SNPTEE
SEMINÁRIO NACIONAL
DE PRODUÇÃO E
TRANSMISSÃO DE
ENERGIA ELÉTRICA

GAT 13
14 a 17 Outubro de 2007
Rio de Janeiro - RJ

GRUPO IV

GRUPO DE ESTUDO DE ANÁLISE E TÉCNICAS DE SISTEMAS DE POTÊNCIA – GAT

AUTOMAÇÃO DE ESTUDOS DE GRANDE PORTE

Luiz Antonio Alves de Oliveira * **Hana Karina Salles Rubinsztejn** **Gilberto Pires de Azevedo**

Leonardo Pinto de Almeida

CEPEL – CENTRO DE PESQUISAS DE ENERGIA ELÉTRICA

RESUMO

A complexidade do sistema elétrico afeta de forma não-linear o porte dos estudos computacionais de planejamento e operação. Em geral o número de casos a serem estudados aumenta combinatorialmente com a complexidade do sistema e, como resultado, o engenheiro pode ver-se diante da necessidade de executar um grande número de casos muito parecidos entre si, o que pode levá-lo a recorrer a simplificações para evitar que o número de casos torne-se humanamente intratável. Não é raro que este processo de geração de casos para estudos de grande porte se revele tedioso, dispendioso, lento, incompleto em conseqüência das simplificações necessárias e suscetível a erros por falhas humanas.

O objetivo deste trabalho é apresentar uma ferramenta que permite o aumento da ordem de grandeza do número de casos estudados e a redução de erros por falha humana em estudos de grande porte, para que o engenheiro de estudos despenda menos tempo em tarefas repetitivas de preparação de dados e de triagem preliminar de resultados e concentre-se no projeto dos casos e na análise dos seus resultados.

Será apresentado um pacote para o uso de linguagens de script nos programas computacionais da área de sistemas elétricos do CEPEL, incluindo: (i) Biblioteca de scripts configuráveis; (ii) Módulos com funções específicas para os programas computacionais.

PALAVRAS-CHAVE

Linguagens de Script, Análise de Redes Elétricas, Planejamento e Operação de Sistemas Elétricos.

1.0 - INTRODUÇÃO

O progressivo aumento das dimensões e da complexidade do sistema elétrico reflete-se no porte dos estudos computacionais de planejamento e operação, e de forma não linear: o número de casos a serem estudados aumenta combinatorialmente à medida que o sistema elétrico cresce.

Outro fator que atua na mesma direção é a evolução gradual da qualidade da modelagem do sistema elétrico, devida principalmente ao aperfeiçoamento dos modelos internos usados nos programas e a um melhor conhecimento dos parâmetros elétricos reais dos equipamentos. Modelagens mais precisas permitem a operação mais próxima aos limites do sistema elétrico, o que para ser viabilizado sem riscos demanda estudos mais precisos. Estas exigências adicionais podem restringir o uso de simplificações que diminuiriam o número de casos a serem estudados.

(*) Rua Horácio Macedo, 354 – sala 1020 – DSE – CEP 21941-911 Cid. Universitária Rio de Janeiro, RJ – Brasil
Tel: (+55 21) 2598-6130 – Fax: (+55 21) 2598-6451 – Email: laao@cepel.br

O trabalho do engenheiro no preparo e processamento de grandes massas de casos em geral envolve, *para cada caso*:

- A preparação dos dados de entrada do caso. Em geral os casos a serem processados são muito parecidos entre si, com diferenças pontuais. Partindo de um caso montado previamente, o engenheiro tipicamente precisa realizar pequenas alterações em duas ou mais seções do arquivo ou do banco de dados para gerar um novo caso, em um processo que se repete dezenas ou centenas de vezes. Esta é uma tarefa suscetível a erros, como alterações inconsistentes entre si (mistura de dados de casos diferentes), omissão de uma alteração necessária (alteração em apenas uma seção dos dados, quando seria necessário alterar outras também), omissão de um caso inteiro, digitação incorreta de dados etc..

- Execução do caso em um programa computacional. Muitas vezes, isto requer que se passe por diversos comandos da interface do programa, o que também pode ser uma fonte de erros – por exemplo, pode-se deixar de ativar uma determinada opção de execução.

