Leonardo de Paula Batista Benevides

Meu Projeto

Projeto Final de Graduação

Projeto Final apresentado ao Curso Engenharia de Computação como requisito parcial para obtenção do título de Engenheiro de Computação

Orientador: Prof. Waldemar Celes

Rio de Janeiro
Novembro de 2012

Leonardo de Paula Batista Benevides

Meu Projeto

Centro Técnico Científico, Departamento de Informática,
Engenharia de Computação.

Prof. Waldemar Celes
Orientador
Departamento de Informática — PUC–Rio

Rio de Janeiro, 21 de Novembro de 2012

Leonardo de Paula Batista Benevides

Ficha Catalográfica

Meu Projeto / Leonardo de Paula Batista Benevides; orientador: Waldemar Celes. — Rio de Janeiro : PUC–Rio, Departamento de Informática, 2012.

v., 30 f: il. ; 29,7 cm

1. Projeto Final de Graduação - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui referências bibliográficas.

1. Informática – Tese. 2. Linguagem visual, Programaï¿½ï¿½o por usuï¿½rios finais. I. Celes, Waldemar. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 510

Resumo

; Celes, Waldemar. **Meu Projeto**. Rio de Janeiro, 2012. 30p. Projeto Final de Graduação — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Este documento apresenta os principais conceitos de um sistema de *script* de uma aplicaï¿½ï¿½o voltada para o pï¿½ï¿½s-processamento de simulaï¿½ï¿½es de reservatï¿½ï¿½rios de petrï¿½ï¿½leo e a proposta e implementaï¿½ï¿½o de uma ferramenta que disponibilze parte dos recursos obtidos a partir dos *scripts* ï¿½ï¿½ usuï¿½ï¿½rios nï¿½ï¿½o programadores. A ferramenta implementada ï¿½ï¿½ uma uma calculadora de propriedades que permite que o usuï¿½ï¿½rio gere novas propriedades a partir de simples expressï¿½ï¿½es aritmï¿½ï¿½ticas desenvolvidas em uma interface grï¿½ï¿½fica similar a uma calculadora comum. Este documento tambï¿½ï¿½m apresenta um breve resumo dos estudos feitos acerca de linguagens de programaï¿½ï¿½ï¿½o visual e a proposta e inï¿½ï¿½cio de implementaï¿½ï¿½ï¿½o de uma linguagem de programaï¿½ï¿½ï¿½o visual simplificada.

Palavras–chave

Linguagem visual, Programaï¿½ï¿½ï¿½o por usuï¿½ï¿½rios finais.

Abstract

; Celes, Waldemar. **A visual programming tool for postprocessing of oil reservoir simulations** . Rio de Janeiro, 2012. 30p. MsC Thesis — Department of Informática, Pontifícia Universidade Católica do Rio de Janeiro.

This document presents the main concepts of a script system of an application focused on the postprocessing of oil reservoir simulations and the proposal and the implementation of a tool that provides part of the scripts resources to users that are not programmers. This tool is an property calculator that lets the users generate new properties from simple arithmetic expressions that are developed in an graphic interface that looks like an ordinary calculator. This document also presents a brief summary of studies about visual programming languages and the proposal and the beginning of a simplified visual programming language.

Keywords

Visual language, End users pogramming .

Sumário

Lista de figuras

Lista de tabelas

1
Introduï¿½ï¿½o

O **Geresim** é uma aplicação desenvolvida por meio de uma parceria firmada entre o **TeCGraf/PUC-Rio** e a **PETROBRAS (E&P/ER)** com o intuito de melhorar o processo de modelagem do escoamento em meio poroso em reservatórios de petróleo. Ele é constituï¿½ío de um ambiente de trabalho que permite construir, alterar, manter, importar e exportar modelos numéricos para os principais simuladores comerciais, tirando proveito de uma série de facilidades voltadas para a agilidade e a eficácia da montagem dos modelos. Adicionalmente, o **Geresim** executa uma análise de consistência que elimina a maioria dos erros mais comuns no processo de modelagem e oferece um ambiente para analisar modelos e dados de reservatórios, utilizando visualizações bidimensionais de mapas e seções, tridimensionais do modelo ou gráficas de elementos do modelo(**?**).

Uma das aplicações com maior usabilidade no **Geresim** é um poderoso sistema de *scripts* que possibilita acesso programável aos dados dos modelos de reservatórios visualizados e disponibiliza diversos recursos como geração de dados de forma procedimental e flexibilização nas operações com os dados. Esse acesso programável é feito através da linguagem de programação **Lua** (**?**).

No entanto, como requer certo conhecimento de tï¿½cnicas de programaï¿½ï¿½o do usuï¿½rio e este na maioria das vezes tem pouca ou atï¿½ mesmo nenhuma experiï¿½ncia como programador, o sistema de *scripts* muitas vezes acaba sendo subutilizado e os que necessitam do mesmo acabam delegando aos prï¿½prios desenvolvedores do software a implementaï¿½ï¿½o de novos *scripts* de programaï¿½ï¿½o.

Nesse contexto, torna-se interessante e necessï¿½rio (pois, foi sugerido pelo prï¿½prio cliente) o estudo de alternativas ï¿½ programaï¿½ï¿½o textual que disponibilize ao usuï¿½rio nï¿½o programador pelo menos parte dos recursos do sistema de *scripts* do **Geresim**. Por isso, esse projeto teve como metas principais pesquisar e desenvolver ferramentas que cumpram esse objetivo.

