



User Guide

Carlos Magno
Abreu

1. O processador de Workflows Sagitarii

Descrever o Sagitarii. Projeto em andamento. Funcionalidades não implementadas. Tese do Prof. Ogasawara.

Login admin / admin

2. O processamento de tarefas em nós de execução

Descrever o Teapot Cluster. Processamento distribuído. Tela de monitoramento. Main Cluster.

3. Definindo um workflow

A primeira ação, e aparentemente a mais óbvia, é entender profundamente o trabalho a ser executado. Cada passo do workflow deverá estar bem definido, o objetivo bem estabelecido e o formato dos dados de entrada, intermediários e de saída de cada atividade deverá estar bem delineado. No workflow de exemplo, o objetivo é filtrar dados eleitorais das eleições para presidente do ano de 2014 (primeiro turno) e obter arquivos compactados com as informações sobre a totalização dos votos válidos, a totalização dos eleitores aptos a votar naquele estado, a totalização dos eleitores que realmente votaram naquele estado a UF do estado e o sexo dos eleitores. Os arquivos deverão ser separados por UF (estado) e sexo dos eleitores. Vale ressaltar que o workflow proposto serve meramente para ilustrar as funcionalidades do Sagitarii e não possui nenhuma aplicação prática. Também será gerado um gráfico para ilustrar a integração com o “R” [RPROJECT] e uma atividade SPLIT (descompactação) para ilustrar o download de arquivos pelos nós de processamento.

Para criar um novo workflow no Sagitarii, clique no ícone “Home”. O sistema exibirá a tela de gerenciamento de workflows. Clique em “New Workflow”, preencha os campos com uma etiqueta (uma identificação curta) e uma descrição do workflow. Clique em “Send” e o novo workflow estará criado. A figura 1 ilustra a tela de gerenciamento de workflows com o workflow de exemplo já criado.

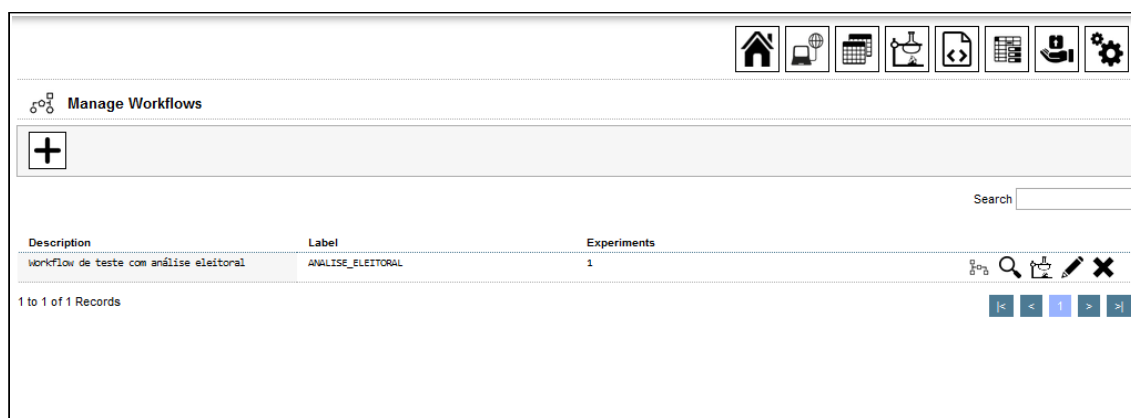


Figura 1. Tela de gerenciamento de Workflows

4. Origem dos dados

Os dados utilizados neste exemplo foram obtidos no site do Superior Tribunal Eleitoral (<http://www.tse.jus.br/hotSites/pesquisas-eleitorais/>). O workflow de exemplo utilizará

duas tabelas de dados, sendo a primeira “Detalhe da apuração por município e zona”, que armazena os quantitativos dos votos por zona eleitoral e a segunda “Eleitorado 2014”, que armazena dados dos eleitores, como sexo e grau de escolaridade. O site do TSE fornece as tabelas em formato CSV e compactadas em arquivo ZIP.

5. Preparando as tabelas para processamento

O Sagitarii, no estágio atual, trabalha exclusivamente em banco de dados, portanto, será necessário criar as tabelas que serão utilizadas pelo workflow, incluindo as que irão conter os dados iniciais, que deverão ser importados dos arquivos no formato CSV que foram baixados do site do STE. Como o Sagitarii exige que todas as relações de entrada e saída de dados estejam prontas antes da execução do workflow, é necessário um profundo entendimento do trabalho a ser executado. Caso uma tabela não esteja com sua estrutura (esquema) perfeitamente em acordo com os dados a serem produzidos ou consumidos, poderá ocorrer desde erros na execução do experimento (o que impedirá sua conclusão) até um experimento com informações incorretas (o experimento é concluído, mas produzirá dados incorretos). O apêndice “A” contém as estruturas das tabelas utilizadas neste exemplo, bem como a descrição da sua finalidade. É aconselhável documentar as estruturas das tabelas antes de iniciar sua criação no Sagitarii.

O Sagitarii permite que as tabelas que serão utilizadas no workflow possam ser criadas diretamente no sistema. Clique no ícone “Custom Tables”. O sistema exibirá a tela de gerenciamento de tabelas do usuário. Clique em “New Relation”. Preencha os campos com um nome (as mesmas exigências para nomes de tabelas em linguagem DDL SQL) e uma descrição. Clique no ícone “New Field” (à direita da tela) quantas vezes forem necessárias a fim de cadastrar todos os atributos da nova tabela. Preencha os campos com os nomes dos atributos e selecione o tipo apropriado conforme o tipo de dado que será armazenado. O Sagitarii comporta 4 tipos básicos de atributos (integer, string, date e float). Clique em “Send” para criar a nova tabela. O Sagitarii irá criar a tabela no banco de dados e armazenar suas informações para posterior consulta. A figura 2 ilustra a tela de gerenciamento de tabelas do usuário com algumas tabelas já criadas. Os atributos `id_experiment`, `id_custom`, `id_activity`, `index_number` e `taskid` são criados automaticamente pelo Sagitarii para controle interno.

Manage Relations

Form fields:

- Name:
- Description:
- Field: Integer
- Field: String

Buttons: Cancel, Send

Search:

Name	Description	Schema
sel_001	Seleção da tabela de junção	CREATE TABLE sel_001(id_custom serial NOT NULL, id_experiment integer, id_activity integer, index_number integer, taskid character varying(20), numero_zona integer, qtd_aptos integer, qtd_comparecimento integer, qtd_votos_nominais integer, uf character varying(250), faixa_etaria character varying(250), qtd_eleitores_no_perfil integer, sexo character varying(250), nr_zona integer, CONSTRAINT sel_001_pkey PRIMARY KEY (id_custom), CONSTRAINT sel_001_fkex FOREIGN KEY (id_experiment) REFERENCES experiments (id_experiment), CONSTRAINT sel_001_fkac FOREIGN KEY (id_activity) REFERENCES activities (id_activity))
reduce_001	Tabela com os resultados do workflow	CREATE TABLE reduce_001(id_custom serial NOT NULL, id_experiment integer, id_activity integer, index_number integer, taskid character varying(20), filename character varying(250), num_registers integer, CONSTRAINT reduce_001_pkey PRIMARY KEY (id_custom), CONSTRAINT reduce_001_fkex FOREIGN KEY (id_experiment) REFERENCES experiments (id_experiment), CONSTRAINT reduce_001_fkac FOREIGN KEY (id_activity) REFERENCES activities (id_activity))
perfil_eleitorado_proj	Projeção da tabela de perfil do eleitorado	CREATE TABLE perfil_eleitorado_proj(id_custom serial NOT NULL, id_experiment integer, id_activity integer, index_number integer, taskid character varying(20), uf character varying(250), sexo character varying(250), faixa_etaria character varying(250), nr_zona integer, qtd_eleitores_no_perfil integer, CONSTRAINT perfil_eleitorado_proj_pkey PRIMARY KEY (id_custom), CONSTRAINT perfil_eleitorado_proj_fkex FOREIGN KEY (id_experiment) REFERENCES experiments (id_experiment), CONSTRAINT perfil_eleitorado_proj_fkac FOREIGN KEY (id_activity) REFERENCES activities (id_activity))

Figura 2. Tela de gerenciamento de tabelas do usuário

6. Atividades de um workflow

Cada passo da execução do workflow é chamado de atividade. Uma atividade consome dados e produz dados. Existem tipos diferentes de atividades dependendo de como os dados serão processados. Basicamente as atividades podem modificar uma informação (MAP), selecionar informações (SELECT), agrupar informações (REDUCE) ou expandir informações (SPLIT) [OGASAWARA, 2011]. As atividades de seleção são executadas diretamente pelo servidor, pois trabalham exclusivamente executando scripts SQL, selecionando dados de tabelas de entrada e inserindo diretamente em uma tabela de saída. As demais atividades são de responsabilidade do usuário definir o que será executado. Normalmente, as máquinas que irão executar as tarefas do workflow possuem softwares específicos para esta tarefa. A fim de manter uma padronização na comunicação de dados entre o servidor Sagitarii e os nós que executarão as tarefas e manter o Sagitarii flexível o suficiente, criou-se o conceito de “wrapper”. Um wrapper é um software, normalmente feito em java, que recebe os dados no formato utilizado pelo Sagitarii (CSV), executa o software específico da tarefa desejada, passando estes dados como parâmetro, recebe os dados desta tarefa, transforma o resultado no formato utilizado pelo Sagitarii e encaminha-o ao servidor para armazenamento. Por exemplo: supondo que uma tarefa produza uma série de arquivos em disco e envie ao Sagitarii os nomes e localizações destes arquivos. Uma segunda tarefa deve compactar estes arquivos usando um critério qualquer e enviar ao Sagitarii os arquivos compactados. Para tanto, deve-se usar um software de compactação existente nos computadores que estão processando as tarefas, por exemplo: “*tar -zcf resultado.tar arquivo1.csv arquivo2.csv arquivo3.png*”.

O wrapper receberá os nomes dos arquivos no formato CSV, vindos do Sagitarii, e se encarregará de chamar o comando tar com os parâmetros corretos, aguardará a conclusão da compactação e enviará o resultado de volta ao Sagitarii. Em resumo, um

wrapper é um “tradutor” entre o que o Sagitarii precisa fazer e o que há disponível nos nós de execução para executar a tarefa. É possível existir wrappers que executem diretamente uma tarefa, sem chamar nenhum programa externo, como seria o caso, no exemplo anterior, se o próprio wrapper implementasse um algoritmo de compactação.

O Sagitarii exige que os wrappers necessários ao workflow já tenham sido criados. Sendo assim, antes de prosseguir com a criação do workflow, é imperativo codificar um wrapper para cada atividade que será executada nos nós de processamento. Para o workflow de exemplo, foram codificados quatro wrappers, conforme tabela 1. Estes wrappers já implementam o código necessário para realizar a atividade, portanto não chamam programas externos. Existe ainda um wrapper específico para executar script “R”, mas este é genérico o suficiente para cobrir qualquer experimento que utilize o processador “R”. Recomenda-se não apagar este wrapper, pois faz parte do próprio Sagitarii.

Tabela 1. Wrappers para as atividades do workflow de exemplo

perfil_eleitorado.jar	Executa a projeção das colunas da tabela “perfil_eleitorado” na tabela “perfil_eleitorado_proj”. É executado pela atividade “PROJ_PER_ELEIT”.
detalhe_vot_zona.jar	Executa a projeção das colunas da tabela “detalhe_votação_zona” na tabela “detalhe_votação_zona_proj”. É executado pela atividade “PROJ_DET_VOT”.
consolida_dados.jar	Executa a compactação dos dados e envia os arquivos ao Sagitarii.
split_demo.jar	Faz o download dos arquivos do Sagitarii, descompacta-os e envia o seu conteúdo de volta ao Sagitarii.
r-wrapper.jar	Wrapper do Sagitarii para executar scripts “R”. Não apagar.

É necessário informar ao Sagitarii a existência dos wrappers, assim o sistema poderá enviar estes aos nós de execução para que possam processar as atividades de maneira correta quando os comandos forem enviados. As atividades podem ser executadas tanto por instruções SQL quanto por aplicativos em nós de execução (programas), então, a maneira como uma atividade é executada no Sagitarii é chamada de “Activation Executor”. Uma atividade usa um executor para cumprir sua tarefa, que pode ser um programa ou uma instrução SQL. Para cadastrar um executor do tipo wrapper, clique no menu em “View Activation Executors”. Uma lista dos executores já criados será exibida. Clique em “New Activation Executor”. O sistema exibirá a tela de cadastro de executores. A figura 3 ilustra a lista de executores e a figura 4 ilustra a tela de cadastro de executores.

Alias	Type	Wrapper	SQL	Pre-selection
SPLIT_001	SPLIT_WAP	split_demo.jar		
SEL_001	SELECT		insert into sel_001 (id_experiment, sexo, qtd_votos_nominais, nr_zona, qtd_aptos, faixa_etaria, qtd_compartimento, qtd_eleitores_no_perfil, numero_zona, uf) select s1.id_experiment, s1.sexo, s1.qtd_votos_nominais, s1.nr_zona, s1.qtd_aptos, s1.faixa_etaria, s1.qtd_compartimento, s1.qtd_eleitores_no_perfil, s1.numero_zona, s1.uf from join_001 as s1 where (s1.qtd_aptos > 70000)	
PROD_PER_ELEIT	WAP	perfil_eleitorado.jar		
PROD_DET_VOT	WAP	detalhe_vot_zona.jar		
JOIN_001	SELECT		insert into join_001 (id_experiment, faixa_etaria, uf, sexo, qtd_eleitores_no_perfil, nr_zona, numero_zona, qtd_votos_nominais, qtd_aptos, qtd_compartimento) select s1.id_experiment, s1.faixa_etaria, s1.uf, s1.sexo, s1.qtd_eleitores_no_perfil, s1.nr_zona, s2.numero_zona, s2.qtd_votos_nominais, s2.qtd_aptos, s2.qtd_compartimento from perfil_eleitorado_proj as s1, detalhe_votacao_zona_proj as s2 where (s1.nr_zona = s2.numero_zona) and (s1.id_experiment = s2.id_experiment)	
generate_chart	RSRIPT	gchart.r	uf,sexo	
executor_r	RAUNNER	r-wrapper.jar		
CONSOLIDA_DADOS	REDUCE	consolida_dados.jar	uf,sexo	
BAR_FUNCTION	RSRIPT	bar.r		

1 to 9 of 9 Records

< < 1 > >

Create Executor

Alias

Activity type

▼

Activation wrapper

Selecionar arquivo...

Nenhum arquivo selecionado.

Selection / Grouping executor

1

Login

Name	Description	
sel_001	Seleção da tabela de junção	
reduce_001	Tabela com os resultados do workflow	
perfil_eleitorado_proj	Projeção da tabela de perfil do eleitorado	
perfil_eleitorado	Dados dos eleitores por Zona Eleitoral	
join_001	Junção das tabelas de projeção	

Preencha os campos com um apelido para o executor, que será referenciado na tela de cadastro de atividades, um arquivo de aplicativo wrapper, se for o caso e uma instrução SQL, se for o caso. É necessário ao menos um wrapper ou uma instrução SQL. Selecione o tipo de atividade a que este executor deverá ser associado. Quando criar uma atividade, somente os executores do tipo apropriado serão exibidos como opção para o tipo de atividade selecionado. Quando um executor possuir uma instrução SQL e um wrapper, a instrução será executada primeiro e os dados resultantes serão passados ao wrapper nos nós de execução. Os dados retornados pelos nós de execução serão encaminhados à tabela de saída da atividade. Quando um executor possuir somente intrução SQL, esta deverá conter uma instrução INSERT na tabela de saída da atividade, pois será executada diretamente no servidor.

