Gerenciamento das Atividades de um Workflow Científico: um Estudo de Caso com o Portal Fisiocomp

Bruno Archetti dos Santos

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Orientador: Prof. Ciro de Barros Barbosa

Juiz de Fora, MG

Dezembro de 2009

Bruno Archetti dos Santos

Gerenciamento das Atividades de um Workflow Científico: um Estudo de Caso com o Portal Fisiocomp

Monografia submetida ao corpo docente do Departamento de Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora, como parte integrante dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Ciro de Barros Barbosa

JUIZ DE FORA - MG
DEZEMBRO, 2009

Gerenciamento das Atividades de um Workflow Científico: um Estudo de Caso com o Portal Fisiocomp

Bruno Archetti dos Santos

Monografia submetida ao corpo docente do Departamento de Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora, como parte integrante dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em _____ de _____ de 2009.

Comissão Examinadora:

_____

Ciro de Barros Barbosa orientador.
DSc em Ciência da Computação University of Twente, UT, Holanda/2001

_____

Regina Maria Maciel Braga Villela
DSc em Engenharia de Sistemas e Computação COPPE/UFRJ

_____

Rodrigo Weber dos Santos
DSc em Matemática Physikalisch-Technische Bundesanstalt, P.T.B., Alemanha/2004

JUIZ DE FORA
DEZEMBRO, 2009

Resumo

Este trabalho tem por objetivo apresentar uma linguagem para definição de processos em workflow científico e sua integração em um ambiente de simulação de experimentos científicos. Os experimentos científicos, cada vez mais necessitam de um controle sistemático, reprodutível e documentado. Sendo assim, os sistemas de gerência de workflow científico são apontados como o principal recurso e solução para os problemas em questão. Portanto, este trabalho descreve os principais conceitos e requisitos dos workflows científicos e apresenta algumas tecnologias de apoio para a gerência de workflow, como por exemplo, a linguagem BPEL. Por fim, são detalhados os passos realizados para a elaboração da aplicação desenvolvida.

Abstract

This paper has the purpose of presenting a programming language for the definition of a process in a scientific workflow, and its integration with a simulation environment of scientific experiments. A correct scientific experiment requires a systematic, documented and repeatable control. Thus scientific workflow management systems are main solution for these problems. With that purpose, this research describes the main concepts and requirements for scientific workflows and presents some support technologies for the workflow management, for example, the BPEL language. Finally, the steps required for the preparation of the developed applications are detailed.

Agradecimentos

Agradeço a Deus. Aos meus pais, irmão e amigos pelo apoio e incentivo. Ao meu amigo e colega de sala Mateus pelo conhecimento passado em Java e Servlets. Agradeço ao meu orientador pela ajuda para completar mais este desafio.

Sumário

Lista de Figuras

Lista de Reduções

AGOS	*API Generator for ODE Solution*
API	*Application Program Interface*
BPEL	*Business Process Execution Language*
BPEL4WS	*Business Process Execution Language for Web Services*
BPMN	*Business Process Modeling Notation*
EDO	Equações Diferenciais Ordinárias
GPL	*General Public License*
IDE	*Integrated Developement Environment*
OASIS	*Organization for the Advancement of Structured Information Standards*
PVI	Problema de Valor Inicial
SCUFL	*Simple Conceptual Unified Flow Language*
SOAP	*Simple Object Access Protocol*
SWC	Sistema de Workflow Científico
SWI	*Scientific WorkflowInterpreter*
UDDI	*Universal Description, Discovery and Integration*
URL	*Uniform Resource Locator*
WfMC	*The Workflow Management Coalition*
WS-BPEL	*Web Services - Business Process Execution Language*
WSDL	*Web Service Definition Language*
WSFL	*Web Services Flow Language*
XML	*Extensible Markup Language*
XScufl	*XML Simple Conceptual Unified Flow*

Capítulo 1 – Introdução

A Ciência da Computação está revolucionando a forma como as pesquisas do século 21 estão sendo conduzidas. A busca pelo conhecimento faz com que instituições de pesquisa procurem formas de melhorar a qualidade dos experimentos científicos e reduzir o tempo necessário para sua execução. A utilização de recursos computacionais auxiliam os cientistas a combinarem os recursos científicos, como programas e dados, para executar uma função de alto nível. Neste caso, os programas e dados são combinados em uma cadeia de execução de forma que uma saída de um processamento serve como entrada para o seguinte. As comunidades científicas desenvolvem suas aplicações em diversos componentes que são combinados para o processamento dos dados. Dentro deste cenário, pode-se dizer que as aplicações científicas são vistas como workflows científicos (MEDEIROS, 1995).

Um workflow pode ser caracterizado como um fluxo de encadeamento de tarefas e suas dependências para execução. Em um workflow científico, os componentes específicos são os dados e programas. Os programas, em geral, processam grandes conjuntos de dados, principalmente em áreas como a biologia. Portanto, um problema que pode surgir durante a execução de um workflow científico é o eventual excesso de tempo necessário para finalizar o seu processamento. A execução de alguns programas pode ser demorada acarretando conseqüentemente um tempo elevado para processar o workflow como um todo (VIVACQUA, 2006). Assim, o paralelismo pode ser utilizado para melhorar o desempenho desses programas e de workflows científicos em geral. Dentre as possibilidades para explorar o paralelismo podemos citar a utilização de máquinas multiprocessadoras, explorar um ambiente distribuído como os Clusters ou a utilização de um ambiente de Grid. Este último vem emergindo como plataformas para alto desempenho e para integração de recurso em rede. Um Grid, é administrado por organizações independentes em um ambiente onde recursos heterogêneos e distribuídos podem ser compartilhados e agregados para formar um supercomputador virtual (VIVACQUA, 2006).