Este trabalho foi dividido da seguinte forma: a seï¿½ï¿½o 2 apresenta de forma breve o atual sistema de script do **Geresim**; a seï¿½ï¿½o 3 descreve

a implementaï¿½ï¿½o e funcionamento de uma calculadora de propriedades desenvolvida no projeto; a seï¿½ï¿½o 4 apresenta um breve resumo dos estudos feitos acerca de lingugens de programaï¿½ï¿½o visual; a seï¿½ï¿½o 5 apresenta o inï¿½cio da implementaï¿½ï¿½o de uma ferramenta de programaï¿½ï¿½o visual; a seï¿½ï¿½o 6 apresenta algumas conclusï¿½es e propostas de trabalhos futuros.

2
O sistema de Scripts

O sistema de *scripts* do **Geresim** possibilita acesso programï¿½vel aos dados dos modelos de reservatï¿½rio visualizados. Esse acesso programï¿½vel ï¿½ feito atravï¿½s da linguagem de programaï¿½ï¿½o **Lua**. Por isso o sistema disponibiliza diversos recursos de programaï¿½ï¿½o sendo possï¿½vel a implementaï¿½ï¿½o de *scripts* diversos que podem acessar todos os dados dos modelos de reservatï¿½rio.

Assim ï¿½ possï¿½vel gerear dados de forma procedimental, como cirar novos modelos e novos dados a serem associados a modelos, manipular dados de forma flexï¿½vel, como usï¿½-los como operandos de diversas operaï¿½ï¿½es sem muitas restriï¿½ï¿½es e criar *scripts* temporï¿½rios. Por isso o sistema pode funcionar como uma mï¿½quina de calcular que opera com os dados dos modelos, recurso que serï¿½ mais discutido na seï¿½ï¿½o 3.

Nessas seï¿½ï¿½o serï¿½ feita uma rï¿½pida descriï¿½ï¿½o da linguagem de programaï¿½ï¿½o **Lua** e uma apresentaï¿½ï¿½o do funcionamento do sistema de *script* do **Geresim**.

2.1
Lua

Lua ï¿½ uma linguagem de programaï¿½ï¿½o poderosa, rï¿½pida e leve, projetada justamente para estender aplicaï¿½ï¿½es. Combina sintaxe simples para programaï¿½ï¿½o procedural com poderosas construï¿½ï¿½es para descriï¿½ï¿½o de dados baseadas em tabelas associativas e semï¿½ntica extensï¿½vel. **Lua** ï¿½ tipada dinamicamente, ï¿½ interpretada a partir de bytecodes para uma mï¿½quina virtual baseada em registradores, e tem gerenciamento automï¿½tico de memï¿½ria com coleta de lixo incremental. Essas caracterï¿½sticas fazem de **Lua** uma linguagem ideal para configuraï¿½ï¿½o, automaï¿½ï¿½o (*scripting*) e prototipagem rï¿½pida. A linguagem ï¿½ usada em muitas aplicaï¿½ï¿½es industriais, com ï¿½nfase em sistemas embutidos e jogos, onde ï¿½ atualmente a linguagem de *script* mais usada. **Lua** tem uma merecida reputaï¿½ï¿½o de ï¿½timo desempenho, em vï¿½rios benchmarks aparece como a linguagem mais rï¿½pida dentre as linguagens de *script* interpretadas. (**?**)

Mais do que apenas estender aplicaï¿½ï¿½es, **Lua** tambï¿½m suporta uma abordagem de desenvolvimento de software baseada em componentes, onde ï¿½ possï¿½vel criar aplicaï¿½ï¿½es juntando componentes de alto-nï¿½vel prï¿½-existentes. Normalmente esses componentes sï¿½o escritos em linguagens compiladas e estaticamente tipadas como **C** ou **C++**. Assim, **Lua** pode ser usada conectar tais componentes, que normalmente representam conceitos mais concretos e de baixo nï¿½vel (como por exemplo estruturas de dados) que nï¿½o estï¿½o sujeitos a muitas alteraï¿½ï¿½es durante o desenvolvimento do programa e normalmente exigem um maior poder de processamento. (**?**)

2.2 Conceitos bï¿½sicos do sistema de Script

O sistema de macros do **Geresim** provï¿½ acesso a diversos objetos que representam o modelo e seus dados. Para cada objeto, tem-se acesso a suas informaï¿½ï¿½es e a um conjunto de mï¿½todos que permitem a manipulaï¿½ï¿½o alteraï¿½ï¿½o das informaï¿½ï¿½es associadas. Os objetos a que se tem acesso no sistema de macros sï¿½o:

- **Model:** objeto que representa um modelo de reservatï¿½rio;
- **Field:** objeto que representa um campo do modelo;
- **Region:** objeto que representa uma regiï¿½o do modelo;
- **Well:** objeto que representa um poï¿½o do modelo;
- **Completion:** objeto que representa uma completaï¿½ï¿½o de um poï¿½o do modelo;
- **Data:** objeto que representa um conjinto de valores de propriedades, podendo ser dos seguintes tipos:
  - **MapProp:** propriedade de mapa, representada por um vetor de dados de dimensï¿½o 4: ni x nj x nk x nï¿½mero de timesteps[1] de mapa;
  - **FieldProp:** propriedade de campo, representada por um vetor de dados de dimensï¿½o 1: nï¿½mero de timesteps de poï¿½o;
  - **RegionProp:** propriedade de regiï¿½o, representada por um vetor de dados de dimensï¿½o 2: nï¿½mero de regiï¿½es x nï¿½mero de timesteps de poï¿½o;
  - **GroupProp:** propriedade de grupo, representada por um vetor de dados de dimensï¿½o 2: nï¿½mero de grupos x nï¿½mero de timesteps de poï¿½o;

[1] explicaï¿½ï¿½o de timestep

  - **WellProp:** propriedade de poï¿½o, representada por um vetor de dados de dimensï¿½o 2: nï¿½mero de poï¿½os x nï¿½mero de timesteps de poï¿½o;
  - **CompProp:** propriedade de completaï¿½ï¿½o, representada por um vetor de dados de dimensï¿½o 3: nï¿½mero de poï¿½os x nï¿½mero de mï¿½ximo de completaï¿½ï¿½es por poï¿½o x nï¿½mero de timesteps de poï¿½o;
  - **ExtraProp:** propriedade avulsa, representada por um vetor de dados de dimensï¿½o 1: nï¿½mero de timesteps de poï¿½o;
- **Slice:** referï¿½ncia para uma "fatia" de um determinado conjunto de dados.