- Identificação dos resultados relevantes. Quando não dispõe de ferramentas para análise de resultados de múltiplos casos (como por exemplo a FormCEPEL), o engenheiro precisa examinar individualmente os resultados de cada caso para executar uma triagem preliminar que identifique aqueles relevantes para o estudo em questão.

Uma das abordagens para lidar com este problema consiste em criar interfaces específicas nos programas para lidar com alguns tipos de estudos envolvendo grandes massas de dados. Estas interfaces codificam e automatizam os procedimentos que seriam executados manualmente, eliminando a possibilidade de erros e melhorando a produtividade. Esta solução, porém, tem um inconveniente grave: ela não contempla a quase infinita diversidade de estudos possíveis. Pequenas alterações na metodologia do estudo podem tornar inútil a interface desenvolvida.

Outra abordagem, mais versátil, mais poderosa e de fácil implementação é a adaptação dos programas ao uso de linguagens de script. Tais linguagens vêm evoluindo rapidamente nos últimos anos, tornando-se cada vez mais poderosas, de utilização simples e viabilizando-se como alternativas reais para a automação de estudos elétricos de grande porte em programas comerciais. Tais linguagens permitem criar scripts de automação suportando condicionais e laços de repetição, além de possibilitar acesso a funções internas do programa e do sistema operacional. Permitem também a criação de bibliotecas especiais para facilitar as atividades mais comuns dos usuários.

Este artigo descreve a aplicação de uma linguagem de script à automação de estudos de grande porte nos programas computacionais do CEPEL voltados para a análise do desempenho elétrico de sistemas de potência. O programa ANATEM – Análise de Transitórios Eletromecânicos – é utilizado nos exemplos.

2.0 - LINGUAGEM DE SCRIPT

Nas linguagens de programação de terceira geração como C, Fortran e Pascal, o programador manipula os tipos de dados primitivos da linguagem - inteiro, ponto flutuante e caracter – para, a partir deles, definir tipos de dados mais complexos a serem usados na definição de objetos de dados. Nessas linguagens a implementação do algoritmo para solução do problema é feita após a definição de todas as estruturas de dados. Isto dificulta o uso da linguagem por pessoas com pouca experiência, situação em que em geral o desenvolvimento se torna lento e suscetível a falhas.

As linguagens de script permitem minimizar esses problemas. Nelas, o programador não necessita definir tipos de dados complexos, sendo por isto consideradas *linguagens fracamente tipadas*. Toda a funcionalidade já está presente na própria linguagem ou disponível em bibliotecas.

Segundo Lutz Prechelt (1) um dos objetivos das linguagens de script é combinar componentes desenvolvidos em outras linguagens. Por esta razão são também chamadas de *linguagens de integração de sistemas* (*system integration languages*) ou *linguagens de colagem* (*glue languages*).

Uma das características que facilita esta integração é que as linguagens de script são interpretadas. Isto, apesar de gerar uma menor eficiência computacional em relação às linguagens compiladas (o que não chega a ser um problema no contexto de uso proposto), por outro lado favorece a portabilidade e reutilização de código. Mas, para os objetivos deste trabalho, a principal vantagem das linguagens de script é possibilitar a alteração do código sem necessidade de compilação. O usuário pode fazer adaptações para atender às suas necessidades específicas rapidamente e sem precisar compilar trechos do programa. Se as interfaces apropriadas forem desenvolvidas, as linguagens script podem inclusive interagir com métodos do programa, chamando-os ou sendo chamadas por eles. Tomados os indispensáveis cuidados, isto permite que o usuário desenvolva trechos de

código que interajam com o programa para, por exemplo, coletar e processar automaticamente resultados em estudos de grande porte da forma que lhe for mais conveniente.

A maioria das linguagens de script são *open-source* e estão disponíveis gratuitamente. Em geral, são desenvolvidas por um grande número de pessoas e dispõem de uma boa documentação. Algumas das linguagens de script mais utilizadas são Perl, Tk/Tcl, Python e Lua. A primeira é muito utilizada em aplicações Web; Tk/Tcl é largamente utilizada em empresas de telecomunicações; Python é muito utilizada em desenvolvimento de sistemas e em aplicações comerciais; já Lua é utilizada em jogos e aplicações para celulares, dentre outros usos.