Conforme a figura 3, para o workflow de exemplo foram criados nove executores: os do tipo “MAP”, executam aplicativos no nós de processamento para projetar as colunas das tabelas com os dados originais (“PROJ_PER_ELEIT” para executar o wrapper “perfil_eleitorado.jar” e “PROJ_DET_VOT” para executar o wrapper “detalhe_vot_zona.jar”). Depois da projeção das colunas feita por estes wrappers, é necessário unir os dados das duas tabelas resultantes da projeção, então foi criado o executor “JOIN_001”, do tipo “SELECT” que executa um JOIN das tabelas “perfil_eleitorado_proj” e “detalhe_votacao_zona_proj” diretamente na tabela “join_001”. Este executor é processado internamente pelo Sagitarii e não é enviado aos nós de processamento. Depois da junção, é necessário filtrar os dados segundo um critério. Criou-se então o executor “SEL_001”. Este é um executor do tipo “SELECT”, que insere diretamente na tabela “sel_001” o resultado da seleção dos dados da tabela “join_001” segundo o critério “*quantidade de eleitores aptos a votar maior que 7.000*”. Este executor também é processado internamente pelo Sagitarii. Após projetar as colunas, unir as tabelas e filtrar os dados, é hora de compactar os arquivos. Criou-se o executor “CONSOLIDA_DADOS”, do tipo “REDUCE”. Este executor combina uma sequencia de campos da tabela “sel_001” separados por vírgula com o wrapper “consolida_dados.jar”. O Sagitarii agrupa os dados da tabela “sel_001” (que é a tabela de entrada da atividade que roda este executor) usando os campos informados e envia os dados para os nós de processamentos onde o wrapper “consolida_dados.jar” trabalha estes dados (salva em um arquivo, compacta e envia ao Sagitarii). Em paralelo com a compactação, o executor “GENERATE_CHART”, do tipo “RSCRIPT” estará gerando os gráficos e enviando os arquivos de imagem ao Sagitarii (figura 4.a). Este executor é na verdade um script “R” que será enviado ao executor “EXECUTOR_R”, do tipo “RRUNNER” pelo próprio nó de processamento, que executará o processador de “R” do Sagitarii (“r-wrapper.jar”). O nó de processamento, ao receber um executor do tipo “RSCRIPT” (um script “R”) ele inicia o wrapper “r-wrapper.jar” e passa o script como parâmetro.

Após o término da compactação, será feita uma descompactação dos arquivos gerados e um upload de seu conteúdo, apenas para demonstrar como executar este tipo de tarefa. O executor “SPLIT_001”, do tipo “SPLIT_MAP” executará o wrapper “split_demo.jar”, que receberá os nomes dos arquivos compactados (gerados pelo executor “CONSOLIDA_DADOS”) e fará download destes arquivos do Sagitarii. Após o download, irá descompactar estes arquivos e enviar seu conteúdo de volta ao Sagitarii, gravando o resultado em uma tabela (figura 4.b).

Além destes , ainda existe um outro executor do tipo “RSCRIPT” cadastrado (“BAR_FUNCTION”) que não será executado por nenhuma atividade. Ele foi cadastrado simplesmente porque é uma biblioteca usada pelo script “GENERATE_CHART” e necessita estar presente em todos os nós de processamento junto com os outros executores. Todos os executores cadastrados serão baixados pelos nós de processamento quando forem iniciados, então uma boa forma de enviar arquivos aos nós de processamento é cadastrando-os como executores.

Vale lembrar que todas estas atividades foram elaboradas meramente para efeito didático a fim de exemplificar o uso do sistema e não possuem aplicação prática.

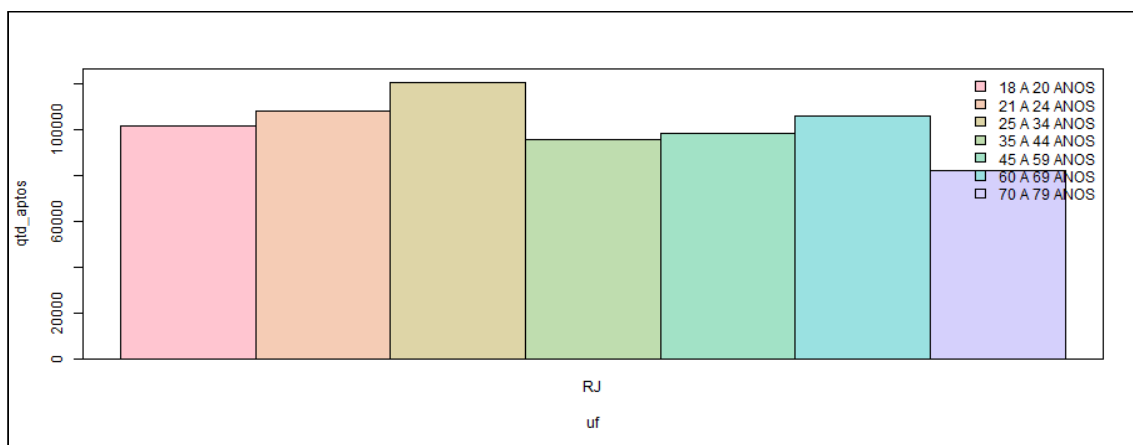


Figura 4.a. Gráfico gerado pelo processador “R”

	id_custom [PK] serial	id_experimen integer	id_activity integer	source_id integer	index_numbe integer	source_table character vai	taskid character vai	zipfile character varying(contentfile character varying(250)
86	86	1	10	5	18	reduce_001	09AD21B1	09AD21B1.zip	SP-FEMININO_18.txt
87	87	1	10	5	19	reduce_001	09AD21B1	09AD21B1.zip	SP-FEMININO_19.txt
88	88	1	10	5	20	reduce_001	09AD21B1	09AD21B1.zip	SP-FEMININO_20.txt
89	89	1	10	5	21	reduce_001	09AD21B1	09AD21B1.zip	SP-FEMININO_21.txt
90	90	1	10	5	22	reduce_001	09AD21B1	09AD21B1.zip	SP-FEMININO_22.txt
91	91	1	10	5	23	reduce_001	09AD21B1	09AD21B1.zip	SP-FEMININO_23.txt
92	92	1	10	5	24	reduce_001	09AD21B1	09AD21B1.zip	SP-FEMININO_24.txt
93	93	1	10	5	25	reduce_001	09AD21B1	09AD21B1.zip	SP-FEMININO_25.txt
94	94	1	10	5	26	reduce_001	09AD21B1	09AD21B1.zip	SP-FEMININO_26.txt
95	95	1	10	5	27	reduce_001	09AD21B1	09AD21B1.zip	SP-FEMININO_27.txt
96	96	1	10	6	0	reduce_001	FC427F50	FC427F50.zip	SP-MASCULINO_0.txt
97	97	1	10	6	1	reduce_001	FC427F50	FC427F50.zip	SP-MASCULINO_1.txt
98	98	1	10	5	28	reduce_001	09AD21B1	09AD21B1.zip	SP-FEMININO_28.txt
99	99	1	10	6	2	reduce_001	FC427F50	FC427F50.zip	SP-MASCULINO_2.txt
100	100	1	10	6	3	reduce_001	FC427F50	FC427F50.zip	SP-MASCULINO_3.txt
101	101	1	10	5	29	reduce_001	09AD21B1	09AD21B1.zip	SP-FEMININO_29.txt
102	102	1	10	6	4	reduce_001	FC427F50	FC427F50.zip	SP-MASCULINO_4.txt

Figura 4.b. Tabela “split_001” com o resultado de sua atividade.

7. Criando as Atividades

Tendo cumprido os requisitos básicos (criação das tabelas e executores), é hora de criar as atividades do workflow. Nesta etapa é que será informada a ordem de execução das atividades, que tipo de tarefa cada atividade vai executar (os executores) e que tabelas servirão como fonte de dados de consumo e destino dos dados produzidos. O Sagitarii oferece uma interface gráfica para a criação das atividades. É importante observar que, embora as atividades do tipo “SELECT” já tenham suas tabelas de consumo e produto (caso haja) definidas em seus scripts SQL, o Sagitarii infere estas mesmas informações baseado no que está sendo informado no momento da criação de cada atividade. Por exemplo: a atividade “JOIN_001” tem suas tabelas de produto e consumo já definidas no script SQL do executor, mas quando ela foi criada, o Sagitarii inferiu que as tabelas de entrada serão “perfil_eleitorado_proj” e “detalhe_votacao_zona_proj”, pois estas tabelas são as tabelas de saída (produto) das atividades imediatamente anteriores à atividade “JOIN_001” (“PROJ_PER_ELEIT” produz na tabela “perfil_eleitorado_proj” e “PROJ_DET_VOT” produz na tabela “detalhe_votacao_zona_proj”). Neste caso, estes dados são apenas informativos, pois as tabelas que serão usadas são de responsabilidade do usuário no momento do script SQL no cadastro do executor. As atividades do tipo “MAP” não possuem script SQL, sendo necessário informar as

tabelas de consumo e produto no momento da criação da atividade. Atividades do tipo “REDUCE” possuem a tabela de consumo como sendo a mesma tabela de produto da atividade anterior (e onde irão atuar os atributos de agrupamento separados por vírgula informados no cadastro do executor) e a tabela de produto é informada no momento da criação da atividade. O Apêndice “B” contém a lista das atividades cadastradas e suas relações de consumo e produto. A tabela 2 possui a lista de executores e seus respectivos scripts SQL e wrappers.

A saída da atividade “GEN_BAR_PLOT”, que gera os gráficos do exemplo, não produz nenhuma informação que deva ser armazenada em banco de dados (somente arquivos de imagem que são armazenados em local apropriado pelo sistema, devidamente identificados, quando enviados pelos nós de processamento) então sua tabela de saída é a mesma de entrada, simplesmente por não ser possível deixar esta informação em branco.

Tabela 2. Lista de Executores usados no exemplo

Alias	Type	Wrapper	SQL
SPLIT_001	SPLIT_MAP	split_demo.jar	
SEL_001	SELECT		insert into sel_001 (id_experiment, sexo, qtd_votos_nominais, nr_zona, qtd_aptos, faixa_etaria, qtd_comparecimento, qtd_eleitores_no_perfil, numero_zona, uf) select s1.id_experiment, s1.sexo, s1.qtd_votos_nominais, s1.nr_zona, s1.qtd_aptos, s1.faixa_etaria, s1.qtd_comparecimento, s1.qtd_eleitores_no_perfil, s1.numero_zona, s1.uf from join_001 as s1 where (s1.qtd_aptos > 70000)
PROJ_PER_ELEIT	MAP	perfil_eleitorado.jar	
PROJ_DET_VOT	MAP	detalhe_vot_zona.jar	
JOIN_001	SELECT		insert into join_001 (id_experiment, faixa_etaria, uf, sexo, qtd_eleitores_no_perfil, nr_zona, numero_zona, qtd_votos_nominais, qtd_aptos, qtd_comparecimento) select s1.id_experiment, s1.faixa_etaria, s1.uf, s1.sexo, s1.qtd_eleitores_no_perfil, s1.nr_zona, s2.numero_zona, s2.qtd_votos_nominais, s2.qtd_aptos, s2.qtd_comparecimento from perfil_eleitorado_proj as s1, detalhe_votacao_zona_proj as s2 where (s1.nr_zona = s2.numero_zona) and (s1.id_experiment = s2.id_experiment)
generate_chart	RSCRIPT	genchart.r	uf,sexo
executor_r	RRUNNER	r-wrapper.jar	
CONSOLIDA_DADOS	REDUCE	consolida_dados.jar	uf,sexo
BAR_FUNCTION	RSCRIPT	bar.r	

Para criar as atividades do workflow, clique no ícone “Home” na barra de menu. Localize o workflow desejado e clique no ícone “Manage Activities”. O sistema apresentará a tela de cadastro de atividades para o workflow selecionado. A figura 5 ilustra a tela de cadastro de atividades.

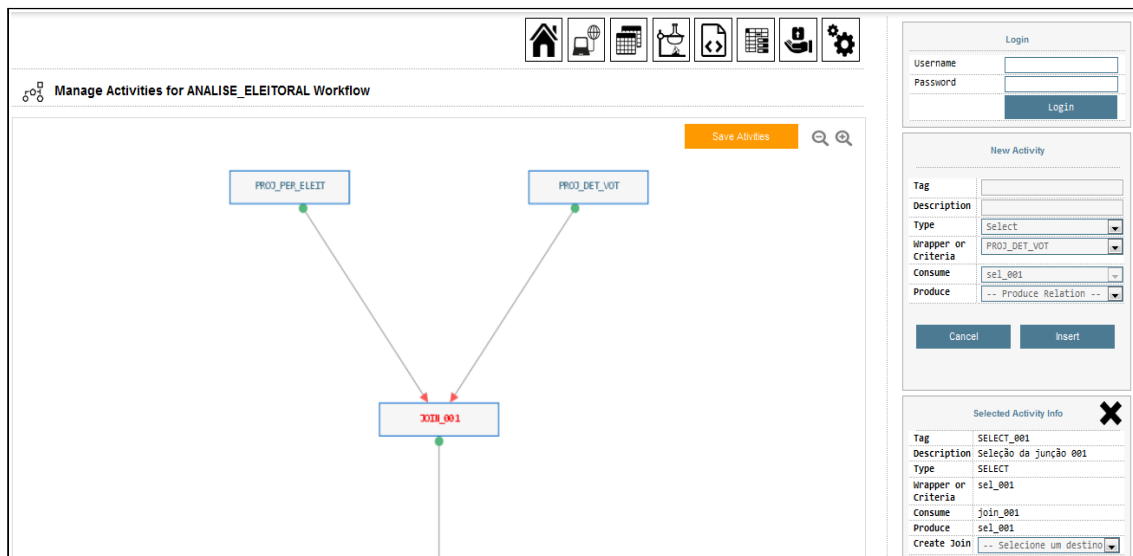


Figura 5. Tela de cadastro de atividades

Na lateral direita da tela, selecione “New Activity”. Preencha os campos com uma etiqueta descritiva da atividade (um nome curto), uma descrição do que a atividade faz, o tipo de atividade, um executor para esta atividade, a tabela de onde os dados serão obtidos e a tabela onde os dados resultantes deverão ser salvos.

Quando um tipo de atividade é selecionado da lista, a lista de executores é automaticamente filtrada para exibir somente os executores correspondentes a este tipo de atividade. Atividades do tipo “SELECT” e “REDUCE” terão seus respectivos scripts SQL de seus executores exibidos na parte inferior da área de cadastro de atividades.

Para criar a atividade seguinte no fluxo, clique em uma atividade (ela ficará na cor azul para mostrar que foi selecionada) e então clique em “New Activity”. A nova atividade será colocada após a atividade selecionada no fluxo de execução do workflow. Uma seta será criada ligando as duas atividades: a ponta de origem da seta conterá um círculo preenchido de verde e a ponta de destino (sentido do fluxo) conterá um triângulo preenchido de vermelho.

Para criar uma atividade de junção (que utilizam mais de uma atividade como origem), selecione uma outra atividade como origem e, em sua caixa de informações, exibida no lado direito da tela, selecione a atividade de junção na caixa “Create Join”. A figura 6 ilustra o procedimento de criação da atividade “JOIN_001” : foi criada a atividade “PROJ_PER_ELEIT”, clicou-se nesta atividade para selecioná-la, foi criada a atividade “JOIN_001” como passo seguinte no fluxo. Então foi criada a atividade “PROJ_DET_VOT”, clicou-se nesta atividade para selecioná-la e então a atividade “JOIN_001” foi selecionada em sua lista “Create Join”. O sistema automaticamente cria mais uma seta de fluxo entre as atividades e adiciona a tabela de produto da atividade anterior como tabela de consumo da atividade seguinte. Somente atividades do tipo “SELECT” ou “REDUCE” aparecem na lista “Create Join”.

Selected Activity Info

Tag

PROJ_DET_VOT

Description

Projeção da tabela de detalhe da votação por zona

Type

MAP

Wrapper or Criteria

detalhe_vot_zona.jar

Consume

detalhe_votacao_zona

Produce

detalhe_votacao_zona_proj

Create Join

-- Selecione um destino

-- Selecione um destino --

JOIN_001

SELECT_001

REDUCE_001

Figura 6. Criação de uma atividade de junção

Para remover uma linha de fluxo (seta de ligação entre as atividades), clique na linha que deseja remover e clique no ícone “Remove Dependency Connector”, na caixa de informações no lado direito da tela. Para remover uma atividade, selecione a atividade e clique no ícone “Delete Activity”, na caixa de informações da atividade, no lado direito da tela. A figura 7 ilustra as atividades do workflow e suas linhas de fluxo. Ao terminar de cadastrar todas as atividades, clique em “Save Activities”.