Pode-se ainda criar referï¿½ncias para outras porï¿½ï¿½es de um dado. Por exemplo, ï¿½ vï¿½lido criar uma referï¿½ncia para um mapa associado a um determinado timestep.

Existem dois tipos de vetores de dados: dados originais associados ao modelo e dados criados dinamicamente pelo sistema de *script*. Os dados originais podem ser consultados, mas nï¿½o podem ser alterados. Jï¿½ os dados criados dinamicamente podem ser livremente alterados. Analogamente, referï¿½ncias a dados originais nï¿½o sï¿½o editï¿½veis, enquanto que referï¿½ncias a dados criados dinamicamente sï¿½o editï¿½veis (na verdade, altera-se o vetor de dados referenciado).

Logicamente, o programa de *script* pode tambï¿½m incluir a manipulaï¿½ï¿½o de valores dos tipos existentes em **Lua**: booleanos, valores escalares, cadeias de caracteres (*string*), entre outros.

2.3 Acesso e criaï¿½ï¿½o de dados

O objeto *Model* representa um resultado de simulaï¿½ï¿½o, com suas propriedades de mapa e de poï¿½o. A funï¿½ï¿½o que implementa o *script* recebe um ou mais modelos como parï¿½metro de entrada e a partir desses objetos *Model*, tem-se acesso a seus atributos e seus mï¿½todos.

ï¿½ atravï¿½s dos atributos de um objeto *Model* que se dï¿½ o acesso programï¿½vel aos dados dos modelos de reservatï¿½rio. Esses dados nï¿½o podem ser alterados no *script*, ï¿½ possï¿½vel apenas consulta-los e usar os valores retornados. Assim ï¿½ possï¿½vel acessar dados de geometria, como o nï¿½mero de cï¿½lulas nas direï¿½ï¿½es I, J ou K, acessar dados relacionados a timesteps, acessar dados relacionados aos demais objetos do modelo, como o nï¿½mero de poï¿½os ou de regiï¿½es, acessar os prï¿½prios objetos e tambï¿½m a todas as propriedades do modelo.

Como dito acima, o objeto *Model* tambï¿½m proviï¿½ um conjunto de mï¿½todos que possibilitam o acesso iï¿½ estrutura do reservatiï¿½rio, a criaï¿½ï¿½o de novos modelos, criaï¿½ï¿½o de novos vetores de propriedade, verificar a existiï¿½ncia de timesteps e propriedades e associar os novos dados vetoriais criados durante o *script* aos modelos.

2.4 Vetores de dados de propriedades

Como visto anteriormente, os valores das propriedades sï¿½o representados em vetores de dados. Vetores de propriedade de mapa, por exemplo, tï¿½m dimensï¿½o 4 (ni x nj x nk x nï¿½mero de timesteps de mapa) e sï¿½ podem ser usados em operaï¿½ï¿½es com outros vetores de mapa de mesmas dimensï¿½es. ï¿½ permitida a criaï¿½ï¿½o de "fatias" (*slices*) que fazem referï¿½ncia a uma parte de um vetor 4D, ou seja, pode-se representar uma propriedade de uma camada de um reservatiï¿½rio atravï¿½s de uma referï¿½ncia ao vetor 4D, onde a terceira dimensï¿½o ï¿½ fixada.

As dimensï¿½es de uma fatia que representa uma camada sï¿½o: ni x nj x 1 x nï¿½mero de timesteps de mapa. Conceitualmente, uma camada ï¿½ um vetor 3D, jï¿½ que o ï¿½ndice k nï¿½o varia. No entanto, internamente, ela ï¿½ representada por um vetor 4D, como se os valores associados ï¿½ camada em questï¿½o fossem replicados para as demais camadas do grid. Desta forma, pode-se fazer operaï¿½ï¿½es entre um grid e uma camada especï¿½fica. Generalizando, pode-se fazer operaï¿½ï¿½es entre vetores e fatias de vetores de mesmas dimensï¿½es. Tambï¿½m ï¿½ possï¿½vel associar uma fatia a um modelo, para ser visualizada atravï¿½s da interface grï¿½fica. Neste caso, os valores sï¿½o replicados ao longo das dimensï¿½es fixadas.

Conceitualmente, uma propriedade de campo ï¿½ um vetor 1D (nï¿½mero de timesteps), uma propriedade de regiï¿½o ï¿½ um vetor 2D (nï¿½mero de regiï¿½es x nï¿½mero de timesteps), uma propriedade de grupo ï¿½ um vetor 2D (nï¿½mero de grupos x nï¿½mero de timesteps), uma propriedade de poï¿½o ï¿½ um vetor 2D (nï¿½mero de poï¿½os x nï¿½mero de timesteps), e uma propriedade de completaï¿½ï¿½o ï¿½ um vetor 3D (nï¿½mero de poï¿½os x nï¿½mero de completaï¿½ï¿½es x nï¿½mero de timesteps).