Python (2) foi criada em 1990 e é atualmente mantida por uma comunidade virtual e garantida pelo CNRI (Corporation for National Research Initiatives). Neste trabalho optou-se pela linguagem Python por possuir orientação a objeto e por sua sintaxe simples, e também por ser portável e extensível. No entanto, a estrutura desenvolvida é compatível também com outras linguagens.

3.0 - DESCRIÇÃO DO PROBLEMA

Um arquivo de caso do ANATEM é um arquivo de texto simples que pode conter centenas de mnemônicos próprios. Um mnemônico define uma seção do arquivo na qual o engenheiro especifica dados para simulação. A figura 1 apresenta trechos de um arquivo de caso ANATEM típico.

```
( DADOS DE EVENTOS
DEVT
(Tp) (Tmp) (El)(Pa)Nc(Ex)(% )(ABS )MqUn(Bl)P( Rc )( Xc )( Bc )
MDSH  0.20  101              -5500.
MDSH  0.30  101              +5500.
ABCI  0.30  101 100 1
9999
( DADOS DE MAQUINAS E ASSOCIACAO DAS MAQUINAS AOS CONTROLES
DMAQ
(No) O Mq (P) (Q) Un (Mg) (Mt)u(Mv)u(Me)u(Xvd)(Nb)
  10   10 100 100  1  100  100u 140u 170u       10          ANGRA-1--1GR
(INICIO_BLOCO
  11   10 100 100  1  101  101u 141u 171u       11          ANGRA-2--1GR
  12   10 100 100  6  103  103u 143u            12          LCBARRET-6GR
  14   10 100 100  1  105  105u 145u            14          FUNIL-1--1GR
(FIM_BLOCO
9999
```

FIGURA 1 – Trecho arquivo de caso ANATEM

Na figura 1 os mnemônicos DEVT e DMAQ definem respectivamente as seções para definição de eventos e dados de máquina; ambas são finalizadas com o mnemônico 9999.

Nos estudos de grande porte tem-se grandes massas de casos de teste, que são representadas por um número também grande de arquivos de caso, sendo que muitos destes arquivos diferem entre si apenas por poucas linhas. Isto obriga o engenheiro de estudos a editar muitos arquivos e implementar cuidadosamente, em cada um, pequenas modificações. Este processo é suscetível a erros, além de consumir tempo em atividades repetitivas.

O objetivo é disponibilizar para o engenheiro uma ferramenta com a qual ele possa, de forma rápida e fácil:

- especificar todas as diferenças entre os diversos arquivos de caso;
- submetê-los ordenadamente ao programa ANATEM; e
- identificar e acessar os resultados de cada arquivo de estudo.

4.0 - O PROCESSO DE AUTOMAÇÃO

O processo de automação baseia-se em um *arquivo modelo*, referente ao arquivo de entrada de um determinado programa. Este é o arquivo que servirá de modelo para a geração de todos os arquivos de casos a serem usados no estudo, a partir de alterações em seções bem definidas. Estas alterações são especificadas em separado, assim como a lógica que será usada para gerar os arquivos de caso a partir do arquivo modelo e das alterações.

Na solução desenvolvida é fornecido um módulo - denominado **dseStudyFiles** – que disponibiliza um conjunto de funções e classes que realizam o trabalho de montagem dos arquivos de caso a partir da lógica especificada pelo engenheiro. Este módulo, desenvolvido em C, contém funções que interagem com a linguagem Python mas não exclui a possibilidade de uso de algumas outras linguagens de script.

A figura 2 apresenta um script Python simples para criação de casos para o programa ANATEM.

```
import dseStudyFiles

caso = dseStudyFiles.Case("modelo.stb","alteracoes.xml", "ANATEM")
caso.replaceCode("DEVT","1")
caso.save("caso1.stb")

caso.replaceCode("DEVT","2")
caso.replaceBlock("INICIO_BLOCO","FIM_BLOCO", "BLOCO","1")
caso.save("caso2.stb")
```

FIGURA 2: Exemplo de uso do módulo **dseStudyFiles**

O comando **import dseStudyFiles** ativa o módulo **dseStudyFiles** e o torna disponível para uso imediato. A classe **Case**, que fornece grande parte das funcionalidades do módulo, é instanciada neste exemplo através do comando:

```
caso = dseStudyFiles.Case("modelo.stb","alteracoes.xml", "ANATEM")
```

