Ministério da Ciência, Tecnologia e Inovação
Centro Regional Sul de Pesquisas Espaciais
Laboratório de Computação para Clima Espacial

Relatório Final

Projeto: Laboratório Virtual Para Clima Espacial
Bolsista: Luiz Felipe Nascimento Schleder
Orientador: Adriano Petry

1. OBJETIVOS

O objetivo deste projeto é disponibilizar dados ao software SUPIM-DAVS, permitindo que este execute assimilação de dados e obtenha simulações da ionosfera mais precisas. A assimilação de dados é o processo de integrar a um modelo matemático dados de ionossondas, satélites, radares, estações locais, etc. Estes dados representam observações reais, portanto são utilizados para melhorar a qualidade das simulações, evitando que estas se desviem muito da realidade. Em termos práticos, para uma assimilação ser realizada, o sistema precisa de dados observacionais vindos de diversas fontes, completos, com data e hora específicas.

2. ATIVIDADES DESENVOLVIDAS

Para atingir os objetivos deste projeto, foi decidido que seria implementado um cliente de web services, seguindo padrões impostos pelo domínio em questão: o formato de dados que o SUPIM-DAVS pode ler; os servidores que disponibilizam dados; as tecnologias utilizadas.

Web services são APIs criadas segundo certos padrões que, geralmente através do protocolo de comunicação HTTP, permitem o compartilhamento de informações entre diferentes sistemas, não importando a linguagem ou a arquitetura em que foram desenvolvidos.

Neste projeto nós seguimos o padrão REST, ou Representational State Transfer, no qual os métodos GET, POST, PUT, DELETE, são disponibilizados pelo servidor e acessados pelo cliente através de URIs específicas.

Após os primeiros meses de estudos, foi criado um protótipo de cliente de web services que recebe dados, os analisa e os envia para o SUPIM-DAVS, após convertê-los ao formato AMAP. Também foi desenvolvido um módulo de ordenação e correção de arquivos AMAP pré-existentes, pois este recurso era necessário em outras partes do sistema.

Após a finalização deste protótipo, exaustivos testes foram realizados, comprovando a eficácia do modelo de desenvolvimento planejado. Na fase de integração destes módulos com o sistema SUPIM-DAVS, o protótipo foi reprogramado na linguagem Java (originalmente havia sido desenvolvido na linguagem Python), para aumentar a velocidade de execução e facilitar futuras manutenções, posto que as outras partes do sistema haviam sido criadas ou em Java ou em C++.

3. DETALHES DAS IMPLEMENTAÇÕES

3.1 Protótipo

Após a fase de estudos, optou-se por efetuar a fase de programação utilizando principios derivados metodologias ágeis de desenvolvimento. Muito pode-se falar sobre metodologias ágeis, mas o princípio utilizado mais importante foi o de realizar entregas frequentes de software funcional – ou seja, ao invés de implementar todas as funcionalidades antes da entrega final, desenvolver os software através de pequenos módulos auto-contidos, realizando testes e utilizando cada módulo imediatamente após o seu desenvolvimento. Este é o chamado *desenvolvimento incremental*.

Esta abordagem possui muitas vantagens. Uma delas é que podemos validar se um recurso realmente é necessário – e, se sim, podemos testar qual o melhor algoritmo para realiza-lo. Outra vantagem é a maior adaptabilidade a eventuais alterações nos requisitos iniciais do programa. Frequentemente, conforme se consegue mais informações sobre um projeto ou o domínio no qual se trabalha, percebe-se que o modelo inicial não é o ideal para resolver um problema. Quando se desenvolve um programa de forma incremental, a possibilidade de se precisar alterar ou descartar grandes quantidades de código é menor.

Para adaptar-se melhor a esta metodologia, o protótipo foi desenvolvido na linguagem Python. Ela é multi-paradigmas, possui sintaxe simples e legível, além de ser uma linguagem de alto nível, ou seja, permite que os requisitos sejam implementados com poucas linhas de código.

O primeiro recurso implementado foi o de conectar-se a um servidor informado previamente, receber como entrada uma data e horário, fazer download dos dados e convertê-los para o formato que AMAP.

```
def chamar_url(url):
    try:
        handler = urlopen(url)
    except URLError as e:
        print "\nErro ao abrir a url", url
        print e.reason
```

Figura 1: Trecho inicial da função que recebe os dados do servidor.

```
def gerar_matriz(dados):
    matriz = [linha.split(';') for linha in dados.split('\n')]
    return matriz
```