Em todas as operaï¿½ï¿½es que envolvem dois vetores de dados como operandos, a operaï¿½ï¿½o sï¿½ ï¿½ possï¿½vel se os timesteps associados aos vetores forem compatï¿½veis. O primeiro operando ï¿½ usado com referï¿½ncia na operaï¿½ï¿½o. Para cada timestep existente no primeiro operando, deve haver um timestep correspondente no segundo operando. A correspondï¿½ncia entre timesteps ï¿½ feita buscando-se a ocorrï¿½ncia entre datas prï¿½ximas,

dentro de uma determinada tolerï¿½ncia, que pode ser configurada pelo programador. Se a operaï¿½ï¿½o entre dois vetores nï¿½o puder ser realizada, o sistema reporta uma mensagem de erro. De posse de um vetor de dados, pode-se fazer diversas operaï¿½ï¿½es e invocar diversos mï¿½todos associados. ï¿½ possï¿½vel realizar as operaï¿½ï¿½es aritmï¿½ticas [2] de soma, subtraï¿½ï¿½o, multiplicaï¿½ï¿½o, divisï¿½o e potenciaï¿½ï¿½o. Essas operaï¿½ï¿½es resultam em um novo vetor de dados, de mesmas dimensï¿½es, onde cada valor representa a operaï¿½ï¿½oaplicada aos valores associados aos vetores fornecidos.(**?**)

Estï¿½o disponï¿½veis tambï¿½m operadores relacionais e lï¿½gicos e alguns mï¿½todos de manipulaï¿½ï¿½o de valores.

[2] Tambï¿½m ï¿½ possï¿½vel realizar operaï¿½ï¿½es aritmï¿½ticas entre vetores de propriedades e valores escalares.

3
Calculadora de propriedades

Como visto anteriormente, um recurso interessante do sistema de *script* ï¿½ a possibilidade de usï¿½-lo como uma espï¿½cie de mï¿½quina de calcular que opera sobre os dados de modelos de reservatï¿½rios. Sendo assim, a primeira soluï¿½ï¿½o encontrada para resolver o problema apresentado na introduï¿½ï¿½o, oferecer parte dos recursos do sistema de *script* para o usuï¿½rio nï¿½o programador, foi implementar justamente uma calculadora de propriedades onde o resultado das operaï¿½ï¿½es fossem propriedades que pudessem ser posteriormente visualizadas pelo usuï¿½rio. Sendo assim, a ferramenta proposta deveria ter uma interface grï¿½fica na forma de uma calculadora e gerar um cï¿½digo compatï¿½vel com o sistema de *script* e que obviamente resultasse na propriedade desejada pelo usuï¿½rio.

3.1
Sintaxe das expressï¿½es

Inicialmente a calculadora foi pensada para funcionar apenas para propriedades de mapa. As propriedades de mapa sï¿½o valores associados ï¿½s cï¿½lulas do modelos, por isso o acesso aos dados dessas propriedades ï¿½ sempre independente dos demais objetos vistos na seï¿½ï¿½o 2.2. Portanto, a sintaxe das expressï¿½es deveria ser a sintaxe comum de expressï¿½es aritmï¿½ticas levemente modificada para atender trï¿½s requisitos:

1. Possibilitar o uso nï¿½o somente de propriedades mas tambï¿½m de "fatias" de propriedades com o timestep fixado.

2. Permitir o uso de funï¿½ï¿½es, para incluir outras funcionalidades do sistema de *script*.

3. Ser compatï¿½vel com **Lua**, para facilitar a implementaï¿½ï¿½o.

As "fatias" foram incluidas da seguinte forma: uma fatia ï¿½ o nome de propriedade (identificador) seguido de uma data entre aspas, como por exemplo: SO"10/10/2010". Em **Lua**, essa expressï¿½o ï¿½ encarada como uma chamada de funï¿½ï¿½o.

Posteriormente, foram incluï¿½das na calculadora tambï¿½m propriedades de campo, regiï¿½o, grupo e poï¿½o. No caso de tais propriedades, abriu-se a possibilidade de incluir tambï¿½m os respectivos elementos nas expressï¿½s. Dessa manera, a fatia da propriedade associada a um elemento (grupo, regiï¿½o ou poï¿½o) seria referenciada da seguinte maneira: nome do elemento seguido do nome da propriedade entre colchetes, como por exemplo: WELL[BHP]. Isso indicaria a fatia da propriedade BHP associada ao poï¿½o WELL. Em **Lua**, essa ecpressï¿½o ï¿½ encarada como uma indexaï¿½ï¿½o de uma tabela.

3.2 Implementaï¿½ï¿½o

A calculadora foi implementada na linguagem de programaï¿½ï¿½o **Lua** e a interface grï¿½fica foi implementada utilizando o toolkit para construï¿½ï¿½o de interfaces grï¿½ficas IUP. (**?**) A implementaï¿½ï¿½o foi dividida em trï¿½s mï¿½dulos: Calculator.lua, que implementa a interface grï¿½fica e controla a entrada de dados do usuï¿½rio, CalcComp.lua, responsï¿½vel pela geraï¿½ï¿½o de cï¿½digo e validaï¿½ï¿½o das expressï¿½es e por ï¿½ltimo CalcLib.lua, que implementa algumas funï¿½ï¿½es para serem incluidas no sistema de *script* que garantem o funcionamento do cï¿½digo gerado.

3.2.1 Mï¿½dulo Calculator.lua

Figura 3.1: Interface da Calculadora

A interface ï¿½ composta pelos seguintes campos:

– Um campo de texto onde ï¿½ possï¿½vel entrar o nome da nova propriedade, caso nï¿½o seja especificado um nome, ï¿½ usado a prï¿½pria expressï¿½o como nome.