Instancia-se assim um objeto da classe **Case**, referenciado pela variável **caso**. O construtor da classe necessita de três parâmetros: "**modelo.stb**" identifica o arquivo modelo; "**alteracoes.xml**" identifica o arquivo no formato XML (3) onde estão especificadas e identificadas as alterações a serem feitas no arquivo modelo (pequenos trechos de seções definidas pelos respectivos mnemônicos e também blocos de texto, sem no entanto incluir a lógica a ser usada na geração dos casos); finalmente, "**ANATEM**" especifica que os arquivos de caso a serem gerados destinam-se ao programa computacional ANATEM.

4.1 Arquivo modelo

O arquivo modelo é um arquivo de caso válido, que é utilizado como modelo no processo de automação. Todos os arquivos de casos gerados pelo objeto **caso** derivarão do arquivo modelo. O arquivo modelo, além dos mnemônicos próprios, pode conter linhas de comentários para definição de blocos para substituição. Na figura 1, que apresenta um exemplo de arquivo de modelo, um bloco pelos comentários é delimitado por **INICIO_BLOCO** e **FIM_BLOCO,** os quais definem uma seção do arquivo que poderá ser modificada ou substituída.

4.2 Arquivo de alterações

É um arquivo do tipo XML que contém os dados necessários para a modificação do arquivo modelo durante a criação dos novos arquivos de caso, sem no entanto especificar a lógica a ser usada. O padrão XML, amplamente utilizado atualmente, é aberto e provê uma representação estruturada de dados, permitindo codificá-los para uma variedade de aplicações. Um arquivo XML é de fácil edição e interpretação pelo usuário.

A figura 3 apresenta o arquivo "alteracoes.xml" utilizado no exemplo da figura 2.

```
<!--  Definição da tag DEVT com identificação 1 -->
<DEVT ID="1">
DEVT
(Tp) (Tmp) (El)(Pa)Nc(Ex)(% )(ABS )MqUn(Bl)P( Rc )( Xc )( Bc )
MDSH  0.20  102              -5500.
MDSH  0.30  102              +5500.
ABCI  0.30  102 100 1
9999
</DEVT>

<DEVT ID="2">
DEVT IMPR
(Tp) (Tmp) (El)(Pa)Nc(Ex)(% )(ABS )MqUn(Bl)P( Rc )( Xc )( Bc )
MDSH  0.20  140              -2500.
MDSH  0.30  140              +2500.
ABCI  0.30  101 100 1
9999
</DEVT>

<BLOCO ID="1">
  11   10 100 100  2  101  101u 141u 171u       11          ANGRA-2--1GR
</BLOCO>
```

FIGURA 3: Arquivo alteracoes.xml utilizado no script da figura 2

Os arquivos permitem a especificação de dois tipos de alterações:

- Seções inteiras do arquivo modelo, com base em mnemônicos. Neste caso deve ser especificado no arquivo de alterações todo o conteúdo da seção correspondente a um mnemônico a ser substituído.
- Blocos livremente definidos, sem correspondência com mnemônicos do programa. Neste caso basta definir as linhas a serem substituídas ou inseridas.

No arquivo "alteracoes.xml" são modificados o tempo de eliminação da falta e a potência de curto-circuito monofásico, para por exemplo sistematizar o estudo destes parâmetros para verificar sob quais condições de falta o sistema permanece estável. São definidos um bloco iniciado pela tag XML **<BLOCO ID="1">** e finalizado por **</BLOCO>** e duas seções **DEVT,** uma iniciada pela tag XML **<DEVT ID="1">** e outra iniciada por **<DEVT ID="2">**, ambas finalizadas por **</DEVT>**. É importante ressaltar que ambas as seções apresentam valores de especificação de eventos distintos (grafados em negrito) e são identificadas por **ID="1"** e **ID="2"** respectivamente (na identificação são permitidos valores numéricos ou alfanuméricos). O nome da tag XML deve conter o mesmo mnemônico da seção especificada pela mesma. Já para especificação de blocos o nome da tag é definido pelo usuário, assim como o identificador da tag.