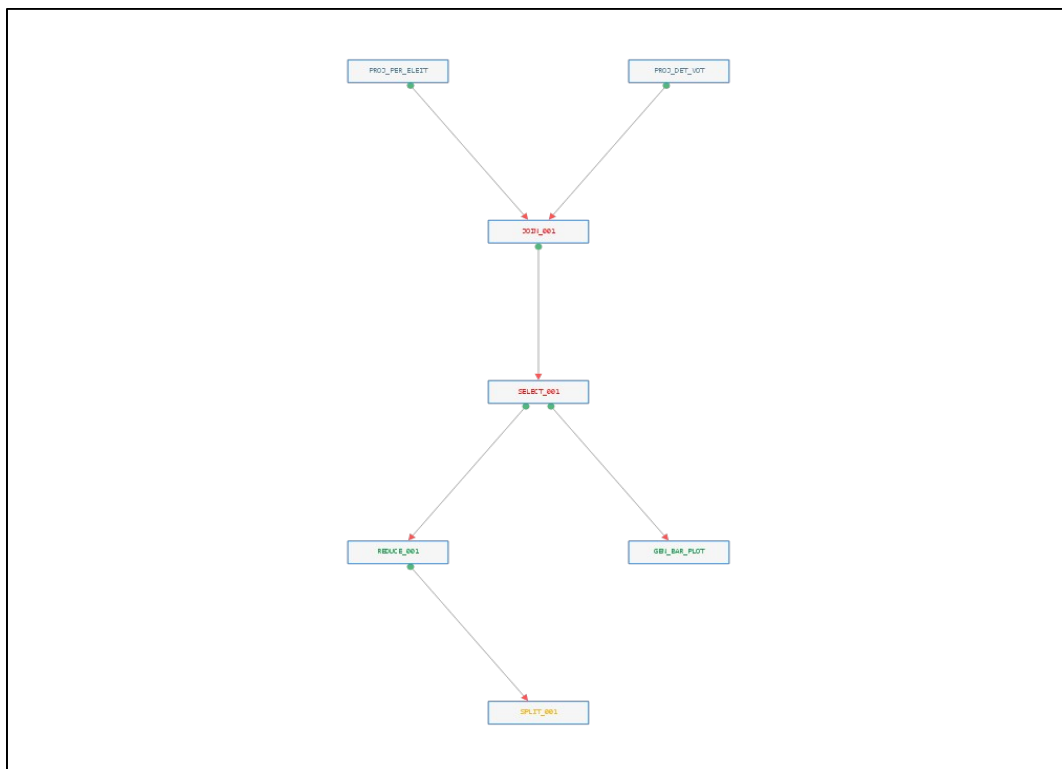


Figura 7. Atividades criadas para o workflow de exemplo.

8. Criando um Experimento

O Sagitarii trata os workflows somente como “moldes” para criar experimentos. Um experimento é uma instância executável de um workflow. Um workflow poderá possuir quantos experimentos forem necessários, sendo que cada experimento poderá ter suas atividades alteradas para melhor conveniência do usuário, sem que isso altere o workflow.

Para criar um experimento, clique no ícone “Home” na barra de menus. O sistema apresentará a lista de workflows disponíveis. Localize o workflow desejado e clique no ícone “Generate New Experiment”. O sistema irá criar um experimento (uma réplica das informações do workflow). A figura 8 ilustra a tela de detalhes do workflow com um experimento criado.

Para verificar as informações de um experimento, localize o experimento desejado na lista e clique no ícone “Verify and Execute”. O sistema apresentará a tela de detalhes do experimento. Esta tela é dividida em três áreas: informações básicas, detalhes do fluxo de atividades (exibindo as relações de consumo e produto) e as informações de fragmentação (quando o sistema separa as atividades em grupos de execução para melhorar o desempenho [OGASAWARA, 2011]).

Os nomes das relações nas atividades podem ser clicados. Uma caixa informativa na lateral direita da tela irá exibir os atributos (campos) da tabela selecionada. Também é possível verificar os pontos de início (em azul) e término (em vermelho) do experimento, bem como todo o fluxo intermediário de produção e consumo dos dados.

9. Importando dados iniciais

É possível importar os dados iniciais de uma atividade para as tabelas apropriadas no Sagitarii usando o aplicativo de processamento de atividades “Sagitarii Teapot”, desde que as tabelas que irão receber os dados possuam, no mínimo, os mesmos atributos (campos) que o arquivo CSV possui (colunas).

Clique no ícone “Download Teapot Cluster” no topo da tela. Após fazer o download do arquivo, descompacte-o em um diretório qualquer. O arquivo CSV que será importado deverá estar neste mesmo diretório. Crie uma pasta “outbox” logo abaixo deste diretório e grave ali todos os demais arquivos que serão importados juntamente com os dados CSV. Estes arquivos deverão estar referenciados em alguma coluna dentro do CSV. Edite o arquivo “config.xml”, colocando o endereço do servidor Sagitarii na tag “hostURL”.

Para importar os dados, execute o Teapot usando os parâmetros:

```
java -jar teapot-1.0.125.jar upload <nome_do_arquivo_csv>  
<nome_da_tabela_destino> <tag_do_experimento> <pasta_de_trabalho>
```

Exemplo:

```
java -jar teapot-1.0.125.jar upload mydata.csv mytable 2D7B8C77-8822-49B  
/home/user/insurance
```

Neste caso, mydata.csv está na pasta /home/user/insurance e os arquivos de dados em /home/user/insurance/outbox. É importante verificar o arquivo CSV e

confirmar se as colunas da primeira linha do arquivo possuem os mesmos nomes dos atributos da tabela que receberá os dados do arquivo. É importante verificar se o CSV está bem formatado no padrão RFC 4180, sendo obrigatório a presença dos nomes das colunas. A figura 9 ilustra o trecho inicial do arquivo CSV com os dados do perfil de eleitorado, baixado do site do STE e a figura 10 ilustra o esquema da tabela do Sagitarii que receberá os dados deste arquivo.



Figura 8. Tela de detalhes do workflow com o experimento de exemplo.

Se desejar enviar arquivos para um experimento, use o comando

```
java -jar teapot-1.2.0-beta.jar upload-folder <nome-da-pasta>
<tag_do_experimento>
```

Todo o conteúdo da pasta <nome-da-pasta> será enviado ao Sagitarii aos cuidados do experimento <tag-do-experimento>. Todos os arquivos de um experimento precisam ter nomes únicos, pois serão solicitados pelo nome quando os nós de processamento precisarem fazer download.

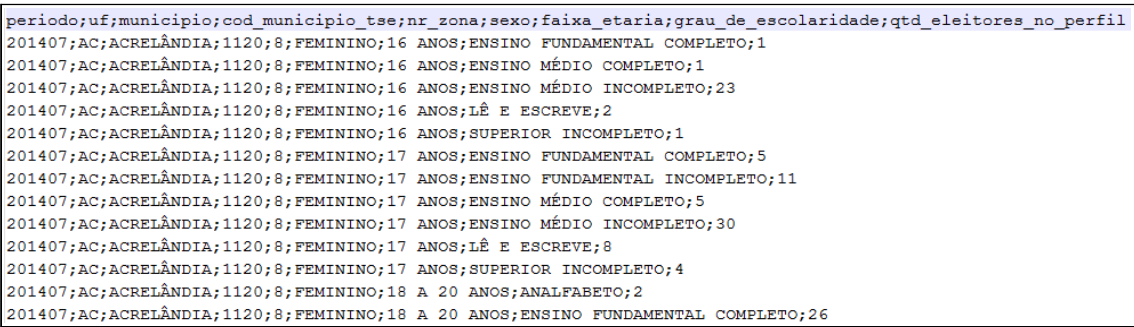


Figura 9. Trecho inicial do arquivo CSV perfil_eleitorado_2014.txt

```

CREATE TABLE perfil_eleitorado
(
    id_custom serial NOT NULL,
    id_experiment integer,
    id_activity integer,
    index_number integer,
    taskid character varying(20),
    periodo character varying(250),
    uf character varying(250),
    municipio character varying(250),
    cod_municipio_tse integer,
    nr_zona integer,
    sexo character varying(250),
    grau_de_escolaridade character varying(250),
    qtd_eleitores_no_perfil integer,
    faixa_etaria character varying(250),
    CONSTRAINT perfil_eleitorado_pkey PRIMARY KEY (id_custom ),
    CONSTRAINT perfil_eleitorado_fkac FOREIGN KEY (id_activity)
        REFERENCES activities (id_activity) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE NO ACTION,
    CONSTRAINT perfil_eleitorado_fkex FOREIGN KEY (id_experiment)
        REFERENCES experiments (id_experiment) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE NO ACTION
)

```

Figura 10. Esquema da tabela perfil_eleitorado

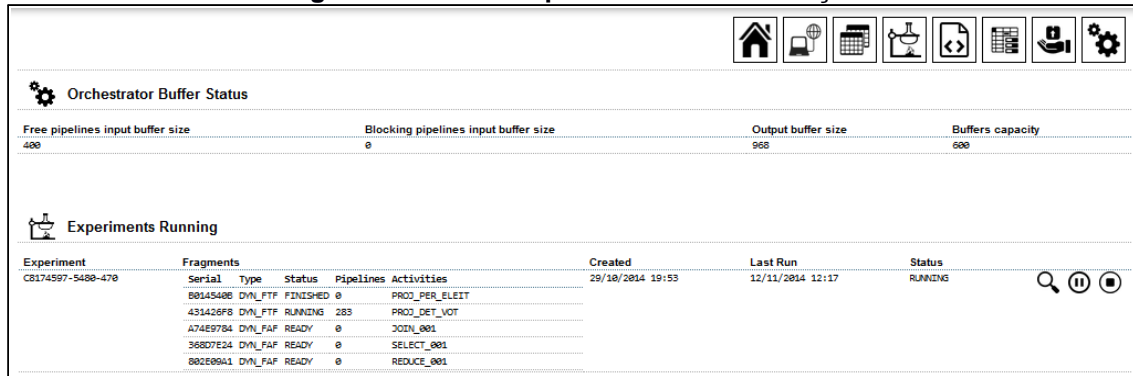
Para o experimento de exemplo (etiqueta C8174597-5480-470), foram importados os dois arquivos baixados do site do STE: “perfil_eleitorado_2014.txt” para a tabela “perfil_eleitorado” e “detalhe_votacao_munzona_2014_BR.txt” para a tabela “detalhe_votacao_zona”.

Após importar todos os dados necessários para as atividades de início do experimento, volte à página de detalhes do experimento e confirme se o botão “Start Experiment” surge na cor azul. Isto significa que o experimento pode ser executado.

10. Executando um experimento

Após clicar em “Start Experiment”, é possível acompanhar a execução das atividades. Clique no ícone “Running Experiments” na barra de menu. O sistema apresentará a tela de experimentos em execução (figura 11). Nesta tela são informados detalhes da execução do experimento, como número de pipelines gerados, número de pipelines que estão na fila de processamento (input buffer) e os que foram já processados (output buffer). Também é possível verificar o tamanho do buffer de processamento. Pipelines são as unidades de processamento do Sagitarii e são compostos por blocos de atividades que podem ser executados em sequencia e que são enviados a um mesmo nó de processamento a fim de aumentar o desempenho [OGASAWARA, 2011]. Um pipeline pode conter uma ou mais atividades a serem executadas e pode ser executado pelo próprio servidor (atividades que envolvam consultas SQL) ou pelos nós de processamento (atividades que envolvam processamento por aplicativo ou manuseio de arquivos). O buffer de processamento é um espaço na memória do servidor onde ele armazena uma certa quantidade de pipelines que serão entregues ao nós de processamento. Isto evita o acesso frequente ao banco de dados.

Figura 11. Tela de experimentos em execução











Embora o experimento esteja sendo executado, nenhum pipeline foi processado, pois não há nós de processamento em execução. Para este exemplo, é suficiente executar apenas um nó de processamento, portanto, utilize o mesmo programa “Teapot” que foi usado para fazer a migração dos dados dos arquivos CSV. Execute o “Teapot” sem parâmetros:

```
java -jar teapot-1.2.0-beta.jar
```

O nó de processamento irá atualizar os aplicativos de wrapper, baixando-os do servidor Sagitarii e então irá enviar os dados do computador que está executando o programa para o Sagitarii. O Sagitarii então irá verificar se existe algum pipeline disponível para ser processado e irá enviá-lo ao nó de processamento. Esta tarefa de enviar dados da máquina e receber tarefas do servidor se repete a cada intervalo de tempo configurado no arquivo “config.xml” do “Teapot” (o mesmo que foi editado para informar o endereço do servidor Sagitarii) na tag “poolIntervalMiliSeconds”. O valor informado é em milissegundos.

Os nós de processamento ativos podem ser verificados no próprio Sagitarii. Clique no ícone “View Running Nodes” na barra de menu. O sistema apresentará a tela de nós de processamento, onde é possível verificar as informações de cada computador que está trabalhando para o Sagitarii, bem como sua carga de trabalho (gasto de CPU) atual. É possível ver também quantos pipelines um determinado nó de processamento já executou. Na área de informação do nó de processamento é possível ver também quantos “threads” estão executando tarefas no momento com o nome da atividade e o código do pipeline. A figura 12 ilustra a tela de nós de processamento ativos, com um nó de processamento em execução, um “thread” neste nó executando um pipeline da atividade “PROJ_DET_VOT” e o “Main Cluster”, que é o nó de processamento interno do Sagitarii, encarregado de processar as atividades que envolvam consultas SQL.

       							
Operational System	Machine Name	MAC Address / Serial	IP Address	Java	Active Tasks	Finished Tasks	Cluster CPU Load
Main Cluster	Sagitarii Server	S0-A0-G0-I0-T0-A0-R0-II	Local Machine	0.0	0	0	
Last Announce	Max Allowed Tasks	Cores	Status	Last Error			
11/11/2014 17:10:11	6	8	IDLE				

Operational System	Machine Name	MAC Address / Serial	IP Address	Java	Active Tasks	Finished Tasks	Cluster CPU Load
Windows 7	HAN.casnav.mb	9C-8E-99-35-12-BA-FD35	10.5.114.217	1.7.0_03	1	11	
Last Announce	Max Allowed Tasks	Cores	Status	Last Error			
11/11/2014 17:10:15	20	4	ACTIVE				

18042A4F-D6E2-4 - RUNNING
1

Figura 12. Tela de nós de processamento ativos

É possível acompanhar os pipelines que foram entregues para processamento aos nós e não retornaram ainda. Clique no ícone “View Pipeline Delivery Control”. O sistema irá exibir a tela de controle de entrega de pipelines (figura 13). O campo “Age” informa quantos ciclos um pipeline está demorando para ser processado pelo nó de processamento (endereço MAC no campo “Delivered to”). O Sagitarii será programado para identificar pipelines que estão com o campo “Age” muito alto (muito acima da média dos pipelines do mesmo fragmento) e interpretará que ele foi perdido, providenciando seu reenvio (invalidando uma possível resposta tardia do pipeline perdido após o reenvio).