- Lista que indica o tipo de propriedade que serï¿½ calculada, atualmente os tipos suportaqdos sï¿½o propriedade de mapa, campo, regiï¿½o e poï¿½o.
- Um toggle e a lista de elementos, que ï¿½ preencida de acordo com o valor da lista de tipos, se o tipo for mapa, esses dois controles nï¿½o sï¿½o habilitados, pois nï¿½o se aplicam a propriedades destes tipos.
- Lista de propriedades, que ï¿½ preenchida de acordo com o valor da lista de tipos.
- Um toggle e lista de timesteps, que ï¿½ preenchida de acordo com o valor da lista de tipos.
- Botï¿½o de inserir propriedade.
- "Teclado" formado por botï¿½es que disponibilizam os nï¿½meros, os operadores e algumas funï¿½ï¿½es para o usuï¿½rio.
- Um campo de texto onde pode-se visualizar a expressï¿½o.
- Botï¿½es de calcular e limpar expressï¿½o.

A interface funciona da seguinte forma: o usuï¿½rio pode entrar nï¿½meros e operadores ou movimentar o cursor livremente atravï¿½s de seu teclado ou clicando no teclado virtual da interface. Jï¿½ a entrada de variï¿½veis(propriedades) e funï¿½ï¿½es sï¿½ pode ser feita atravï¿½s da interface grï¿½fica. O objetivo disso ï¿½ evitar a geraï¿½ï¿½o de cï¿½digos inconsistentes.

Para se inserir funï¿½ï¿½es basta-se clicar no respectivo botï¿½o da funï¿½ï¿½o no teclado virtual para que a funï¿½ï¿½o seja inserida na posiï¿½ï¿½o corrente do cursor.

Para se inserir propriedades ï¿½ preciso clicar no botï¿½o inserir ao lado da lista de timesteps. O valor inserido depende dos valores das listas e dos toggles. Se nenhum dos toggles estiver habilitado, serï¿½ inserido apenas o nome da propriedade selecionada na lista de propriedades. Se o toggle da lista de elementos estiver habilitado, serï¿½ inserido a propriedade selecionada na lista de propriedades indexando o elemento selecionado na lista de elementos (exemplo: WELL[BHP]). Caso o toggle de timestep esteja habilitado, serï¿½ adicionado ao fim a data selecionada na lista de timesteps (exemplo: SO"10/10/2010").

Para que nï¿½o haja casos de entrada de nï¿½meros, operadores, funï¿½ï¿½es ou propriedades em posiï¿½ï¿½es indesejadas (como por exemplo no meio do nome de uma propriedade) foram implementadas algumas funï¿½ï¿½es que controlam o movimento e posiï¿½ï¿½o do cursor, evitando cenï¿½rios desse tipo.

3.2.2 Mï¿½dulo CalcComp.lua

Quando se pensa em geraï¿½ï¿½o de cï¿½digo para um problema como este, a primeira abordagem que nos vem a cabeï¿½a ï¿½ compilar as expressï¿½es e gerar o cï¿½digo desejado a partir delas. Porï¿½m, compilar cï¿½digos, por mais simples que sejam as expressï¿½es encontradas nesse caso, nï¿½o ï¿½ uma tarefa trivial. O desenvolvimento de um mini-compilador envolveria a implementaï¿½ï¿½o de um parser, a elaboraï¿½ï¿½o de uma gramï¿½tica, o uso de tï¿½cnicas como criaï¿½ï¿½o de ï¿½rvores de sintaxe e etc. Por isso, a abordagem que foi seguida vai praticamente na direï¿½ï¿½o oposta. Em vez de compilar as expressï¿½es e gerar um cï¿½digo de *script* a partir delas , o que fizemos foi adaptar o ambiente do sistema de *script* para que as prï¿½prias expressï¿½es pudessem ser executadas dentro desse ambiente e gerar o resultado esperado. ï¿½ por isso que era imprescindï¿½vel que as expressï¿½es fossem compatï¿½veis com **Lua**.

As adaptaï¿½ï¿½es sï¿½o feitas da seguinte maneira: ï¿½ desempenhada uma iteraï¿½ï¿½o sobre os identificadores presentes na expressï¿½o (nomes de propriedades ou dos demais elementos, como nomes de poï¿½os), ï¿½ verificado se o identificador se refere a uma propriedade ou a um elemento e entï¿½o ï¿½ gerado um cï¿½digo que inicializa uma varii¿½vel local de mesmo nome da propriedade ou elemento com uma chamada de funï¿½ï¿½o que retorna um objeto especial que possibilite a execuï¿½ï¿½o do cï¿½digo da expressï¿½o. Essa funï¿½ï¿½o recebe como parï¿½metro uma referï¿½ncia para o objeto que representa o modelo de reservatï¿½rio, um *string* indicando o tipo de dado sendo trabalhado (propriedade de mapa, de poï¿½o e etc) e um *string* contendo o nome da propriedade ou objeto. Alï¿½m disso, tambï¿½m ï¿½ gerado o cï¿½digo que armazena o resultado da expressï¿½o em uma varii¿½vel temporï¿½ria e o da chamada de funï¿½ï¿½o que associa o vetor de dados resultante ao modelo para ser posteriormente visualizado.

Segue abaixo o exemplo de um cï¿½digo gerado pelo mï¿½dulo para a expressï¿½o OIL1[QO] + QW“10/07/1981”:

```
calc.GetMacroEnv()
local OIL1 = calc.CreateElem(rf,"well","OIL1")
local QO = calc.CreateProp(rf,"well","QO")
local QW = calc.CreateProp(rf,"well","QW")
local newprop = OIL1[QO] + QW"10/07/1981"
calc.AttachProp(rf,temp,'newprop','well')
```

Nesse caso, QO e QW sï¿½o propriedades de poï¿½os, OIL1 ï¿½ o nome de um poï¿½o e rf representa um objeto do tipo *Model*.

A implementaï¿½ï¿½o dessas funï¿½ï¿½es e dos objetos retornados por elas serï¿½ discutida na prï¿½xima seï¿½ï¿½o.

O mï¿½dulo tambï¿½m disponibiliza uma funï¿½ï¿½o que testa a expressï¿½o com o intuito de prevenir erros de sintaxe e que funï¿½ï¿½es sejam chamadas sem que seus parï¿½metros tanham sido especificados.