Figura 2: Função que converte os dados recebidos em uma matriz.

```
def gerar_amap(matrix, lat_initial=0.0, long_initial=0.0, incr=0.5):
    filename = "{0:0>2}".format(header_date.hour) + '_' +
               "{0:0>2}".format(header_date.minute) + '_' + \
        "{0:0>2}".format(header_date.month) +
        "{0:0>2}".format(header_date.day) + '_' + \
        "{0:0>2}".format(header_date.year) + '.amap'
    with open(filename, 'w') as f:
        f.write('DATE/TIME: ' + "{0:0>2}".format(header_date.year) + 3 * ' '
      + "{0:0>2}".format(header_date.month) \
      + 3 * ' ' + "{0:0>2}".format(header_date.day) + 3 * ' '
      + "{0:0>2}".format(header_date.hour) \
            + 3 *' ' + "{0:0>2}".format(header_date.minute)+'\n')
        f.write('LONGITUDE / LATITUDE / TEC [x10^16/m^2] \n')
        latitude = lat_initial
        longitude = long_initial
        for row in matrix:
            col = matrix[:, matrix.index(row)]
            for value in col:
                if value == '-1.0000':
                    value = '999.000'
                longitude = "%.2f" % longitude
                latitude = "%.2f" % latitude
                f.write(longitude + 3 * ' '+ latitude + 3 * ' ' + value +'\n')
                latitude = float(latitude) + incr
                longitude = float(longitude)
            longitude += incr
            latitude = lat_initial
```

Figura 3: Função que converte a matriz em um arquivo do formato AMAP.

```
DATE/TIME: 2014   01   01   11   00
LONGITUDE / LATITUDE / TEC [x10^16/m^2]
270.00   -60.00   999.000
270.00   -59.50   999.000
270.00   -59.00   999.000
270.00   -58.50   999.000
270.00   -58.00   999.000
270.00   -57.50   999.000
270.00   -57.00   999.000
270.00   -56.50   999.000
270.00   -56.00   999.000
270.00   -55.50   999.000
```

Figura 4: Trecho inicial de um arquivo no formato AMAP.

Assim como foi previsto, outros recursos além dos planejados mostraram-se necessários, como funções de checagem da integridade dos dados ou ordenar as latitudes e longitudes corretamente. Alguns exemplos:

```
# regex para capturar números que podem ser int ou float, com/sem sinal:
regex = r"""
        [-+]? # sinal opcional
        (?: (?: \d* \. \d+ ) | (?: \d+ \.? ) )
        (?: [Ee] [+-]? \d+ ) ?
        """
```

Figura 5: Expressão regular para ler números em um arquivo AMAP.

```
def existe_repeticao(matriz):
    for linha in matriz:
        if matriz.count(linha) > 1:
            return linha, matriz.index(linha)
    return False
```

Figura 6: Função que busca por linhas repetidas.

3.2 Versão Final

Após o desenvolvimento do protótipo, este foi testado e suas saídas validadas. O passo seguinte seria integra-lo ao SUPIM-DAVS, tornando-o uma parte do sistema. Conforme já foi mencionado, foi realizada uma “tradução” do protótipo para Java, o que compreende não apenas uma alteração de sintaxe, mas de paradigmas e organização do programa. A linguagem Java possui melhor desempenho computacional do que Python, e esta possui desenvolvimento mais rápido do que Java. Portanto, acreditamos que a metodologia de desenvolvimento utilizada permitiu que se encontrasse rapidamente a solução, e que esta fosse executada da forma mais eficiente na versão final do sistema.

```
public class Ordenador {

	private String[][] matriz;

	private ArrayList<String[]> resposta = new ArrayList<String[]>();


	private void ordenarSegundaCol() {

		String chaveAtual = "";

		ArrayList<String[]> grupo = new ArrayList<String[]>();


		// quando encontra na primeira col um item diferente,

		// atualiza a chave e separa estes itens em grupo.

		// este grupo é ordenado e adicionado ao ArrayList resposta.

		for (int linha = 0; linha < matriz.length ; linha++) {

			if (!matriz[linha][0].equals(chaveAtual)) {

				chaveAtual = matriz[linha][0];

				grupo = gerarGrupo(chaveAtual);

				grupo = ordenarGrupo(grupo);

				addGrupoResposta(grupo);

			}

		}

	}
```

Figura 7: Trecho de uma das classes da versão final do programa.

4. RESULTADOS OBTIDOS

A versão final do programa foi integrada e está rodando diariamente. A assimilação de dados realizada pelo SUPIM-DAVS está mais precisa, devido aos dados extras que o sistema recebe. Como trabalho futuro, pode-se utilizar o programa desenvolvido para integrar o SUPIM-DAVS a novas fontes de dados, conforme estas estiverem disponíveis.

Santa Maria, 15 de Julho de 2014.

Luiz Felipe Nascimento Schleder Bolsista	Dr. Adriano Petry Orientador