       							
 Pipeline Delivery Control Status							
Pipeline ID	Delivered to	Pipeline Status	Pipeline Type	Age	Activations		
E000067D-5634-4	9C-8E-99-35-12-BA-9A69	RUNNING	NAP	49	Workflow	Experiment	Fragment
929AED6B-A066-4	9C-8E-99-35-12-BA-C378	RUNNING	NAP	49	ANALISE_ELEITORAL	CB174597-5480-470	9178EABC
A721AFB0-7CA0-4	9C-8E-99-35-12-BA-9A69	RUNNING	NAP	46	Workflow	Experiment	Fragment
1819A075-F18A-4	9C-8E-99-35-12-BA-C378	RUNNING	NAP	46	ANALISE_ELEITORAL	CB174597-5480-470	9178EABC
C54375E2-C4C0-4	9C-8E-99-35-12-BA-9A69	RUNNING	NAP	43	Workflow	Experiment	Fragment
2CD98B06-7CFD-4	9C-8E-99-35-12-BA-C378	RUNNING	NAP	43	ANALISE_ELEITORAL	CB174597-5480-470	9178EABC
68EE86D1-B9F9-4	9C-8E-99-35-12-BA-C378	RUNNING	NAP	42	Workflow	Experiment	Fragment
C262AD97-C59F-4	9C-8E-99-35-12-BA-9A69	RUNNING	NAP	42	ANALISE_ELEITORAL	CB174597-5480-470	9178EABC
33E1C0B7-CAC6-4	9C-8E-99-35-12-BA-9A69	RUNNING	NAP	42	Workflow	Experiment	Fragment
58411678-4037-4	9C-8E-99-35-12-BA-C378	RUNNING	NAP	42	ANALISE_ELEITORAL	CB174597-5480-470	9178EABC
13E185E7-AF80-4	9C-8E-99-35-12-BA-9A69	RUNNING	NAP	39	Workflow	Experiment	Fragment
					ANALISE_ELEITORAL	CB174597-5480-470	9178EABC

Figura 13. Tela de controle de entrega de pipelines

A figura 14 ilustra o resultado final do experimento (além das figuras 4.a e 4.b): os arquivos contendo o sumário desejado e o arquivo compactado contendo os arquivos de sumário.

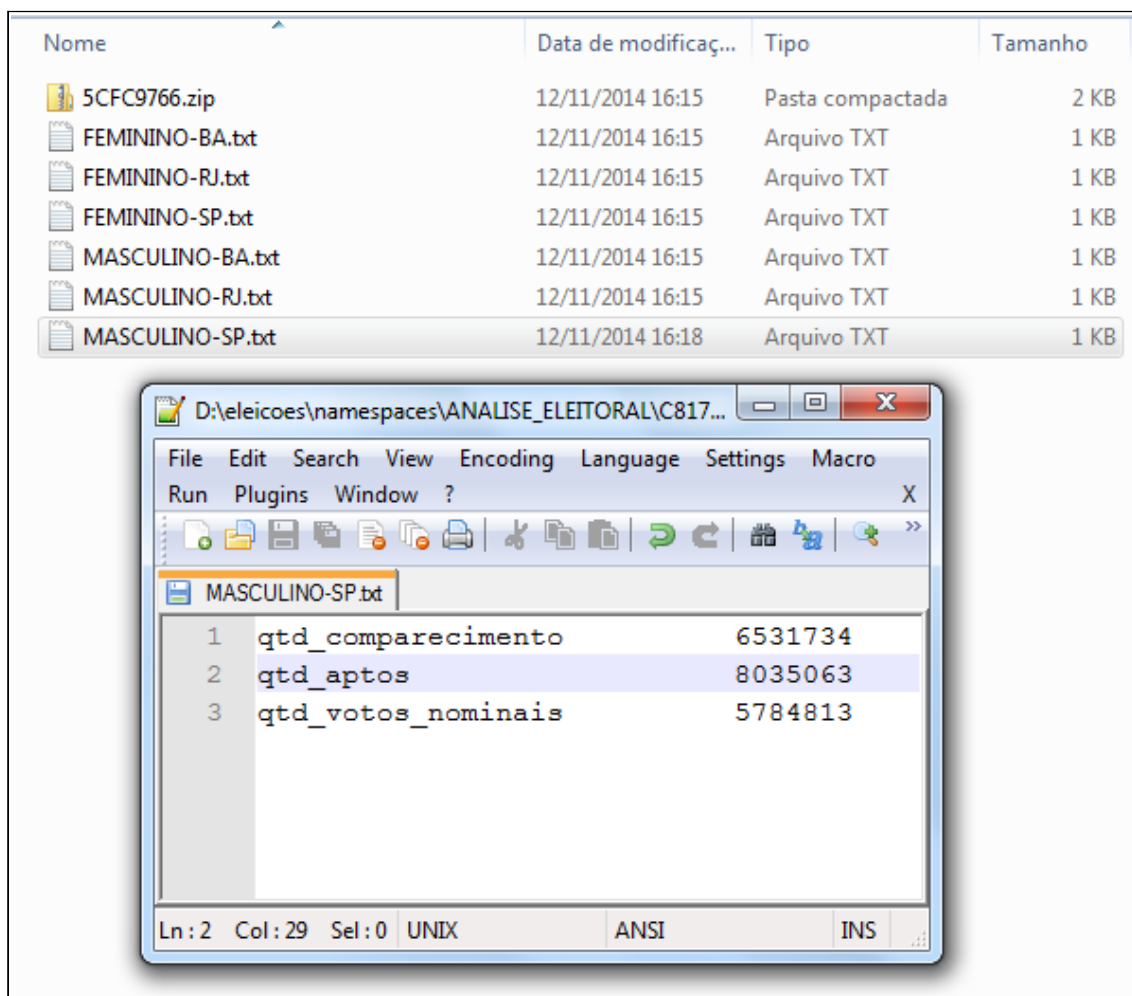


Figura 14. Resultado do experimento

11. Inspeccionando um Experimento

O Sagitarii oferece uma tela para efetuar consultas SQL diretamente pelo sistema, desta forma, será possível acompanhar a evolução do experimento e a criação dos dados e arquivos.

Para inspecionar um experimento, clique no ícone “Experiments”. Localize o experimento na lista e clique no ícone “View More”. O sistema apresentará a tela de detalhes do experimento. Clique no botão “Inspect Data”. A tela de inspeção do experimento será exibida. Nesta tela, é exibida uma lista com as tabelas do usuário e algumas tabelas de sistema (em cor vermelha). Para checar os atributos de uma determinada tabela, clique no ícone “View Schema”. Uma lista com os atributos da tabela será exibida na lateral direita da tela. Abaixo desta tabela, o sistema exibirá uma tabela contendo os arquivos que foram criados pelo experimento nos nós de processamento e enviados ao Sagitarii. É possível efetuar o download de qualquer arquivo clicando no respectivo link na tabela.

As figuras 15a e 15b ilustram a tela de inspeção de experimentos.

Inspect Experiment C8174597-5480-470

ID / Experiment	1 / C8174597-5480-470	Status	FINISHED
ID / Workflow	1 / ANALISE_ELEITORAL	Workspace	dt/workspace
Description	workflow de teste com anlise eleitoral	Frangments	5

Order ID	Serial	Pipelines	Activities
0	6	F81E3C44	508
		ID	Serial
		6	438F686D PROD_PER_ELEIT
1	7	F8C64658	1091
		ID	Serial
		10	8D2C3D89 PROD_DET_VOT
2	8	2790A606	1
		ID	Serial
		7	32C338CA DDTN_001
3	9	8F073D5F	1
		ID	Serial
		8	A1180E17 SELECT_001
4	10	86A34E86	1
		ID	Serial
		9	68F928CA REDUCE_001

SQL Query

```

1 select
2     cast(act.starttime as time),
3     cast(act.endtime as time), p.serial
4 from
5     activations act
6 join
7     pipelines p on act.pipelineid = p.serial limit 100

```

SQL Result

Search

endtime	serial	starttime
15:08:38	A743C96C-8B04-4	15:08:37
15:08:38	31838376-D080-4	15:08:36
15:08:39	90808C89-3747-4	15:08:39
15:08:48	8F972870-E17F-4	15:08:48
15:08:48	7E0528E8-7D6F-4	15:08:48

1 to 5 of 100 Records

1

2

3

4

5

6






Username

Password

Login

Table "pipelines"

qid_activations	integer
status	character varying
id_fragment	integer
serial	character varying
id_pipeline	integer
type	character varying
content	text

System and Custom Tables		
		Search <input type="text"/>
Name	Description	
[SYSTEM TABLE] Activations	Data generated by nodes	
[SYSTEM TABLE] Pipelines	Pipelines sent to nodes	
detalhe_votacao_zona	Detalhe da votação por município e Zona Eleitoral	
detalhe_votacao_zona_proj	Projeção da tabela de detalhe da votação	
join_001	Junção das tabelas de projeção	

12. Integração com o “R”

O Sagitarri oferece uma integração especial com o “R”. Segundo definição do próprio site:

“R is a language and environment for statistical computing and graphics. It is a [GNU project](#) which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.

R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.

One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.”

Naturalmente, a interação com o “R” poderia ser feita usando um wrapper, mas o Sagitarri possui um wrapper especial, que utiliza a biblioteca JRI instalada com o rJava.

13. Instalando o “R”

Após baixar e instalar o “R”, é necessário configurar as variáveis de ambiente “PATH” e “R_HOME”. A variável “PATH” precisa conter os caminhos para as pastas “bin\x64” e “library”, ambas na pasta de instalação do “R”. A variável “R_HOME” deve apontar para a pasta de instalação do “R”.

Após instalar o “R”, é necessário instalar o rJava, apenas para usar o JRI. Caso o usuário queira criar um wrapper que utilize o JRI, é necessário incluir a pasta das bibliotecas no “classpath”:

```
java -Djava.library.path=<RJAVA_ROOT>/jri -jar <SEU_WRAPPER>.jar
```

IMPORTANTE: A pasta onde estão localizadas as bibliotecas do JRI deve ser configurada no arquivo “config.xml” em todos os nós de execução (Teapot), na tag “rPath”.

Para testar a instalação do “R” e a integração com o Java, pode-se executar o wrapper criado para executar atividades “R” do Sagitarri:

```
java -Djava.library.path=/usr/lib64/R/library/rJava/jri/  
-jar r-wrapper.jar /home/my/work/folder
```

Este wrapper, como todos os outros, só estarão disponíveis após o nó de execução Teapot ter sido executado pelo menos uma vez, pois são baixados do servidor Sagitarri pelo Teapot.

14. Criando executores para o “R”

O Sagitarii possui o executor “executor_r”, do tipo “RRUNNER”, que executa o wrapper “r-wrapper.jar”. Este wrapper é encarregado de executar os scripts R do usuário e já está presente na instalação padrão do Sagitarii, não podendo ser removido.

As atividades irão utilizar os executores do tipo “RSCRIPT” para executar scripts “R”. Em resumo, quando o nó de execução Teapot é iniciado, ele irá fazer o download de todos os executores, incluindo os wrappers e os scripts “R”. Quando uma atividade do tipo “RSCRIPT” usa um script “R” no lugar de um wrapper, então o wrapper “RRUNNER” (“r-wrapper.jar”) é executado, chamando o script em questão e passando os dados recebidos pelo arquivo CSV. O wrapper “r-wrapper.jar” passará ao script “R” o caminho da pasta de trabalho da atividade que será executada usando o JRI. O script precisará desta informação para acessar o arquivo CSV com os dados necessários a sua execução, bem como os arquivos de dados enviados pelo servidor. O script terá esta informação disponível para acesso imediato na variável “sagitariiWorkFolder”. A tabela 3 possui os caminhos das pastas que deverão ser utilizadas pelo script e sua finalidade.

Atividades do tipo “RSCRIPT” podem ser associadas a todos os operadores, exceto “QUERY”, pois este é executado internamente pelo próprio servidor Sagitarii.

Tabela 3. Lista das pastas usadas pelo script “R”

sagitariiWorkFolder	Caminho completo da pasta de trabalho da instância da atividade que está executando o script. Esta pasta contém o arquivo “sagi_input.txt”, que é o arquivo CSV com os dados de execução da atividade. Nesta pasta o script DEVERÁ gravar o arquivo “sagi_output.txt” contendo os dados CSV de saída da atividade.
SagitariiWorkFolder / inbox	Pasta contendo os arquivos de dados do usuário enviados pelo servidor, que foram produzidos pela atividade anterior ou por carga inicial.
SagitariiWorkFolder / outbox	Pasta de saída dos arquivos produzidos pela atividade. Qualquer arquivo que deva ser passado para a atividade seguinte deverá ser gravado nesta pasta e referenciado em uma das colunas do arquivo CSV “sagi_output.txt”.

Para usar um executor “RSCRIPT” com uma atividade “REDUCE”, é necessário informar um critério de agrupamento, que são os campos da tabela de entrada, separados por vírgula (figura 16). O Sagitarii irá aplicar um “SELECT DISTINCT” na tabela de entrada usando estes atributos e gerar uma instância para cada grupo destes registros onde os demais atributos forem diferentes. A figura 16.a ilustra uma possível situação de agrupamento. Usando os dados da figura, o Sagitarii faria um “select distinct”, resultando em “MASCULINO,RJ”, “FEMININO,RJ”, “MASCULINO,SP” e “FEMININO,SP”. Depois criaria instância onde os dados diferem destes resultados (figura 16.b).

generate_chart	RSCRIPT	genchart.r	uf,sexo
executor_r	RRUNNER	r-wrapper.jar	
CONSOLIDA_DADOS	REDUCE	consolida_dados.jar	uf,sexo
BAR_FUNCTION	RSCRIPT	bar.r	

Figura 16. Executor para script “R”

A figura 17 ilustra a tela de cadastro de executores com os dados do executor “generate_chart” do exemplo.

	SEXO	UF	QTD	
	MASCULINO	RJ	300	
	FEMININO	RJ	250	
	MASCULINO	RJ	150	
	FEMININO	SP	340	
	MASCULINO	SP	134	

Figura 16.a. Exemplo de dados a serem agrupados pelo “REDUCE”

Instance 1	MASCULINO	RJ	300
	MASCULINO	RJ	150
Instance 2	FEMININO	RJ	250
Instance 3	FEMININO	SP	340
Instance 4	MASCULINO	SP	134

Figura 16.b. Exemplo de instâncias geradas pelo “REDUCE” agrupando por SEXO,UF

Create R Script Executor

Alias: MYRSCRIPT

R Script File: activity.R

Comma separated grouping attributes

- 1 sexo,uf

Figura 17. Criação de um executor de script “R”

O Teapot, ao se deparar com uma atividade do tipo “RSCRIPT”, executa o wrapper “r-wrapper.jar”, que usa o JRI para configurar a pasta de trabalho e executar o script “R”, que deverá se encarregar de ler os arquivos necessários (sagi-input.txt e a pasta inbox) e gravar os arquivos produzidos (sagi_output.txt e pasta outbox, caso hajam). A figura 18 ilustra um script “R” usado como exemplo. Caso o script necessite de outros scripts (funções e bibliotecas), cadastre-os como “Biblioteca”, assim o Teapot poderá fazer download de todos os scripts necessários. Este recurso foi usado no script de exemplo da figura 18, linha 15. A figura 18a exibe a tela de cadastro de bibliotecas de apoio para os wrappers (quaisquer arquivos que serão utilizados pelos wrappers para seu funcionamento – jar, lib, dat, etc...). Estes arquivos de apoio serão baixados pelo Teapot e colocados na mesma pasta que os wrappers.

```

1  #!/usr/bin/env Rscript
2
3  inputFileFolder <- paste( sagitariiWorkFolder, "inbox", sep = "/" )
4  outputFileFolder <- paste( sagitariiWorkFolder, "outbox", sep = "/" )
5
6  inputFile <- paste( sagitariiWorkFolder, "sagi_input.txt", sep = "/" )
7  inputData <- paste( inputFileFolder, "wine.data", sep = "/" )
8  outputFile <- paste( sagitariiWorkFolder, "sagi_output.txt", sep = "/" )
9
10 param <- read.table( inputFile, header = TRUE, sep = "," )
11 wine <- read.table( inputData, header = TRUE, sep = "," )
12
13 library(nnet)
14
15 source("data-preprocessing.R")

```

Figura 18. Script “R” utilizando a variável de pasta de trabalho

Create new Library for Executor support

Alias: MY_FUNCTION_LIB

Library File: Selecionar arquivo... data-preprocessing.R

Figura 18a. Cadastro de uma biblioteca de funções para o script “R”

Na figura 18 pode-se perceber o uso da variável contendo a pasta de trabalho da atividade (linhas 3 até 8) na composição das pastas de trabalho, entrada e saída de dados do script (na linha 11, o script carrega um arquivo de dados enviado pelo Sagitarii, que está na pasta “inbox”). Estas linhas provavelmente estarão presentes em todos os scripts R usados pelo Sagitarii. A figura 19 ilustra uma atividade de exemplo que usa um executor R.