3.2.3 Mï¿½dulo CalcLib.lua

O mï¿½dulo CalcLib.lua ï¿½ responsï¿½vel por implementar as funï¿½ï¿½es presentes no cï¿½digo gerado pelo mï¿½dulo CalcComp.lua. Essas funï¿½ï¿½es servem para adaptar o ambiente do sistema de *script* para que as prï¿½prias expressï¿½es possam ser executadas dentro desse ambiente e gerar o resultado esperado. As funï¿½ï¿½es sï¿½o armazenadas em uma tabela que ï¿½ inserida no ambiente de *script* (tabela calc). O principal desafio de implementar esse mï¿½dulo foi desenvolver objetos que se comportassem ora como operandos de expressï¿½es aritmï¿½ticas e ora como funï¿½ï¿½es (no caso de propriedades com o timestep especificado, que ï¿½ interpretado em **Lua** como uma chamada de funï¿½ï¿½o). Uma limitaï¿½ï¿½o que tambï¿½m teve que ser superada foi a de que, em operaï¿½ï¿½es aritmï¿½ticas entre vetores de dados e escalares no sistema de *script*, os escalares devem sempre ser o segundo operando.

Para solucionar esses problemas, foi utilizado o recurso de metamecanismos presente em **Lua**, que nos permite especificar como determinados objetos devem se comportar em algumas situaï¿½ï¿½es. Isso ï¿½ disponibilizado por meio da implementaï¿½ï¿½o de *metatables*, que sï¿½o tabelas que definem atravï¿½s de seus *metamï¿½todos* como os objetos associados a ela devem se comportar.

Os objetos retornados pela funï¿½ï¿½o *CreateProp*, que representam as propriedades nas expressï¿½es, sï¿½o tabelas que possuem campos que especificam o nome da propriedade, o tipo da propriedade e se o valor utilizado nos cï¿½lculos serï¿½ a propriedade associada a um determinado elemento ou serï¿½ o vetor de dados inteiro. Seu comportamento (determinado por sua *metatable*) ï¿½ o seguinte:

- O metamï¿½todo de todas as operaï¿½ï¿½es binï¿½rias ajusta a ordem dos termos, armazena o resultado da operaï¿½ï¿½o em um objeto intermediï¿½rio e o retorna.
- O metamï¿½todo responsï¿½vel pela situaï¿½ï¿½o em que o objeto ï¿½ chamado como uma funï¿½ï¿½o, armazena a “fatia” da propriedade referente

ao timestep passado como parï¿½metro em um objeto intermediï¿½rio e o retorna.

– O metamï¿½todo responsï¿½vel pelo tratamento de indexaï¿½ï¿½es ï¿½ o responsï¿½vel por buscar os vetores de dados que serï¿½o usados nos cï¿½lculos, por isso os demais metamï¿½todos acessam esses valores apenas indexando a prï¿½pria tabela.

O objeto intermediï¿½rio, que ï¿½ apenas uma tabela que armazena o resultado das operaï¿½ï¿½es, se mostrou necessï¿½rio para resolver a limitaï¿½ï¿½o da ordem dos termos em operaï¿½ï¿½es envolvendo escalares citada acima. Ele ï¿½ associado ï¿½ mesma *metatable* do objeto que representa as propriedades e por isso sempre ajusta a ordem dos termos nas operaï¿½ï¿½es.

Os objetos que representam nas expressï¿½es os elementos (poï¿½os, regiï¿½es e etc), sï¿½o criados pela funï¿½ï¿½o *CreateElem*. Eles sï¿½o tabelas contendo campos com o nome do elemento, o seu tipo e uma referï¿½ncia para o modelo que pertence. A *metatable* que ï¿½ associada a eles contï¿½m apenas o metamï¿½todo responsï¿½vel pelo tratamento de indexaï¿½ï¿½es. Ao ser indexada por um objeto que representa uma propriedade, esse metamï¿½todo extrai o nome da propriedade e cria um novo objeto para mesma preopriedade (chamando a funï¿½ï¿½o *CreateProp*) porï¿½m especificando que a propriedade estï¿½ associada agora ao elemento. Portanto, o vetor de dados utilizado nos cï¿½lculos serï¿½ referente a "fatia" da propriedade referente a esse elemento.

A funï¿½ï¿½o *AttachProp* associa o resultado da expressï¿½o ao modelo.

Este mï¿½dulo tambï¿½m inplementa as funï¿½ï¿½es disponï¿½veis na interface da calculadora:

– **max**: retorna o maior valor encontrado dentre os vetores de dados passados como parï¿½metros.

– **min**: retorna o menor valor encontrado dentre os vetores de dados passados como parï¿½metros.

– **sum**: retorna o somatï¿½rio dos valores do vetor de dados.

– **abs**: retorna um novo vetor de dados de mesma dimensï¿½es do vetor passado como parï¿½metros com os valores sendo igual aos respectivos valores absolutos do vetor original.

– **sqrt**: retorna um novo vetor de dados de mesma dimensï¿½es do vetor passado como parï¿½metros com os valores sendo igual ï¿½s respectivas raizes quadradas dos valores do vetor original.

– **delta**: retorna um novo vetor de dados de mesma dimensï¿½es do vetor passado como parïï¿½metros com os valores sendo igual diferenï¿½a entre posiïï¿½ï¿½es vizinhas.

