BitBtn4Click
  Glyph.Data = {
    76010000424D7601000000000000760000002800000020000000100000000100
    04000000000000010000120B0000120B00001000000000000000000000000000
    8000008000000008080008000000080008000808000007F7F7F00BFBFBF000000

```