Selected Activity Info

Tag	GEN_R_BAR_PLOT
Description	Generate a bar plot using R
Type	REDUCE
Wrapper or Criteria	generate_chart
Consume	sel_001
Produce	reduce_001
Create Join	-- Select a target --

Figura 19. Atividade para o exemplo de execução de script “R”

O script “R” deverá se encarregar de gravar os dados de saída na pasta “outbox” e no arquivo “sagi-output.txt”. O arquivo “sagi_output.txt” deverá conter os dados que serão gravados na tabela de saída da atividade, no formato CSV, com os nomes das colunas coincidindo com os atributos daquela tabela. Qualquer coluna que contenha nomes de arquivos que existam na pasta “outbox” fará com que o Teapot envie estes arquivos para o Sagitarii. Esta coluna deverá estar identificada como tipo “File” na tabela de destino. A figura 20 ilustra um trecho de um script “R” gravando seus dados de saída.

Na linha 52, uma nova coluna “filename” é adicionada ao *array* de dados. Na linha 63 o nome do arquivo é criado. Na linha 64, o nome do arquivo é adicionado ao caminho completo da pasta de saída “outbox” (criado na linha 4 do script da figura 18, agregando a variável passada pelo Teapot com a pasta de trabalho

“SagitariiWorkFolder” com o nome da pasta de saída “outbox”).

Na linha 65, o nome do arquivo é colocado no CSV de saída, abaixo da coluna “filename”, criada na linha 52. A linha 66 grava então este arquivo de dados na pasta de saída.

A linha 70 termina por gravar o CSV produzido (“sagi_output.txt”) na pasta de trabalho da atividade (não na pasta “outbox”), representada pela variável “outputFile”, criada na linha 8 do script da figura 18 agregando o nome do CSV de saída “sagi_output.txt” com a variável passada pelo Teapot contendo a pasta de trabalho da atividade (“SagitariiWorkFolder”). É importante que o CSV produzido seja bem formatado no padrão RFC 4180, sendo obrigatório a presença dos nomes das colunas.

```
51 param$resultado <- NULL
52 param$filename <- NULL
53 for (i in 1:nrow(param) ) {
54   tnet <- nnet(data.train, alvo.class.train, size=param[i,"size"],
55     decay=param[i,"decay"], maxit=param[i,"maxit"])
56   pnet <- predict(tnet, data.test, type="raw")
57   roc_data <- roc(pnet, alvo.class.test)
58   tx <- unlist(slot(roc_data, "y.values"))
59
60   newTx <- gsub(",", ".", tx[2])
61
62   param$resultado[i] <- newTx
63   filename <- paste(paste("result-", as.character(i), sep=""), ".csv", sep="")
64   fullFileName <- paste(outputFileFolder, filename, sep="/")
65   param$filename[i] <- filename
66   write.table(pnet, file=fullFileName, row.names = FALSE,
67     col.names = TRUE, sep = ",", dec = ".", quote = TRUE)
68 }
69 param$dataset <- NULL
70 write.table(param, file=outputFile, row.names = FALSE,
71   col.names = TRUE, sep = ",", dec = ".", quote = FALSE)
```

Figura 20. Script “R” gravando os dados de saída

A tabela 4 resume os passos necessários para criar e executar um script R.

Tabela 4. Resumo de atividades para criar executar um script “R” no Sagitarii

Ao criar um script, utilize a variável fornecida pelo Teapot “sagitariiWorkFolder” para montar os nomes das pastas e arquivos necessários ao script.
Leia o arquivo “sagi_input.txt” que contém os dados CSV enviados pelo Sagitarii, necessários para a execução da atividade. Estes são os dados da tabela de entrada da atividade. Caso o Sagitarii tenha enviado algum arquivo de dados, eles estarão na pasta “inbox”.
Trabalhe com os dados em seu script normalmente.
Produza o CSV de saída, com os dados que deseja gravar na tabela de saída da atividade. As colunas deste CSV devem coincidir em nome e tipo com os

atributos da tabela de saída existente no banco de dados (exceto no caso colunas com nomes de arquivos, que no banco será “Integer” (File) e a coluna do CSV será “String”, mas o Sagitarii sabe como contornar isso).
Caso seu script R produza arquivos adicionais (imagens, dados compactados, áudio, vídeo, etc...) estes arquivos devem ser gravados na pasta “outbox” e referenciados em colunas no CSV de saída, sendo que estas colunas devem coincidir com os atributos tipo “File” na tabela de saída da atividade.
Grave o CSV com os dados da tabela de saída com o nome “sagi_output.txt”, na pasta de trabalho da atividade (mesmo lugar onde está o arquivo “sagi_input.txt”).
O Teapot vai abrir o CSV de saída “sagi_output.txt” e verificar se precisa enviar os arquivos da pasta “outbox”.
Ao receber o arquivo CSV e os arquivos de dados, o Sagitarii grava os arquivos de dados no repositório de arquivos, recebe seu índice e troca o nome do arquivo pelo seu índice dentro do arquivo CSV, pois o atributo tipo “File” na tabela de dados é na realidade tipo “Integer” com chave estrangeira para a tabela de repositório de arquivos.
Após gravar todos os arquivos e preparar o CSV, o Sagitarii insere as linhas do CSV na tabela de saída da atividade. Colunas existentes no CSV sem atributos correspondentes não serão gravados e atributos existentes na tabela sem correspondentes no CSV ficarão como NULL.

> Experiment files			
			Search <input type="text"/>
File ID	Task Serial	Name	Link
8	B15D7D90	figure2.png	figure2.png 
1 to 1 of 1 Records			    

Figura 21. Arquivo do gráfico cadastrado no experimento (tela de inspeção)

15. Pesquisando os resultados do experimento

Em elaboração.

Referências

OGASAWARA, E. (2011) “Uma Abordagem Algébrica para Workflows Científicos com Dados em Larga Escala”; Universidade Federal do Rio de Janeiro - Programa de Engenharia de Sistemas e Computação, Coppe/Sistemas; Tese de Doutorado, páginas 31-55.

RPROJECT, <http://www.r-project.org>. Visitado em 26 de novembro de 2014.

RJAVA, <http://cran.r-project.org/web/packages/rJava/index.html>. Visitado em 26 de novembro de 2014.

APÊNCICE “A”

Estrutura das tabelas do workflow

Os atributos `id_experiment`, `id_custom`, `id_activity`, `index_number` e `taskid` são criados automaticamente pelo Sagitarii.

Tabela “perfil_eleitorado”: recebe os dados do arquivo CSV “ Eleitorado 2014” baixado do site do STE. Serve como entrada de dados da atividade “PROJ_PER_ELEIT”.

Table "perfil_eleitorado"	
periodo	character varying
sexo	character varying
id_experiment	integer
id_custom	integer
id_activity	integer
grau_de_escolaridade	character varying
faixa_etaria	character varying
qtd_eleitores_no_perfil	integer
cod_municipio_tse	integer
uf	character varying
index_number	integer
nr_zona	integer
taskid	character varying
municipio	character varying

Tabela “perfil_eleitorado_proj”: recebe os dados de saída da atividade “PROJ_PER_ELEIT”. Serve como entrada de dados da atividade “JOIN_001”.

Table "perfil_eleitorado_proj"	
id_custom	integer
uf	character varying
index_number	integer
id_experiment	integer
qtd_eleitores_no_perfil	integer
id_activity	integer
taskid	character varying
sexo	character varying
nr_zona	integer
faixa_etaria	character varying

Tabela “detalhe_votacao_zona” : recebe os dados do arquivo CSV “Detalhe da apuração por município e zona” baixado do site do STE. Serve como entrada de dados da atividade “PROJ_DET_VOT”.

Table "detalhe_votacao_zona"	
taskid	character varying
data_ult_totalizacao	character varying
qtd_secoes_tot	integer
qtd_abstencoes	integer
codigo_cargo	integer
id_activity	integer
sigla_ue	character varying
index_number	integer
sigla_uf	character varying
qtd_aptos	integer
transito	character varying
hora_ult_totalizacao	character varying
qtd_comparecimento	integer
qtd_votos anulados_apu_sep	integer
numero_zona	integer
qtd_votos legenda	integer
qtd_votos brancos	integer
descricao_cargo	character varying
nome_municipio	character varying
id_custom	integer
hora_geracao	character varying
qtd_aptos_tot	integer
num_turno	integer
qtd_secoes agregadas	integer
data_geracao	character varying
qtd_secoes	integer
qtd_votos nominais	integer
codigo_municipio	integer
ano_eleicao	character varying
qtd_votos nulos	integer
descricao_eleicao	character varying
id_experiment	integer

Tabela “detalhe_votacao_zona_proj” : recebe os dados de saída da atividade “PROJ_DET_VOT”. Serve como entrada de dados da atividade “JOIN_001”.

Table "detalhe_votacao_zona_proj"	
id_activity	integer
qtd_comparecimento	integer
qtd_votos nominais	integer
id_experiment	integer
numero_zona	integer
taskid	character varying
index_number	integer
id_custom	integer
qtd_aptos	integer

Tabela “join_001” : recebe os dados de saída da atividade “JOIN_001”. Serve como entrada de dados da atividade “SELECT_001”.

Table "join_001"	
qtd_comparecimento	integer
taskid	character varying
qtd_apos	integer
id_experiment	integer
faixa_etaria	character varying
nr_zona	integer
numero_zona	integer
id_activity	integer
index_number	integer
sexo	character varying
qtd_eleitores_no_perfil	integer
id_custom	integer
uf	character varying
qtd_votos_nominais	integer

Tabela “sel_001” : recebe os dados de saída da atividade “SELECT_001”. Serve como entrada de dados da atividade “REDUCE_0001”.

Table "sel_001"	
qtd_votos_nominais	integer
nr_zona	integer
qtd_eleitores_no_perfil	integer
faixa_etaria	character varying
uf	character varying
taskid	character varying
qtd_apos	integer
sexo	character varying
id_experiment	integer
id_activity	integer
id_custom	integer
qtd_comparecimento	integer
numero_zona	integer
index_number	integer

Tabela “reduce_001”: recebe os dados de saída da atividade “REDUCE_001”. Contém os nomes dos arquivos compactados que foram gerados pela atividade.

Table "reduce_001"	
id_activity	integer
num_registers	integer
id_experiment	integer
filename	character varying
id_custom	integer
taskid	character varying
index_number	integer

Tabela “split_001”: recebe os dados de saída da atividade “SPLIT_001”. Contém os nomes dos arquivos compactados recebidos pela atividade e os nomes dos arquivos de seu respectivo conteúdo.

Table "split_001"	
taskid	character varying
zipfile	character varying
id_experiment	integer
contentfile	character varying
source_id	integer
id_custom	integer
source_table	character varying
index_number	integer
id_activity	integer

Apêndice “B”

Lista das atividades cadastradas e suas relações de consumo e produto

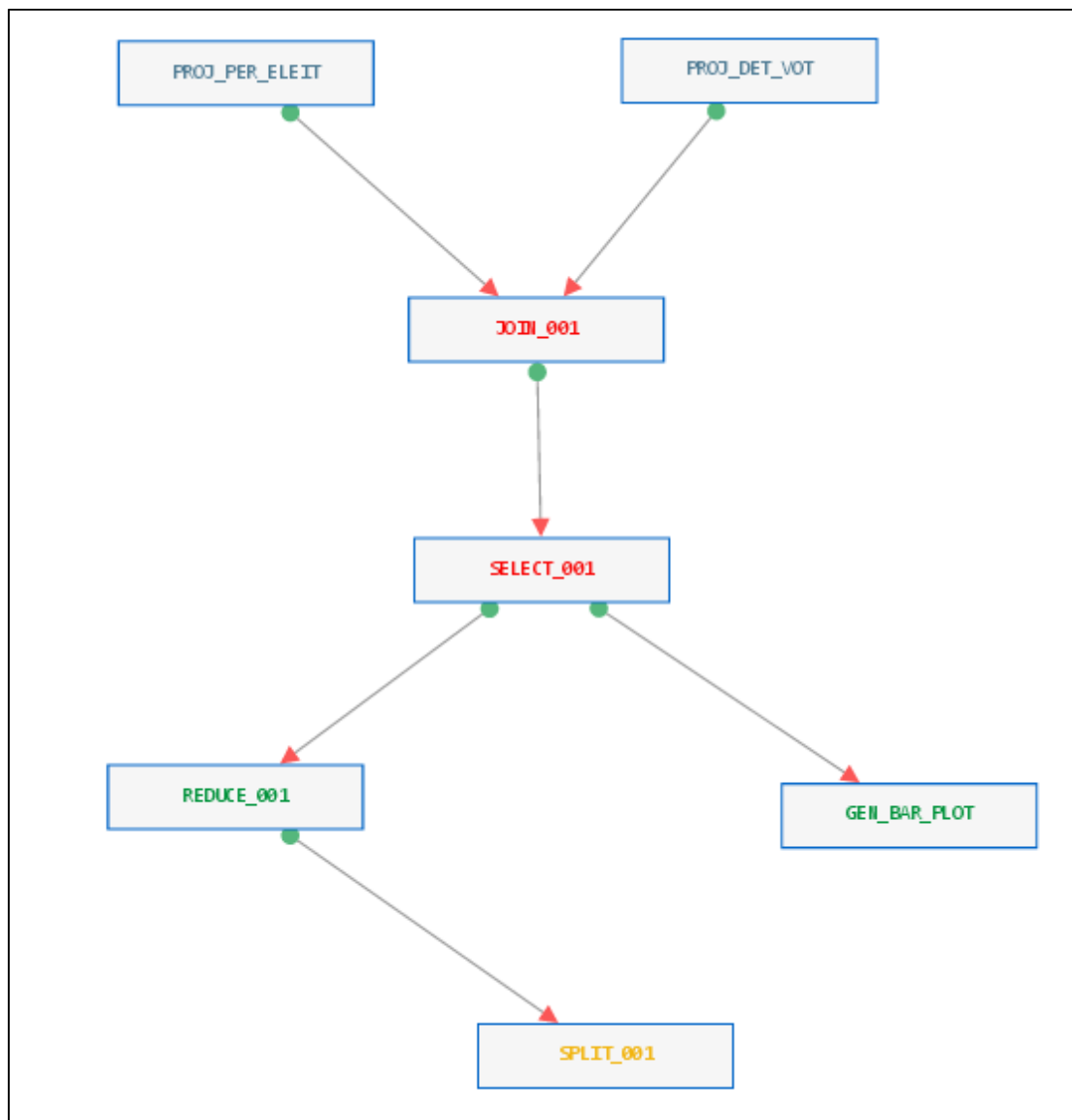
<div><div>Selected Activity Info</div><div><div>Tag</div><div>PROJ_PER_ELEIT</div></div><div><div>Description</div><div>Projeção da tabela de perfil do eleitorado</div></div><div><div>Type</div><div>MAP</div></div><div><div>Wrapper or Criteria</div><div>PROJ_PER_ELEIT</div></div><div><div>Consume</div><div>perfil_eleitorado</div></div><div><div>Produce</div><div>perfil_eleitorado_proj</div></div><div><div>Create Join</div><div>-- Select a target --</div></div></div>	<div><div>Selected Activity Info</div><div><div>Tag</div><div>PROJ_DET_VOT</div></div><div><div>Description</div><div>Projeção da tabela de detalhe da votação por zona</div></div><div><div>Type</div><div>MAP</div></div><div><div>Wrapper or Criteria</div><div>PROJ_DET_VOT</div></div><div><div>Consume</div><div>detalhe_votacao_zona</div></div><div><div>Produce</div><div>detalhe_votacao_zona_proj</div></div><div><div>Create Join</div><div>-- Select a target --</div></div></div>
<div><div>Selected Activity Info</div><div><div>Tag</div><div>JOIN_001</div></div><div><div>Description</div><div>Junção das projeções</div></div><div><div>Type</div><div>SELECT</div></div><div><div>Wrapper or Criteria</div><div>JOIN_001</div></div><div><div>Consume</div><div>perfil_eleitorado_proj detalhe_votacao_zona_proj</div></div><div><div>Produce</div><div>join_001</div></div><div><div>Create Join</div><div>-- Select a target --</div></div></div>	<div><div>Selected Activity Info</div><div><div>Tag</div><div>SELECT_001</div></div><div><div>Description</div><div>Junção das projeções</div></div><div><div>Type</div><div>SELECT</div></div><div><div>Wrapper or Criteria</div><div>SEL_001</div></div><div><div>Consume</div><div>join_001</div></div><div><div>Produce</div><div>sel_001</div></div><div><div>Create Join</div><div>-- Select a target --</div></div></div>
<div><div>Selected Activity Info</div><div><div>Tag</div><div>REDUCE_001</div></div><div><div>Description</div><div>Criação dos arquivos de saída do workflow</div></div><div><div>Type</div><div>REDUCE</div></div><div><div>Wrapper or Criteria</div><div>CONSOLIDA_DADOS</div></div><div><div>Consume</div><div>sel_001</div></div><div><div>Produce</div><div>reduce_001</div></div><div><div>Create Join</div><div>-- Select a target --</div></div></div>	<div><div>Selected Activity Info</div><div><div>Tag</div><div>SPLIT_001</div></div><div><div>Description</div><div>Expanda os arquivos ZIP e armazena os nomes dos arquivos</div></div><div><div>Type</div><div>SPLIT_MAP</div></div><div><div>Wrapper or Criteria</div><div>SPLIT_001</div></div><div><div>Consume</div><div>reduce_001</div></div><div><div>Produce</div><div>split_001</div></div><div><div>Create Join</div><div>-- Select a target --</div></div></div>

Selected Activity Info

Tag	GEN_BAR_PLOT
Description	Generate a bar plot using R
Type	REDUCE
Wrapper or Criteria	generate_chart
Consume	sel_001
Produce	sel_001
Create Join	-- Select a target -- 

Apêndice “C”

Diagrama do workflow de dados eleitorais



16. O que é um wrapper?

O Sagitarii foi concebido para executar praticamente qualquer tipo de tarefa de modo distribuído. Porém, prever quais os tipos de programas seriam executados nos nós de processamento seria uma tarefa inviável, pois comprometeria a versatilidade do Sagitarii.