4.3 Métodos

O módulo **dseStudyFiles** fornece um conjunto de métodos da classe **Case** que facilitam a manipulação do arquivo modelo e a criação de arquivos de casos:

- **replaceCode** - O método **replaceCode** realiza a substituição, no arquivo modelo, da seção especificada pelos parâmetros. Assim o comando

  ```
  caso.replaceCode("DEVT","1")
  ```

  realiza a substituição da seção DEVT existente no arquivo modelo.stb pela seção delimitada pela tag DEVT e identificada por ID="1" no arquivo alteracoes.xml. Na geração de um caso é possível realizar tantas chamadas quantas forem as necessárias ao método replaceCode, desde que em seções (mnemônicos) distintas.

- **replaceBlock** – Este método substitui o bloco definido no arquivo modelo por "**(INICIO_BLOCO**" e "**(FIM_BLOCO**" pelo bloco definido no arquivo alteracoes.xml pela tag **<BLOCO ID="1">**:

  ```
  caso.replaceBlock("INICIO_BLOCO","FIM_BLOCO", "BLOCO","1")
  ```

- **save** - Terminadas as modificações é possível gravá-las em arquivo utilizando o método **save**:

  ```
  caso.save("caso1.stb")
  ```

Assim todas as modificações no arquivo modelo realizadas pelos métodos da classe **`Case`** são gravados no arquivo `caso1.stb`.

O módulo **`dseStudyFiles`** também provê alguns métodos auxiliares para criação e execução de casos em linha de comando. Assim o script pode conter os seguintes comandos:

- `caso.createBat("exec.bat")`
- `dseStudyFiles.execBat("exec.bat")`

que criam e executam o arquivo de lote `exec.bat`. Este arquivo de lote executa o programa ANATEM e passa como parâmetro cada arquivo gravado pela classe **`Case`**.

5.0 - Caso de Uso

De forma a exemplificar a utilização de linguagens de script em um estudo de grande porte, foi desenvolvido um script em Python para geração de casos para os estudos de análise do sistema de integração dos aproveitamentos hidrelétricos do rio Madeira.

Este script foi escrito para a análise de três alternativas de transmissão. Para cada alternativa são estudados dois cenários hidrológicos, um de alta e outro de baixa hidraulicidade. Cada cenário hidrológico ainda é dividido em dois patamares de carga distintos: pesada e leve. Através da combinação dos cenários descritos são formados 12 casos base para um único ano de análise. Esta combinação é apresentada na figura 4. Contudo, se considerarmos agora um segundo ano para análise, o número de casos base dobraria, e assim sucessivamente em função do aumento do número de anos em análise (4).

Na figura 4 é mostrado que para cada caso base analisado são simuladas pelo menos 28 contingências, formando um total de 336 simulações. Portanto, seria necessário que o engenheiro de estudos preparasse 336 arquivos de entrada para o Programa ANATEM, um processo tedioso, lento e suscetível a erros. Por exemplo, para um mesmo caso base (Alternativa I – Alta hidraulicidade – Carga pesada) são necessárias 28 simulações, onde cada simulação difere da outra apenas pela contingência aplicada, ou seja, apenas um código do arquivo de entrada é modificado.

Alternativa	Cenário Hidrológico	Cenário de Carga	No. de Contingências
Alternativa I	Alto	Pesada (AP)	28
		Leve (AL)	28
	Baixo	Pesada (BP)	28
		Leve (BL)	28
Alternativa II	Alto	Pesada (AP)	28
		Leve (AL)	28
	Baixo	Pesada (BP)	28
		Leve (BL)	28
Alternativa III	Alto	Pesada (AP)	28
		Leve (AL)	28
	Baixo	Pesada (BP)	28
		Leve (BL)	28
Total			336