4
Linguagens de Programaï¿½ï¿½o Visual

Ultimamente o uso de ferramentas de programaï¿½ï¿½o visual vem crescendo cada vez mais, principalmente em aplicaï¿½ï¿½es voltadas para ï¿½reas da engenharia como *LabVIEW*, *Simulink*, *CircuitMaker* entre outros. Mesmo tendo apresentado relativa dificuldade na resoluï¿½ï¿½o de problemas de grande porte o uso destas ferramentas em sistemas que apresentam um domï¿½nio especï¿½fico, como as citadas acima, tem se mostrado bastante interessante pois ï¿½ possï¿½vel manter-se dentro do estilo de comunicaï¿½ï¿½o do domï¿½nio utilizando artefatos visuais que reflitam necessidades particulares, diagramas e o vocabulï¿½rio especï¿½fico do domï¿½nio sem nunca precisar deixar esse estilo de comunicaï¿½ï¿½o. Alï¿½m disso, linguagens de programaï¿½ï¿½o visual costumam ser mais intuitivas e de mais fï¿½cil aprendizado do que linguagens de programaï¿½ï¿½o convencionais.

Pelos motivos citados acima, e pela calculadora de propriedades oferecer relativamente poucos recursos ao usuï¿½rio, se mostrou interessante um estudo dessas ferramentas de programaï¿½ï¿½o visual e investigar se esse tipo de recurso solucionaria o nosso problema.

Dentre os sistemas que utilizam programaï¿½ï¿½o visual, o que tem se destacado mais ï¿½ o *LabVIEW (Laboratory Virtual Instrument Engineering Workbench)* [WhNF06], que ï¿½ um ambiente de programaï¿½ï¿½o visual muito usado nos campos de aquisiï¿½ï¿½o de dados, controle de instrumentos e automaï¿½ï¿½o industrial. Por ser um instrumento amplamente usado por engenheiros em diversas ï¿½reas, e como esses sï¿½o os usuï¿½rios finais do **Geresim**, essa ferramenta se mostra uma interessante fonte de pesquisa e um possï¿½vel modelo a ser seguido, jï¿½ que ï¿½ possï¿½vel que parte dos usuï¿½rios finais tenha uma certa experiï¿½ncia com o *LabVIEW*.

A linguagem de programaï¿½ï¿½o do *LabVIEW*, chamada de G, ï¿½ formada por elementos visuais e textuais. Os elementos visuais sï¿½o basicamente pequenas caixas (que podem representar valores, operadores e chamadas de funï¿½ï¿½es) e linhas que as conectam. Elementos compostos como funï¿½ï¿½es condicionais e de repetiï¿½ï¿½o (loop) sï¿½o representados por grandes caixas retangulares cercando o corpo da composiï¿½ï¿½o.

A linguagem segue o paradigma de fluxo de dados. O princï¿½pio central do paradigma ï¿½ definir a computaï¿½ï¿½o como um conjunto de dependï¿½ncias de dados. Um programa escrito nesse paradigma define um grafo de nï¿½s operadores (funï¿½ï¿½es) unidos por arcos conectores. Os arcos sï¿½o as dependï¿½ncias de dados. Assim o paradigma de fluxo de dados expressa computaï¿½ï¿½o como uma sï¿½rie de transformaï¿½ï¿½es nos valores dos dados. Nï¿½o hï¿½ conceitos como varïï¿½veis, ponteiros e atribuiï¿½ï¿½o, o foco estï¿½ em quais funï¿½ï¿½es serï¿½o aplicadas nos valores para produzir o retorno esperado. Outra caracterï¿½stica desse paradigma ï¿½ o paralelismo. A ordem de execuï¿½ï¿½o ï¿½ baseada somente nas dependï¿½ncias de dados, o que ï¿½ relativamente similar ao design de circuitos elï¿½tricos.

Conceitualmente os programas desenvolvidos no *LabVIEW* sï¿½o compostos de funï¿½ï¿½es, que recebem o nome de instrumentos virtuais (VIs). O cï¿½digo de um VI consiste basicamente de trï¿½s elementos: terminais, nï¿½s e fios conectores. Os elementos terminais sï¿½o os valores manipulados, podendo ser valores de entrada, saï¿½da ou valores constantes. Esses elementos sï¿½o representados por pequenas caixas coloridas com a cor do tipo do terminal (por exemplo, laranja para nï¿½meros). Os nï¿½s sï¿½o operadores, chamados de funï¿½ï¿½o e nï¿½s estruturais (if-then-else, case, while etc). Operadores e chamadas de funï¿½ï¿½o sï¿½o representados por ï¿½cones com etiquetas ou imagens, e os nï¿½s estruturais sï¿½o representados por grandes caixas retangulares que delimitam a composiï¿½ï¿½o que deve ser executada dentro da estrutura. Os fios conectores sï¿½o usados para ligar terminais e nï¿½s formando expressï¿½es.

5
Inï¿½cio da implementaï¿½ï¿½o de uma VPL

Apï¿½s o estudo de VPLs e e algumas experiï¿½ncias utilizando o *LabVIEW* (na matï¿½ria Automaï¿½ï¿½o Industrial, cursada nesse perï¿½odo), foi iniciada a implementaï¿½ï¿½o de uma VPL bastante simplificada que segue o paradigma de fluxo de dados. Infelizmente nï¿½o houve tempo suficiente para desenvolver mais a ferramenta e por isso a implementaï¿½ï¿½o econtra-se em estado inicial.

Ao contrï¿½rio do que foi feito na calculadora de propriedades, inicialmente seria implementado algo mais genï¿½rico para depois ser especializado para o ambiente de pï¿½s-processamento do **Geresim**. A princï¿½pio, a linguagem seria composta apenas de blocos e fios conectores que denotariam as dependï¿½ncias de dados entre os blocos. Um bloco poderia ter mais de um dado de entrada e mais de um dado de saï¿½da. Sua compilaï¿½ï¿½o geraria um cï¿½digo **Lua**.