Para resolver este problema, criou-se o conceito de “wrapper” (“empacotador”), que funciona como intermediário entre o Sagitarii e o programa a ser executado nas máquinas que estão processando as atividades (nós de processamento).

Quando existe uma tarefa a ser realizada (compactar arquivos, produzir gráficos, etc) o Sagitarii envia os dados da tarefa para um nó de processamento juntamente com o nome do wrapper que será encarregado de executar esta tarefa. Ao iniciar sua execução, o Teapot Cluster (programa que executa as atividades do Sagitarii nos nós de processamento) baixa do Sagitarii todos os wrappers disponíveis, assim, todas as máquinas que estão servindo ao Sagitarii como nó de processamento possuem todos os wrappers necessários nas mesmas versões.

Ao receber os dados e o nome do wrapper, o Teapot cria uma pasta exclusiva para esta instância de execução do wrapper (chamada de ativação), salva os dados que recebeu do Sagitarii para esta ativação nesta pasta e executa o wrapper, passando seu caminho completo como parâmetro ao wrapper.

O wrapper quando inicia a execução, deve recuperar o caminho de sua pasta exclusiva de seu parâmetro e carregar os dados que recebeu do Sagitarii, formatar estes dados conforme for conveniente e então iniciar o programa para qual foi criado para empacotar, passando os parâmetros de maneira adequada a este programa.

Caso os dados que recebeu do Sagitarii instrua o wrapper a usar algum arquivo (esta já deverá estar armazenado no Sagitarii), existe uma API especial no servidor que possibilita ao wrapper fazer download deste arquivo.

Quando o programa que o wrapper empacotou terminar, o wrapper deve receber os dados de saída deste programa e enviar de volta ao Sagitarii, encerrando assim a ativação.

Existem duas formas de enviar dados ao Sagitarii: enviar um arquivo diretamente ao servidor e/ou produzir dados no formato CSV. Quando um programa cria algum arquivo (imagem, arquivos compactados e arquivos binários em geral, mais ligados ao experimento propriamente dito) é necessário enviar este ao servidor. Já os dados de atividade (mais ligados ao fluxo da informação) são normalmente em formato CSV e são os dados de entrada e saída das atividades. Estes são armazenados nas tabelas de produto e consumo das atividades que executaram a ativação.

Por exemplo, suponha que seja necessário realizar um certo cálculo sobre determinada informação e o resultado deste cálculo deva ser compactado em um arquivo para ser utilizado na atividade seguinte. O Sagitarii envia os dados (oriundos da tabela de entrada da atividade) ao Teapot. Este executa o wrapper sobre estes dados, que os repassa ao programa especialista. O arquivo resultante (compactado) é enviado diretamente ao Sagitarii pelo Teapot ao final da execução do wrapper, mas o wrapper deve salvar este arquivo em sua pasta de saída (outbox), só assim o Teapot saberá que precisa enviar arquivos para o servidor. O wrapper passa os dados necessários para a

tabela de saída de atividade na forma de CSV ao Teapot apenas enviando este CSV (bem formatado) para a saída padrão (tela). Quando o wrapper termina, o Teapot lê sua saída padrão e recebe o CSV, enviando-o ao Sagitarii, que decompõe este CSV e armazena cada coluna em sua coluna correspondente na tabela de saída da atividade. Cada linha do CSV irá se tornar uma tupla nesta tabela. O Sagitarii possui capacidade para descobrir se uma coluna existe ou não na tabela que receberá o CSV, então, as colunas do CSV que existem na tabela receberão dados e as que não existem serão ignoradas. No caso do exemplo, estes dados poderiam conter o nome do arquivo produzido, para que a próxima atividade saiba como encontrá-lo mais tarde.

17. Os tipos de wrapper existentes

Embora o conceito de wrapper tenha tornado praticamente infinitas as tarefas a serem executadas pelo Sagitarii, este precisa classificar os wrappers de acordo com as atividades a serem executadas. Quando uma atividade está sendo criada, somente os wrappers correspondentes são exibidos para serem selecionados. Existem seis tipos de wrappers:

- **MAP:** Os wrappers do tipo MAP executam atividades do tipo MAP. A informação recebida do Sagitarii é um CSV contendo uma linha que representa uma tupla da tabela de entrada da atividade e a informação produzida deverá ser uma linha CSV que será inserida como uma tupla na tabela de saída da atividade. No momento do desenvolvimento de um wrapper do tipo MAP, deve-se esperar e obedecer este formato de dados.
- **REDUCE:** Os wrappers do tipo REDUCE executam atividades do tipo REDUCE. A informação recebida é composta por uma ou mais linhas CSV contendo uma seleção com distinção em cima de atributos da(s) tabela(s) de entrada da atividade (figura 1). A informação produzida deverá ser composta por um CSV contendo uma ou mais linhas que serão inseridas na tabela de saída da atividade e/ou arquivos de dados.
- **SPLIT-MAP:** Wrappers tipo SPLIT-MAP executam atividades tipo SPLIT-MAP. A informação recebida é composta por uma ou mais linhas CSV. A informação produzida deverá ser composta por um CSV contendo uma ou mais linhas que serão inseridas na tabela de saída da atividade e/ou arquivos de dados.
- **R-SCRIPT:** Os wrappers tipo R-SCRIPT são arquivos de texto contendo um script na linguagem “R”. Estes arquivos são executados por wrappers tipo R-RUNNER, que executam o processador “R” e passam o script como parâmetro. Atividades tipo MAP e REDUCE podem executar wrappers do tipo R-SCRIPT.
- **R-RUNNER:** Os wrappers tipo R-RUNNER precisam do processador “R” instalado nos nós de processamento para executar os scripts escritos usando a linguagem “R”. A instalação padrão do Sagitarii oferece um wrapper tipo R-RUNNER, não havendo necessidade de substituição ou criação de um outro wrapper.

É importante informar corretamente o tipo de wrapper ao Sagitarii, para que a formatação dos dados a serem enviados aos nós de processamento seja de acordo com a atividade que está sendo executada. Da mesma forma, é importante formatar os dados de saída de acordo com o tipo de wrapper.

Qualquer arquivo de biblioteca, necessário ao wrapper, que deva ser enviado aos nós de processamento poderá ser cadastrado como um wrapper (futuramente será criado o tipo **LIBRARY**, que não poderá ser associado a nenhuma atividade). Os nós de processamento receberão todos os wrappers cadastrados assim que iniciarem a execução do Teapot Cluster.

	SEXO	UF	QTD
	MASCULINO	RJ	300
	FEMININO	RJ	250
	MASCULINO	RJ	150
	FEMININO	SP	340
	MASCULINO	SP	134
Pipeline 1	MASCULINO	RJ	300
	MASCULINO	RJ	150
Pipeline 2	FEMININO	RJ	250
Pipeline 3	FEMININO	SP	340
Pipeline 4	MASCULINO	SP	134

Figura 1. Dados enviados para um wrapper tipo “REDUCE” (cada pipeline)

18. Como o Teapot Cluster executa um wrapper

Ao iniciar a execução, o Teapot Cluster baixa um arquivo de manifesto (em formato XML) do servidor Sagitarii. Este arquivo descreve todos os wrappers cadastrados no Sagitarii (figura 2). Após baixar o arquivo de manifesto, o Teapot inicia o download de todos os wrappers descritos no arquivo (figura 3).

```

1  <?xml version="1.0" encoding="UTF-8"?>
2  <manifest>
3      <wrapper name="CONSOLIDA_DADOS" type="REDUCE" target="ANY" version="1.0">
4          <activityFile>consolida_dados.jar</activityFile>
5          <reload>true</reload>
6      </wrapper>
7      <wrapper name="BAR_FUNCTION" type="RSCRIPT" target="ANY" version="1.0">
8          <activityFile>bar.r</activityFile>
9          <reload>true</reload>
10     </wrapper>
11     <wrapper name="SEL_001" type="SELECT" target="ANY" version="1.0">
12         <activityFile>null</activityFile>
13         <reload>true</reload>
14     </wrapper>

```

Figura 2. Trecho de um arquivo de manifesto

```

C:\Windows\system32\cmd.exe

D:\eleicoes>java -jar teapot-1.2.0-beta.jar

Sagitarii Teapot v1.2.0-beta          30/10/2014
Carlos Magno Abreu          magno.mabreu@gmail.com

-----

Carregando Repository Manager ...

Carregando Task Manager ...

Este processador possui 4 nucleos
Windows 7 em SCORPIO.home
192.168.25.27 / 00-13-46-94-18-C1-9202
Java 1.7.0_60
Familia de SO WINDOWS
Verificando a cada 700 milissegundos.
Sagitarii em http://localhost:8080/sagitarii/
Processador R em C:\rJava\jri
Nao exibir console das ativacoes.

-----

20/12/2014 21:06:18 Downloading manifest.
20/12/2014 21:06:18 Verifying wrappers...
20/12/2014 21:06:18 Check split_demo.jar 1.0 ANY
20/12/2014 21:06:18 Downloading http://localhost:8080/sagitarii/split_demo.jar
20/12/2014 21:06:18 1.0. Wait...
20/12/2014 21:06:18 split_demo.jar ok.
20/12/2014 21:06:18 Check r-wrapper.jar 1.0 ANY
20/12/2014 21:06:18 Downloading http://localhost:8080/sagitarii/r-wrapper.jar 1
20/12/2014 21:06:18 .0. Wait...
20/12/2014 21:06:18 r-wrapper.jar ok.
20/12/2014 21:06:18 Check detalhe_vot_zona.jar 1.0 ANY
20/12/2014 21:06:18 Downloading http://localhost:8080/sagitarii/detalhe_vot_zon
20/12/2014 21:06:18 a.jar 1.0. Wait...
20/12/2014 21:06:18 detalhe_vot_zona.jar ok.
20/12/2014 21:06:18 Check genchart.r 1.0 ANY
20/12/2014 21:06:18 Downloading http://localhost:8080/sagitarii/genchart.r 1.0.
20/12/2014 21:06:18 Wait...
20/12/2014 21:06:18 genchart.r ok.
20/12/2014 21:06:18 Check consolida_dados.jar 1.0 ANY
20/12/2014 21:06:18 Downloading http://localhost:8080/sagitarii/consolida_dados
20/12/2014 21:06:18 .jar 1.0. Wait...
20/12/2014 21:06:18 consolida_dados.jar ok.
20/12/2014 21:06:18 Check perfil_eleitorado.jar 1.0 ANY
20/12/2014 21:06:18 Downloading http://localhost:8080/sagitarii/perfil_eleitora
20/12/2014 21:06:18 do.jar 1.0. Wait...
20/12/2014 21:06:18 perfil_eleitorado.jar ok.
20/12/2014 21:06:18 Check bar.r 1.0 ANY
20/12/2014 21:06:18 Downloading http://localhost:8080/sagitarii/bar.r 1.0. Wait
20/12/2014 21:06:18 ...
20/12/2014 21:06:18 bar.r ok.
20/12/2014 21:06:18 Done verifying wrappers.

-----

```

Figura 3. Download dos wrappers pelo Teapot Cluster

Quando o Sagitarii envia instruções para um nó de processamento executar uma tarefa, estas instruções vêm em formato XML, contendo dados como: nome do workflow, do experimento, da atividade, o nome dos wrappers que deverão ser

executados e os dados que devem ser passados a este wrapper (figura 4). Este arquivo é chamado de pipeline.

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <pipeline
3   workflow='ANALISE_ELEITORAL'
4   experiment='C8174597-5480-470'
5   serial='0B23F3E2-725B-4'
6   fragment='5349E2A4'>
7   <activity>
8     <order>0</order>
9     <serial>94B7EF2B</serial>
10    <executorType>MAP</executorType>
11    <taskIdChain>
12    </taskIdChain>
13    <executor>PROJ_PER_ELEIT</executor>
14    <type>MAP</type>
15    <inputData>
16      grau_de_escolaridade;uf;sexo;faixa_etaria;qtd_eleitores_no_perfil
17      LÊ E ESCREVE;BA;FEMININO;70 A 79 ANOS;495
18    </inputData>
19    <sourceId>90233</sourceId>
20    <sourceTable>perfil_eleitorado</sourceTable>
21    <command>perfil_eleitorado.jar</command>
22  </activity>
23 </pipeline>
```

Figura 4. Um arquivo de execução de ativação (pipeline)

Um pipeline pode conter instruções para executar vários wrappers em sequência. Ao receber um pipeline, o Teapot identifica o nome do wrapper que deve ser executado (tags “command” e “executor”) e cria uma pasta exclusiva para esta execução, salvando os dados de entrada (tag “inputData”) em um arquivo chamado “sagi_input.txt”. Embaixo desta pasta também são criadas mais duas pastas (inbox e outbox), a pasta “inbox” serve para que o desenvolvedor do wrapper possa gravar os arquivos que porventura venha a baixar do Sagitarii. Embora seja prático possuir uma pasta exclusiva para cada instância de execução do wrapper gravar arquivos, este método não é obrigatório. Já o uso da pasta “outbox” é imperativo quando o desenvolvedor do wrapper necessita enviar os arquivos produzidos pela instância de execução (ativação) para o Sagitarii, pois ao encerrar a execução do wrapper, o Teapot Cluster avalia a pasta “outbox” e envia os arquivos ali contidos ao servidor, devidamente identificados.

Após este trabalho inicial, o wrapper é executado como um “thread” interno do Teapot Cluster. A quantidade de wrappers que são executados ao mesmo tempo é definida pelo arquivo de configuração do Teapot.

19. Parâmetros passados pelo Teapot Cluster para o wrapper

Ao iniciar a execução do “thread” do wrapper, o Teapot passa algumas informações como parâmetro (tabela 1).

Tabela 1. Parâmetros passados ao wrapper pelo Teapot Cluster

Argumento 0	Caminho da pasta exclusiva criada pelo Teapot cluster para a execução do wrapper. O arquivo <code>sagi_input.txt</code> estará nesta pasta e as pastas “inbox” e “outbox” estarão abaixo dela. O Teapot também gravará uma cópia do arquivo XML do pipeline nesta pasta.
-------------	--

20. Como receber e enviar arquivos e dados CSV

Após iniciar a execução, o wrapper deverá carregar os dados que recebeu do Teapot, enviados pelo Sagitarii, abrindo o arquivo indicado pelo primeiro parâmetro. Este arquivo está em formato CSV. A forma como este arquivo será carregado e utilizado dependerá da implementação do desenvolvedor do wrapper, que deverá respeitar o tipo de wrapper que está escrevendo.

Caso o wrapper necessite de algum arquivo (este já deverá estar devidamente armazenado no Sagitarii, seja por carga inicial ou por produto de outra atividade), deverá acessar o endereço da API de download de arquivo do Sagitarii passando o nome do arquivo desejado (figura 5). O quarto parâmetro contém o endereço do Sagitarii, bastando então adicionar o nome da API, o nome do arquivo desejado e o identificador do experimento (quinto parâmetro).

Ao encerrar a tarefa, o wrapper deverá enviar os dados produzidos ao Sagitarii. Para padronizar o trâmite de dados, o wrapper nunca envia dados diretamente ao Sagitarii, apenas com a intervenção do Teapot Cluster.

Os dados da atividade que deverão ser gravados nas tabelas de saída deverão ser enviados diretamente para a saída padrão (console, tela), pois o wrapper está sendo executado como um thread do Teapot. O Teapot receberá estes dados e enviará de forma adequada ao servidor. Os arquivos que precisam ser enviados ao servidor deverão ser gravados na pasta “outbox”. Ao encerrar o thread do wrapper, o Teapot verificará esta pasta e enviará todos os arquivos ali contidos ao servidor Sagitarii.