FIGURA 4: Combinação dos cenários analisados

De maneira a automatizar o processo de preparação destes 336 arquivos foi criado o script em Python apresentado na figura 5. Com isso o engenheiro de estudos ao invés de preparar todos os arquivos, deve se preocupar somente com um arquivo “stb” modelo e com o arquivo “xml” nos quais estarão listadas as diferenças entre os diversos cenários estudados. Estas diferenças podem ser, por exemplo: dados de modelo de carga, dados de eventos, dados de plotagem, dados de modelo de máquinas, etc..

```
import dseStudyFiles

caso =  dseStudyFiles.Case("BTB#00.STB","DEVT.XML", "ANATEM");

a = ['AP', 'AL', 'BP', 'BL']
c = ['ALT1', 'ALT2', 'ALT3']

for j in c:
       caso.replaceCode("DPLT",j)
       for x in a:
       b = j + x
         caso.replaceBlock('StartArqv', 'EndArqv', 'arqvhist', b)
         caso.replaceCode("DMAQ",x)
              for i in range(1,29):
                           caso.replaceCode("DEVT",i)
                           caso.save(c + "_" + a + "_" + str(i) + ".stb")

caso.createBat("exec.bat")
```

FIGURA 5: Script Python para geração dos casos de estudos do rio Madeira

Os casos são gerados seguindo a tabela da figura 4, e ao término da execução do script terão sido gerados todos os 336 casos necessários.

Para isto são realizadas 3 substituições no mnemônico **DPLT** utilizando os identificadores **ALT1**, **ALT2** e **ALT3** na linha **caso.replaceCode("DPLT",j).** Para cada **DPLT** é feito o carregamento do arquivo histório identificado pela concatenação do **DPLT** correspondente com **AP**, **AL**, **BP** e **BL,** na linha **caso.replaceBlock('StartArqv', 'EndArqv', 'arqvhist', b);** e a substituição no mnemônico **DMAQ** utilizando o identificador do **DPLT** correspondente na linha **caso.replaceCode("DMAQ",x).** Para cada **DMAQ** é feita a substituição do mnemônico **DEVT** com identificador variando de 1 a 28, na linha **caso.replaceCode("DEVT",i)**. Desta forma, o primeiro arquivo caso gerado pelo script seria **ALT1_AP_1.stb** , referente à alternativa 1 com cenário de alta hidraulicidade e de carga pesada para a contingência 1 e assim sucessivamente.

6.0 - CONCLUSÃO

Este trabalho apresenta os estágios iniciais de desenvolvimento de uma ferramenta que visa facilitar a realização de estudos de grande porte em programas de planejamento e operação elétrica desenvolvidos pelo CEPEL, baseada no uso de linguagens de script. A ferramenta mostrou-se capaz de gerar grandes quantidades de arquivos de dados com notável economia de tempo para o engenheiro de estudos, permitindo que os esforços sejam redirecionados para a análise de resultados e eliminando a possibilidade de erros casuais na montagem dos arquivos.

A flexibilidade proporcionada pelo uso da linguagem de script Python, aliada aos recursos providos pelo módulo **dseStudyFiles**, permitiu a montagem automática dos arquivos conforme as necessidades específicas de cada estudo e sem exigir profundos conhecimentos da linguagem.

Está sendo avaliada a incorporação destes recursos de automação na interface gráfica dos programas, para que sejam usados facilmente a partir dessas interfaces. Está em exame também a possibilidade de chamada de scripts definidos pelo usuário a partir do código do programa, dando aos scripts acesso a uma parte dos dados e resultados para viabilizar a geração sob medida de relatórios ou novos arquivos de saída. Estes recursos, aliados a uma ampliação das funcionalidades do módulo **dseStudyFiles**, poderão permitir não apenas uma realização mais rápida (e menos suscetível a erros humanos) de estudos de grande porte como também abrem a perspectiva de ampliação do porte desses estudos quando conveniente.

7.0 - REFERÊNCIAS BIBLIOGRÁFICAS

(1) Lutz Prechelt: “Are scripting languages any good? A validation of Perl, Python, Rexx, and Tcl against C, C++, and Java”. Advances in Computers 57: 207-271 (2003)

(2) THE PYTHON Programming Language. [S.l.] : Python Software Foudation, c1990-2006. Disponível em: <www.python.org>. Acesso em 16 mar. 2007.

(3) EXTENSIVE Markup Language (XML). [S.l.] : W3C, c1996-2003. Disponível em: <www.w3.org/XML>. Acesso em 16 mar. 2007.

(4) CRITÉRIOS para estudos, submodulo 23 dos procedimentos de rede do Operador Nacional do Sistema Elétrico (ONS) ; Revisão No. 1; aprovado pela ANEEL em 25/03/2002.