No cï¿½digo gerado, cada bloco seria representado por uma tabela. Essa tabela aramzenaria a funï¿½ï¿½o associada ao bloco e uma lista de objetos parï¿½metros, que seriam tabelas armazenando uma referï¿½ncia a outro bloco e o ï¿½ndice do dado de saï¿½da deste outro bloco. Todos os blocos estariam associados a uma *metatable* cujo metamï¿½todo de indexaï¿½ï¿½o executaria a funï¿½ï¿½o do bloco gerando assim os dados de saï¿½da. Porï¿½m, antes de poder executar a funï¿½ï¿½o associada a este bloco, seria preciso obter todos os parï¿½metros. E isso seria feito iterando a lista de parï¿½metros e indexando os respectivos blocos com os respectivos ï¿½ndices. Caso os dados de saï¿½da desses demais blocos nï¿½o estejam disponï¿½veis, o metamï¿½todo de indexaï¿½ï¿½o agiria entï¿½o nesses blocos. Logo, recursivamente toda a cadeia de blocos seria executada. Dessa maneira, a compilaï¿½ï¿½o seria apenas uma iteraï¿½ï¿½o sobre os fios conectores e a partir deles se geraria a lista de parï¿½metros de cada bloco.

A partir da base descrita acima, jï¿½ seria possï¿½vel implementar um bloco que desempenhe a funï¿½ï¿½o de um operador condicional e talvez seja possï¿½vel implementar um bloco de iteraï¿½ï¿½o.

No atual estï¿½gio da implementaï¿½ï¿½o ï¿½ possï¿½vel manipular os blo-

cos com o mouse e conecta-los com os fios conectores. ï¿½ possï¿½vel tambï¿½m manipular blocos jï¿½ conectados, fazendo com que os fios conectores se redesenhem se ajustando ï¿½ nova disposiï¿½ï¿½o dos blocos. Essa implementaï¿½ï¿½o foi desenvolvida em **C++**, utilizando a API grï¿½fica *OpenGL*.(**?**) A parte **Lua** ainda nï¿½o foi desenvolvida.

6
Consideraï¿½ï¿½es finais e projetos futuros

Neste trabalho foi apresentado um sistema de *script* de uma aplicaï¿½ï¿½o voltada para o pï¿½s-processamento de simulaï¿½ï¿½es de reservatï¿½rios de petrï¿½leo (**Geresim**) e a proposta e implementaï¿½ï¿½o de uma ferramenta que disponibilza parte dos recursos obtidos a partir dos *scripts* ï¿½ usuï¿½rios nï¿½o programadores. Essa ferramenta ï¿½ uma calculadora de propriedades que permite que o usuï¿½rio gere novas propriedades a partir de simples expressï¿½es aritmï¿½ticas desenvolvidas em uma interface grï¿½fica similar a uma calculadora comum. A ferramenta ainda nï¿½o foi utilizada por usuï¿½rios finais, mas jï¿½ estï¿½ integrada ao sistema e apï¿½s ser testada e aprovada por outros membros da equipe de desenvolvimento da aplicaï¿½ï¿½o criou uma boa expectativa em relaï¿½ï¿½o ao cumprimento de seu objetivo. A ferramenta estarï¿½ disponï¿½vel na prï¿½xima versï¿½o do programa, que serï¿½ lanï¿½ada em breve.

Adicionalmente, foram feitos estudos acerca de linguagens de programaï¿½ï¿½o visual. Dentre os sistemas que utilizam esse recurso, o que se mostrou mais interessante foi o *LabVIEW*, pois ï¿½ amplamente usado por usuï¿½rios que apresentam um perfil similar aos usuï¿½rios do **Geresim**. Apï¿½s esse estudo foi iniciada a implementaï¿½ï¿½o de uma VPL bastante simplificada que segue o paradigma de fluxo de dados. Infelizmente nï¿½o houve tempo suficiente para desenvolver mais a ferramenta e por isso sua finalizaï¿½ï¿½o passa a ser um projeto para o futuro.

Outro projeto futuro ï¿½ aumentar o escopo de atuaï¿½ï¿½o da calculadora incluindo as propriedades de completï¿½ï¿½o e incluindo a opï¿½ï¿½o de se trabalhar com camadas e seï¿½ï¿½es das propriedades de mapa.

Referências Bibliográficas

A
Exemplos prï¿½ticos

A.1
Cï¿½lculo da razï¿½o gas-oleo (RGO)

Expressï¿½o: QG / QO
Cï¿½digo gerado:

```
calc.GetMacroEnv()
local QG = calc.CreateProp(rf,"well","QG")
local QO = calc.CreateProp(rf,"well","QO")
local RGO = QG /QO
calc.AttachProp(rf,RGO,'RGO','well')
```

A.2
Cï¿½lculo sedimentos bï¿½sicos e ï¿½gua (BSW)

Expressï¿½o: QW /(QW + QO)
Cï¿½digo gerado:

```
calc.GetMacroEnv()
local QW = calc.CreateProp(rf,"well","QW")
local QO = calc.CreateProp(rf,"well","QO")
local BSW = QW /(QW + QO )
calc.AttachProp(rf,BSW,'BSW','well')
```

A.3
Cï¿½lculo do volume de ï¿½leo em condïï¿½ï¿½es de reservatï¿½rio

Expressï¿½o: H * PHI * SO * DX * DY
Cï¿½digo gerado:

```
calc.GetMacroEnv()
local H = calc.CreateProp(rf,"map","H")
local PHI = calc.CreateProp(rf,"map","PHI")
local SO = calc.CreateProp(rf,"map","SO")
```

Figura A.1: Exemplo de visualizaï¿½ï¿½o

```
local DX = calc.CreateProp(rf,"map","DX")
local DY = calc.CreateProp(rf,"map","DY")
local temp = H *PHI *SO *DX *DY
calc.AttachProp(rf,temp,'H *PHI *SO *DX *DY ','map')
```