```

110  /**
111   * Download files from Sagitarii if you need.
112   */
113  public static void downloadFile(String fileName) throws Exception {
114      // Sagitarii server host address is always args[3]
115      // Remember, the activation working folder is always args[1].
116      // Teapot also gives you an inbox folder at args[1]/inbox so you can put
117      // your files there.
118
119      URL website = new URL( hostUrl + "/getFile?fileName="+ fileName + "&experiment=" + experimentSerial );
120      String destination = workFolder + "/inbox/" + fileName;
121      ReadableByteChannel rbc = Channels.newChannel(website.openStream());
122      FileOutputStream fos = new FileOutputStream( destination );
123      fos.getChannel().transferFrom(rbc, 0, Long.MAX_VALUE);
124      fos.close();
125  }

```

Figura 5. Baixando um arquivo do servidor Sagitarii

Caso seja necessário enviar arquivos ao servidor, normalmente os nomes dos arquivos são enviados como dados CSV, para que sejam encontrados pela próxima tarefa que receber estes dados, mas esta é uma decisão do desenvolvedor do wrapper. O importante é adicionar, como prefixo no nome do arquivo, o identificador único da tarefa (terceiro argumento) para garantir que este será mantido único. Não se deve esquecer que várias instâncias do wrapper executarão o mesmo trabalho e, apesar de poderem gravar arquivos com o mesmo nome (possuem pastas de trabalho diferentes), eles não podem enviar arquivos com mesmo nome ao Sagitarii dentro do mesmo

experimento.

21. Criando um wrapper tipo MAP

Os wrappers possuem várias características em comum, então é natural que possuam trechos de código em comum. O código de um wrapper tipo MAP e tudo o que for comum entre os vários tipos de wrappers será mostrado neste capítulo, enquanto os demais se limitarão a mostrar somente o que for específico daquele tipo.

Juntamente com o código fonte do Sagitarii existe uma classe implementando um wrapper de exemplo. Esta classe chama-se “SampleWrapper” e está no pacote “cmabreu.sagitarii.wrappers”.

Todo wrapper deve estar preparado para receber os parâmetros que serão passados pelo Teapot, então é conveniente preparar alguns atributos privados para facilitar a sua utilização. A figura 6 ilustra a preparação destes atributos.

```
public class MyMAPWrapper{
    // To hold parameters...
    private static String inputFile;        // args[0]
    private static String workFolder;       // args[1]
    private static String taskId;           // args[2]
    private static String hostUrl;          // args[3]
    private static String experimentSerial; // args[4]
```

Figura 6. Criação dos atributos do wrapper para receber os parâmetros

Na inicialização do wrapper (método *main*), estes atributos receberão os valores dos parâmetros passados pelo Teapot. Recomenda-se manter uma padronização na criação dos wrappers, então, mesmo que nem todos os parâmetros sejam utilizados, é conveniente receber todos de qualquer forma. A figura 7 mostra como receber os parâmetros do Teapot nos atributos criados.

Se for necessário receber alguma informação que não foi passada por parâmetro, é sempre possível ao programador que está criando o wrapper ler o arquivo XML de pipeline. Este arquivo é gravado pelo Teapot na pasta exclusiva da ativação (parâmetro “workFolder”) e pode ser acessado usando o caminho

`<workFolder>/sagi_source_data.xml`

Neste caso, o programador que desenvolve o wrapper fica encarregado de criar um método de acesso e interpretação do XML de pipeline (parser).

```
public static void main(String[] args) throws Exception {
    inputFile = args[0];        // CSV input data file
    workFolder = args[1];       // Working folder
    taskId = args[2];           // Task serial ID
    hostUrl = args[3];          // Sagitarii host URL
    experimentSerial = args[4]; // Experiment tag ID
```

Figura 7. Recebendo os parâmetros passados pelo Teapot Cluster

O próximo passo é carregar os dados CSV que o Teapot gravou na pasta exclusiva de execução da ativação. Este arquivo (*sagi_input.txt*) contém os dados CSV que o Sagitarii enviou ao Teapot utilizando o arquivo XML de pipeline (figura 4). O

Teapot, após processar o pipeline, separa os dados CSV e grava neste arquivo antes de iniciar uma instância do wrapper que irá processá-lo. Lembre-se de que cada instância de execução do wrapper (ativação) possui uma pasta exclusiva para trabalhar com arquivos, que é apagada após a execução do wrapper. O caminho completo e o nome do arquivo é entregue pelo Teapot no parâmetro `args[0]` (atributo *inputFile*). É conveniente criar um método separado para carregar este arquivo em uma lista de *strings* (figura 8).

```
public static List<String> readFile(String file) throws Exception {
    String line = "";
    ArrayList<String> list = new ArrayList<String>();
    BufferedReader br = new BufferedReader( new FileReader( file ) );
    if (br != null) {
        while ( (line = br.readLine() ) != null ) {
            list.add( line );
        }
        br.close();
    }
    return list;
}
```

Figura 8. Método para ler o arquivo CSV em uma lista de *Strings*

O método recebe um nome de arquivo (com caminho completo) e devolve uma lista de Strings contendo o conteúdo do arquivo (linhas CSV).

Retornando ao método de inicialização do wrapper (main), é hora de carregar o arquivo CSV usando o parâmetro passado pelo Teapot (*inputFile*). A figura 9 ilustra este processo.

```
// Read the CSV input data file
List<String> inputData = readFile( inputFile );
// If we have data...
if( inputData.size() > 0 ) {
    // we will work only if we have data!
}
```

Figura 9. Carregando os dados CSV enviados pelo Teapot Cluster

Uma vez que se possui os dados enviados pelo Sagitarii, o seu processamento depende de cada wrapper e a implementação deste processamento depende de cada programador e da tarefa que foi atribuída ao wrapper. Um wrapper tipo MAP vai receber um CSV contendo o cabeçalho com os nomes das colunas e uma linha com os respectivos valores e deverá devolver um CSV com o mesmo formato (um cabeçalho com os nomes das colunas e uma linha com os respectivos valores).

É imperativo que os nomes das colunas do CSV produzido esteja de acordo com os nomes das colunas da tabela de saída da atividade a que pertence o wrapper, sendo que a ordem das colunas é irrelevante. As colunas que existirem no CSV e não existirem na tabela de saída da atividade serão ignoradas e as colunas que existirem na tabela de saída da atividade e não existirem no CSV receberão valores nulos.

Para facilitar o processamento do arquivo CSV, é recomendado criar dois métodos para processar as colunas (cabeçalho) e a linha de dados (valores). A figura 10 ilustra a chamada ao processamento do cabeçalho do CSV (modificando o código da figura 9). Processar o cabeçalho fará sentido quando for necessário repassar ao CSV de saída (e por consequência a tabela de saída da atividade) os nomes das colunas do CSV

de entrada (e por consequência, da tabela de entrada da atividade). Normalmente isso é feito quando é necessário manter a rastreabilidade dos dados (quais os valores de entrada produziram este valor de saída). Para isso, basta replicar as colunas desejadas no CSV de saída, e, é claro, seus respectivos valores.

```
// If we have data...
if( inputData.size() > 0 ) {
    // Get the first line : the columns
    String header = inputData.get(0);
    String[] columns = header.split(";");
    // Do something with the columns
    processHeader( columns );
    ...
}
```

Figura 10. Chamando o método de processamento das colunas CSV

A parte mais relevante é o processamento das linhas de dados. Cabe ao programador decidir como trabalhar estas informações de acordo com a tarefa a ser executada pelo wrapper. No caso de um wrapper tipo MAP, os valores resultantes formarão uma linha no CSV de saída e devem estar de acordo com os tipos de dados esperados na tabela de saída da atividade. A figura 11 mostra mais uma alteração no código das figuras 9 e 10, desta vez chamando um método para processar a linha contando os dados em CSV, recebidos do Teapot. Ainda no método de inicialização do wrapper (*main*). Ainda com foco na reutilização e padronização, é conveniente tratar estes dados como se possuíssem mais de uma linha, ainda que os dados provenientes de atividades tipo MAP possuam apenas uma única linha. Na figura 11, um laço tipo “for” processa todas as linhas do arquivo CSV, neste caso, apenas uma única linha será processada. Quando for a hora de criar wrappers de outros tipos, este método de inicialização do wrapper (*main*) estará flexível o suficiente para ter um mínimo de modificações, bastando copiar e colar o código no novo wrapper.

```
// If we have data...
if( inputData.size() > 0 ) {
    // Get the first line : the columns
    String header = inputData.get(0);
    String[] columns = header.split(";");
    // Do something with the columns
    processColumns( columns );
    // If we have lines of data
    if( inputData.size() > 1 ) {
        // With each line of data...
        for( int x=1; x<inputData.size(); x++ ) {
            String[] lineData = inputData.get(x).split(";");
            // ... do something
            processLine( lineData );
        }
    }
}
```

Figura 11. Processando as linhas do arquivo CSV

Um wrapper vai produzir, invariavelmente, um conjunto de dados no formato CSV e/ou arquivos a serem enviados ao Sagitarii. Embora não seja uma regra, os dados

no formato CSV normalmente são tratados como dados de fluxo do experimento (são obtidos de tabelas e serão armazenados em tabelas durante o processamento do experimento). Já os arquivos normalmente são tratados como dados de resultado, pertencentes ao experimento, e serão gravados em local apropriado para serem acessados pelo responsável pelo experimento.

Um wrapper entrega os dados CSV ao Teapot (para ser enviado ao Sagitarii) através da saída padrão do sistema operacional (neste caso, a tela). Isto significa que um wrapper deverá literalmente “escrever” na tela o conteúdo CSV que for produzido.

Por conveniência, é recomendado criar um atributo privado e global para o wrapper manter os dados que forem produzidos para a saída. A figura 12 ilustra uma modificação do código da figura 6 (criação de atributos globais para o wrapper) contendo a declaração deste atributo.

```
public class MyMAPWrapper {
    // To hold parameters...
    private static String inputFile;        // args[0]
    private static String workFolder;       // args[1]
    private static String taskId;           // args[2]
    private static String hostUrl;          // args[3]
    private static String experimentSerial; // args[4]
    // The output CSV data holder ( index 0 = header )
    private static List<String> outputData = new ArrayList<String>();
}
```

Figura 12. Um atributo para manter os dados CSV de saída

A medida que os dados de saída forem produzidos, estes deverão ser armazenados no atributo “*outputData*”, que é uma lista de *Strings*, sendo que a primeira linha desta lista deverá ser a lista de colunas do CSV, separadas por ponto e vírgula, e as demais os dados, também separados por ponto e vírgula.

Uma vez produzidos os dados, é hora de enviá-los ao Teapot, para que este os envie ao Sagitarii. Para tanto, é necessário “escrever” o conteúdo do atributo “*outputData*” na tela. A figura 13 mostra a sugestão para um método que faz este trabalho.

```
/**
 * Print out the outputData content to screen
 * ( send to Teapot by using standard out )
 */
private static void printOutput() {
    for ( String line : outputData ) {
        System.out.println( line );
    }
}
```

Figura 13. Método para enviar os dados CSV de saída ao Teapot

Mais uma vez vamos modificar o código do método de inicialização do wrapper (*main*) para chamar o método de envio dos dados de saída, após processar as colunas e as linhas (no caso do MAP, uma única linha). A figura 14 mostra a alteração no código da figura 11.

```

// If we have data...
if( inputData.size() > 0 ) {
    // Get the first line : the columns
    String header = inputData.get(0);
    String[] columns = header.split(";");
    // Do something with the columns
    processColumns( columns );
    // If we have lines of data
    if( inputData.size() > 1 ) {
        // With each line of data...
        for( int x=1; x<inputData.size(); x++ ) {
            String[] lineData = inputData.get(x).split(";");
            // ... do something
            processLine( lineData );
        }
    }
    // Print out the outputData content to screen ( send to Teapot by using standard out )
    printOutput();
}

```

Figura 14. Método “main” : processando colunas, linhas e enviando a saída

Como o atributo “*outputData*” é global, os métodos de processamento de colunas e linhas podem gravar dados diretamente nele, tendo o cuidado de inserir primeiro as colunas (chamar “processColumns”) e depois as linhas (chamar “processLine”).

Vale ressaltar que os métodos “processColumns” e o método “processLine” devem ser implementados de acordo com a necessidade e exigência da tarefa a ser executada pelo wrapper, sendo esta a razão por não terem sido exibidos neste guia. O método “processColumns”, por exemplo, pode nem ser necessário, já que as colunas a serem produzidas são de prévio conhecimento do programador (são as mesmas da estrutura da tabela de saída da atividade que executou o wrapper). Neste caso, a primeira linha do atributo “*outputData*” (lista de “*Strings*”), que representa as colunas do CSV, pode ser produzida “manualmente” por concatenação, já no início do método “main”.

É importante observar a ordem dos dados produzidos e suas respectivas colunas no atributo de saída. Quando os dados forem produzidos no método “processLine”, estes devem ser concatenados na mesma ordem em que as colunas foram criadas. Por exemplo, supondo que a primeira linha do atributo “*outputData*” contenha as seguintes colunas CSV:

nome;rua;idade

Quando o método “processLine” receber os dados de entrada, processá-los e for concatenar a linha de saída para inserir no atributo “*outputData*”, deverá colocar os dados na sequência:

Carlos Abreu;Rua Jardim;42

Respeitando assim a ordem das colunas, o que é importante para permitir que o Sagitarii insira corretamente os dados na tabela de saída da atividade. Note que a tabela de saída, no exemplo, espera dados nos tipos String para o campo “nome”, String para o

campo “rua” e Integer para o campo “idade”. Inverter a ordem destes campos fará com que a inserção dos dados gere um erro de SQL no Sagitarii, o que causará a perda dos dados e por consequência, uma possível interrupção na execução do experimento. Na melhor das hipóteses (dados invertidos com mesma tipagem) o experimento irá fornecer um resultado incorreto.

Por conveniência, como um wrapper do tipo MAP recebe uma linha de dados e produz uma linha de dados, não será considerado neste capítulo o recebimento e envio de arquivos. O recebimento de arquivos será mostrado no capítulo sobre wrappers do tipo SPLIT e o envio de arquivos será coberto no capítulo sobre wrappers do tipo REDUCE.

Para testar o wrapper, crie um arquivo CSV com dados de teste e execute o wrapper passando os parâmetros esperados na ordem exigida (neste caso, forneça o seu arquivo e crie parâmetros falsos para os demais). O wrapper deverá escrever na tela um CSV no formato correto: primeira linha para as colunas CSV e demais linhas para os dados, separados por ponto e vírgula e sem separação por aspas. Cada nova linha usando o caractere “NEWLINE” (\n). Além disso, é imperativo poder executar o wrapper usando o mesmo formato de chamada que o Teapot usa (sem fornecer o *classpath* ou outros parâmetros que não os já mencionados):

```
java -jar MyMAPWrapper.jar /home/user/mycsvfile.csv aa bb cc dd
```

Tendo o wrapper funcionando corretamente, ele já poderá ser cadastrado no Sagitarii para ser associado à uma atividade.

22. Criando um wrapper tipo SPLIT

Um wrapper tipo SPLIT receberá um CSV com uma linha de dados e deverá produzir um CSV com uma ou mais linhas de dados e/ou um ou mais arquivos a serem enviados ao Sagitarii. Caso o arquivo de dados CSV contenha nomes de arquivos a serem processados, estes deverão ser baixados do Sagitarii.

Em primeiro lugar, será coberto o caso mais simples: o recebimento de um CSV contendo uma linha de dados e o envio de um CSV contendo uma ou mais linhas de dados.

Neste caso, a única modificação em relação ao exposto no capítulo anterior é que a lista de Strings contendo os dados de saída (atributo “*outputData*”) conterá mais de uma linha de dados. Já deixamos o método “*printOutput*”, que envia o conteúdo de “*outputData*” para o Teapot, pronto para enfrentar esta situação (figura 13).

Um outro caso seria a utilização de algum arquivo para processar (descompactar, por exemplo). Neste caso, o nome deste arquivo viria no CSV de entrada e seria necessário baixá-lo do servidor Sagitarii. Este arquivo deverá estar armazenado no Sagitarii seguindo dois caminhos: a) foi enviado pelo responsável pelo experimento, fazendo parte dos dados iniciais (consulte o Guia do Usuário para saber como enviar arquivos ao Sagitarii); b) foi enviado por uma das atividades anteriores. Nesse caso, consulte o capítulo sobre criação de wrappers tipo “REDUCE” para saber como um wrapper envia arquivos para o Sagitarii. O Sagitarii armazena os arquivos no escopo do experimento, o que significa que cada experimento deve possuir arquivos com nomes únicos. Uma maneira de garantir que os arquivos sejam únicos é adicionar o número de série da tarefa ao nome do arquivo. O número de série da tarefa é passado ao wrapper

pelo Teapot usando o terceiro parâmetro.

A não existência do arquivo solicitado causará erro na execução da atividade (representada pela execução de todas as suas ativações) e por consequência, do experimento.

Supondo que o método “processLine”, descrito no capítulo anterior, identifique o nome do arquivo necessário a ser utilizado pelo wrapper, o próximo passo é acessar a API de fornecimento de arquivos do Sagitarii. Esta API responde no endereço:

http://<sagitariihost>/getFile?fileName=<file_name>&experiment=<exp_id>

O endereço do servidor Sagitarii já é fornecido pelo Teapot através do quarto parâmetro, que foi armazenado no atributo “hostUrl” e o número de série do experimento foi passado pelo quinto parâmetro e armazenado no atributo “experimentSerial” (figura 7). A figura 15 ilustra um método para realizar o download de um arquivo do Sagitarii, que podemos acrescentar à classe do wrapper que já foi criado no capítulo anterior ou criar um novo wrapper copiando o código do primeiro.

```
/**
 * Download files from Sagitarii if you need.
 */
public static void downloadFile(String fileName) throws Exception {
    // Sagitarii server host address is always args[3]
    // Remember, the activation working folder is always args[1].
    // So, you must use args[1]/<file_name> to save your downloaded file.

    URL website = new URL( hostUrl + "/getFile?fileName=" + fileName + "&experiment=" + experimentSerial );
    String destination = workFolder + "/inbox/" + fileName;
    ReadableByteChannel rbc = Channels.newChannel(website.openStream());
    FileOutputStream fos = new FileOutputStream( destination );
    fos.getChannel().transferFrom(rbc, 0, Long.MAX_VALUE);
    fos.close();
}
```

Figura 15. Método para baixar arquivos do Sagitarii

Observando o método na figura 15 percebe-se a utilização do atributo “workFolder”, que representa o segundo argumento passado ao wrapper pelo Teapot, contendo o caminho completo da pasta exclusiva criada para a execução do wrapper. O Teapot, por conveniência, já fornece a pasta “inbox” abaixo da pasta da ativação (“workFolder”), então ficará mais prático que os arquivos baixados do Sagitarii sejam gravados nesta pasta, pois ao término da execução da instância do wrapper, tudo será apagado.

Uma vez de posse do arquivo, o wrapper poderá processar e enviar o resultado de volta ao Sagitarii, seja por dados CSV (como mostrado no capítulo anterior) ou enviando arquivos (como será mostrado no próximo capítulo). A figura 16 ilustra um exemplo do método “processLine”, que recebe uma linha de dados do arquivo CSV, identifica o nome do arquivo, faz o download chamando o método da figura 15 e chama um suposto método de descompactação “splitFile” (considerando que o arquivo em questão é um arquivo compactado).

```

public static void processLine( String[] lineData ) {
    // get the filename as first element in the list
    String fileName = lineData[0];
    try {
        // download it from Sagitarii
        downloadFile( fileName );
        // uncompress contents in outbox folder
        // (will send contents back to Sagitarii !)
        splitFile( fileName, workFolder + "/outbox");
    } catch ( Exception e ) {
        e.printStackTrace();
        // Exit code will remain in table data
        // so we will know something was wrong
        System.exit(1);
    }
}
}

```

Figura 16. Exemplo de processamento de uma linha que envolve um download

Na figura 16 pode-se perceber a chamada ao método “splitFile” passando como parâmetro a pasta exclusiva seguida de “outbox”, que é a pasta de saída do wrapper. Qualquer arquivo existente nesta pasta após o encerramento do wrapper será enviado ao Sagitarii dentro do escopo do experimento, então tudo que este exemplo faz é baixar um arquivo compactado, descompactar e enviar seu conteúdo de volta ao Sagitarii.

A figura 17 mostra mais uma alteração no método “main” (exibido pela última vez na figura 14). Desta vez, a inclusão das colunas CSV (linha 0 da lista de saída “outputData”) é feita “manualmente”, informando que a saída será um CSV contendo o nome do arquivo compactado e seu respectivo conteúdo (obviamente o nome do arquivo compactado irá se repetir para todas as linhas, mas isso fará sentido quando todos os dados estiverem juntos e for necessário saber de onde veio cada arquivo). Ainda no método “main” (figura 17), nota-se a chamada ao método “processLine” para cada linha do CSV de entrada (provavelmente apenas uma no tipo SPLIT). Em “processLine” (figura 16) baixamos o arquivo que identificamos como o elemento zero na linha CSV processada e descompactamos seu conteúdo. O apêndice “B” mostra o código do método “splitFile”, onde pode-se notar a inclusão das linhas de dados propriamente ditas na lista de saída “outputData”, respeitando a ordem das colunas, já inseridas na lista (figura 17). Então o método “main” encerra enviando todo o conteúdo de “outputData” para o Teapot (figura 13), que o envia ao Sagitarii.


```

// if we have data...
if( inputData.size() > 0 ) {
    // prepare output CSV header
    // ( must match activity destination table schema in Sagitarii )
    outputData.add("zipfile;contentfile");

    // If we have lines of data
    if( inputData.size() > 1 ) {
        // With each line of data...
        for( int x=1; x<inputData.size(); x++ ) {
            String[] lineData = inputData.get(x).split(";");
            // ... do something
            processLine( lineData );
        }
    }
    // Print out the outputData content to screen
    // ( send to Teapot by using standard out )
    printOutput();
}

```

Figura 17. Criação manual do cabeçalho do CSV de saída

23. Criando um wrapper tipo REDUCE

Em elaboração.

24. Criando um wrapper tipo RRUNNER

Em elaboração.

25. Cadastrando arquivos de suporte (bibliotecas) para os wrappers.

Em elaboração.

26. Cadastrando o wrapper no Sagitarii

Apêndice “A”: Tabela descritiva do arquivo de instância

pipeline	Elemento raiz do documento XML. Possui os atributos “workflow”, “experiment”, “serial” e “fragment”. Cardinalidade: um.
activity	Contém os dados para uma ativação. Elementos filhos: “order”, “serial”, “executorType”, “taskIdChain”, “executor”, “type”, “inputData”, “sourceId”, “sourceTable” e “command”. Sem atributos. Filho de “pipeline”. Cardinalidade: um ou muitos.
order	Número inteiro contendo a ordem de execução da atividade. Sem atributos. Filho de “activity”. Cardinalidade: um.
serial	Alfanumérico representando o número de série da atividade. Filho de “activity”. Sem atributos. Cardinalidade: um.
executorType	Tipo do executor que processará os dados. Enumerado. Filho de “activity”. Sem atributos. Cardinalidade: um.
taskIdChain	Alfanumérico que representa o ID da primeira tarefa que foi executada antes desta. O ID da tarefa é repassado entre os pipelines para manter o rastreamento do processamento. Sem atributos. Filho de “activity”. Cardinalidade: um.
executor	Alfanumérico com o nome (alias) do executor como cadastrado no Sagitarii. Sem atributos. Filho de “activity”. Cardinalidade: um.
type	Enumerado. Tipo da atividade associada a este pipeline. Sem atributos. Filho de “activity”. Cardinalidade: um.
inputData	CSV. Dados a serem repassados ao wrapper para processamento. Sem atributos. Filho de “activity”. Cardinalidade: um.
sourceId	Número inteiro (ou vazio quando for REDUCE). ID da tupla (linha) de origem dos dados de entrada contidos na tag “inputData”, na tabela de entrada da atividade. Sem atributos. Filho de “activity”. Cardinalidade: um.
sourceTable	Alfanumérico com o nome da tabela de entrada da atividade de onde os dados contidos na tag “inputData” foram retirados. Para saber a origem destes dados, executar “ <i>select * from <sourceTable> where id_custom = <sourceId></i> ”. Sem atributos. Filho de “activity”. Cardinalidade: um.
command	Alfanumérico com o nome do arquivo java do wrapper que processará os dados. Sem atributos. Filho de “activity”. Cardinalidade: um.

Apêndice “B”: Método splitFile()

```
/**
 * Uncompress a ZIP file
 *
 * @param zipFile
 * @param outputFolder
 */
public static void splitFile( String zipFile, String outputFolder ) {
    byte[] buffer = new byte[1024];
    try {
        File folder = new File( outputFolder );
        if( !folder.exists() ){
            folder.mkdirs();
        }
        ZipInputStream zis = new ZipInputStream(new FileInputStream( workFolder + "/inbox/" + zipFile ));
        ZipEntry ze = zis.getNextEntry();
        while(ze!=null) {
            String fileName = ze.getName();
            outputData.add(zipFile + ";" + fileName);
            File newFile = new File(outputFolder + File.separator + fileName);
            new File(newFile.getParent()).mkdirs();
            FileOutputStream fos = new FileOutputStream(newFile);
            int len;
            while ((len = zis.read(buffer)) > 0) {
                fos.write(buffer, 0, len);
            }
            fos.close();
            ze = zis.getNextEntry();
        }
        zis.closeEntry();
        zis.close();
    } catch ( IOException ex ){
        ex.printStackTrace();
    }
}
```

A API do Sagitarii recebe uma requisição na URL /externalApi do tipo POST contendo um atributo do tipo STRING de nome externalForm, contendo uma estrutura chave/valor no formato JSON. É necessário requisitar um token de segurança antes de qualquer operação.

URL da API	/externalAPI
Tipo de acesso	POST (formulário HTML)
Nome Atributo / Tipo	externalForm / String
Valor	Estrutura JSON no formato { “chave1” : ”valor1”, “chaven” : ”valorn” }
Retorno	String
<pre><html> <form action='http://www.sagitarii.server.org/externalAPI'> <input type='text' name='externalForm' value='{ "SagitariiApiFunction": "apiGetToken", "user": "john", "password": "doo" }'> </form> </html></pre>	

27. apiGetToken

Autentica na API e recebe um token de segurança para ser usado nas demais operações da mesma seção. O token é descartado após 30 minutos de inatividade.

Atributo	Valor
SagitariiApiFunction	apiGetToken
user	Usuário do Sagitarii
password	Senha do usuário
<pre>{ "SagitariiApiFunction": "apiGetToken", "user": "john", "password": "1234" }</pre>	

Retorno: uma String contendo o token de segurança a ser utilizado nas próximas chamadas.

28. apiCreateExperiment

Cria um novo experimento a partir de um workflow já existente.

Atributo	Valor
SagitariiApiFunction	apiCreateExperiment
workflowTag	Tag do workflow existente
securityToken	Token de segurança
<pre>{ "SagitariiApiFunction": "apiCreateExperiment", "workflowTag": "SPECTRAL_PORTAL", "securityToken": "b4abc9870b2b41bd91e16250cdc76465" }</pre>	

Retorno: uma String contendo o número de série (identificador) do experimento criado.

29. apiStartExperiment

Executa um experimento já existente.

Atributo	Valor
SagitariiApiFunction	apiStartExperiment
experimentSerial	Número de série do experimento
securityToken	Token de segurança
<pre>{ "SagitariiApiFunction": "apiStartExperiment", "experimentSerial": "A39DC069-1726-482", "securityToken": "b4abc9870b2b41bd91e16250cdc76465" }</pre>	

Retorno: uma String contendo o número de série (identificador) do experimento.

30. apiReceiveData

Insere dados em uma tabela no contexto de um experimento.

Atributo	Valor
SagitarIiApiFunction	apiReceiveData
experimentSerial	Número de série do experimento
securityToken	Token de segurança
tableName	Nome da tabela
Data	Array de dados no formato JSON
<pre>{ "SagitarIiApiFunction": "apiReceiveData", "tableName": "spectral_parameters", "experimentSerial": "A39DC069-1726-482", "securityToken": "b4abc9870b2b41bd91e16250cdc76465", "data": [{ "adjacency": "on", "laplacian": "on", "slaplacian": "on", "optiFunc": "\lambda", "caixa1": "min", "gorder": "2", "minDegree": "4", "maxDegree": "7", "triangleFree": "on", "allowDiscGraphs": "on", "biptOnly": "on" }, { ... }, { ... }, { ... }] }</pre>	

Retorno: uma String contendo o número de série da instância de carga gerada e o número de série da atividade de carga gerada, separadas por ponto e vírgula (;). Os valores devem ser enviados entre aspas, mas devem possuir o formato (tipo) esperado pelo atributo de mesmo nome na tabela de destino. No exemplo acima, “totalAge” deverá ser do tipo “Integer” na tabela “spectral_parameters” e “adjacency” deverá ser do tipo “String”.

31. apiGetFilesExperiment

Retorna arquivos produzidos por um experimento. É necessário informar a tag da atividade que produziu os arquivos, bem como o registro inicial e o final (é obrigatório paginar o resultado). O Sagitarii enviará os nomes e índices dos arquivos já gravados pelo experimento mesmo durante a execução do mesmo.

Atributo	Valor
SagitariiApiFunction	apiGetFilesExperiment
experimentSerial	Número de série do experimento
activityTag	Tag da atividade produtora
rangeStart	Paginação: início
rangeEnd	Paginação: fim
securityToken	Token de segurança
<pre>{ "SagitariiApiFunction": "apiGetFilesExperiment", "experimentSerial": "A39DC069-1726-482", "activityTag": "GENFORMULA", "rangeStart": "90", "rangeEnd": "100", "securityToken": "b4abc9870b2b41bd91e16250cdc76465" }</pre>	

Retorno: uma String contendo o número de série (identificador) do experimento e a lista de todos os arquivos produzidos até o momento no formato JSON, com o nome do arquivo como chave e o ID do arquivo como valor.

32. apiCreateTable

Cria uma tabela no Sagitarii.

Atributo	Valor
SagitariiApiFunction	apiCreateTable
tableName	Nome da tabela a ser criada
tableDescription	Descrição da tabela a ser criada
Nome do atributo 1	Tipo do atributo 1
Nome do atributo n	Tipo do atributo n
<pre>{ "SagitariiApiFunction": "apiCreateTable", "tableName": "my_new_table", "tableDescription": "A new table", "my_field_integer": "INTEGER", "my_field_string": "STRING", "my_field_float": "FLOAT", "my_field_date": "DATE", "my_field_time": "TIME", "my_field_file": "FILE", "my_field_stext": "TEXT" }</pre>	

Retorno: String