Para a criação e execução dos workflows científicos são usados os SWC (Sistemas de Workflow Científico). Estes sistemas são capazes de definir, criar e gerenciar a execução de fluxos de trabalho através do uso de software. Os SWC atenuam os cenários improdutivos, especialmente em se tratando de experimentos complexos, que podem envolver muitos programas, grande quantidade de dados e diversos cientistas em localidades geograficamente dispersas. Os SWC apóiam a execução dos workflows de modo

controlado e documentado, possibilita a reutilização de workflows previamente concebidos por outros cientistas, e a coleta de informações que permitam identificar a proveniência dos dados gerados pela execução dos workflows científicos (MATTOSO, 2009). As funcionalidades dos sistemas não são suficientes por si só, pois os cientistas tipicamente não possuem muitos conhecimentos em desenvolvimento de software e, desta forma, precisam de ferramentas que simplifiquem a especificação e execução de experimentos (DIGIAMPIETRI, 2007).

Um exemplo de ferramenta para suporte a experimentação científica baseada em modelagem computacional é o Portal Fisiocomp (FISIOCOMP, 2009). Essa ferramenta, disponível em ambiente Web, permite que seus usuários executem simulações de modelos computacionais, construídos em um formato padronizado e já difundido, e também o compartilhamento de tais modelos com outros pesquisadores. Esse Portal disponibiliza também recursos que auxiliam no manuseio dos artefatos usados e gerados em tais simulações. Essa ferramenta, entretanto, não dispõe de recursos para controlar a execução combinada de diversos modelos, a exemplo do que é preconizado em SWCs.

Para suprir a falta de recursos sobre o controle de execução das funcionalidades no Portal Fisiocomp, implementamos um interpretador chamado de SWI (*Scientific Workflow Interpreter)*, cujo objetivo, além de sua integração com o Portal, é analisar e executar os comandos definidos para a orquestração dos serviços.

1.1 – Motivação e Objetivos

O desenvolvimento de pesquisas científicas, seja qual for a área de conhecimento, depende de soluções computacionais para serem realizadas. Os recursos computacionais facilitam o compartilhamento dos dados e serviços em tempo real, possibilitando a cooperação de diversos grupos científicos separados geograficamente.

O objetivo é compreender e implantar no Portal Fisiocomp os mecanismos utilizados pelos cientistas para automatizar os procedimentos experimentais do ambiente científico. Para tanto, será feito uma análise das principais características de um workflow científico, além de conhecer e analisar linguagens e ferramentas que auxiliam na construção e execução dos workflows científicos.

Para alcançar os objetivos propostos estamos propondo a construção de um interpretador para uma linguagem que especifica a composição das funcionalidades que integram um experimento dentro do Portal. Tais linguagens são conhecidas como scripts e

sua interpretação acarreta a execução das funcionalidades, resultando na automação desejada para os experimentos. O interpretador proposto deverá se adequar as características da arquitetura de software empregada no desenvolvimento do Portal. Exemplos de funcionalidades a serem automatizadas, nesse contexto são: uploads dos modelos matemáticos, a simulação dos modelos e a visualização de resultados.

1.2 – Organização da Monografia

O presente trabalho está dividido da seguinte forma, além desta introdução: No capítulo 2 apresentamos os principais conceitos relacionados a workflow, introduzindo o conceito de workflow científico e apresentando os sistemas de workflow científico Kepler e Taverna. O capítulo 3 apresenta duas tecnologias de apoio para a gerência de workflow, o uso de BPEL e Scripts. No capítulo 4 são apresentadas as ferramentas para serem usadas no Portal: AGOS e JynaCore API. O capítulo 5 descreve o nosso trabalho, desde seu desenvolvimento à sua funcionalidade. E finalmente, no capítulo 6 avaliamos o conteúdo apresentado neste trabalho.

Capítulo 2 - Descrição e Conceitos Introdutórios

Esta seção apresenta os principais conceitos relacionados a Workflow e os Sistemas de Workflow Científico: Kepler e Taverna.

2.1 – Workflow

Atualmente experimentos científicos desenvolvidos com uma grande quantidade de dados vêm necessitando de um controle sistemático, reutilizável e documentado. A tecnologia denominada Workflow, adaptada para o contexto científico, sem dúvida é uma solução para tais experiências.

Segundo (Araújo e Borges, 2001) essa tecnologia teve origem em meados de 1970 associada ao processo de automação de escritórios. Portanto, ela não é uma exclusividade no ambiente científico, sendo que já existe uma enorme discussão na literatura, sobre o tema, e desenvolvimento de ferramentas gerenciais e automações de processos de negócio.

Em 1993 foi criado um consórcio que visa definir padrões, conceitos e interfaces aplicados aos sistemas de workflow, chamado WfMC (*Workflow Management Coalition)*. De acordo com o Modelo de Referência de Workflow (WRM, 1995) especificado pelo WfMC, workflow é:

"A automação de um processo de negócios, por inteiro ou uma parte, durante o qual documentos, informações e tarefas são passadas de um participante para outro por ação respeitando um conjunto de regras procedimentais."

No entanto essa definição é restrita a Workflows de negócios. Uma definição mais simples, genérica, e que exemplifica tanto atividades comerciais quanto científicas, foi apresentada por (Moro, 2001):

"*Qualquer tarefa executada em série ou em paralelo por dois ou mais membros de um grupo de trabalho visando um objetivo comum*".

Essa definição pode ser representada na figura 2.1. Pois temos um conjunto de atividades a serem executadas (em série e paralelo), suas relações de dependência, dados de entrada e dados de saída (MACHADO, 2007).

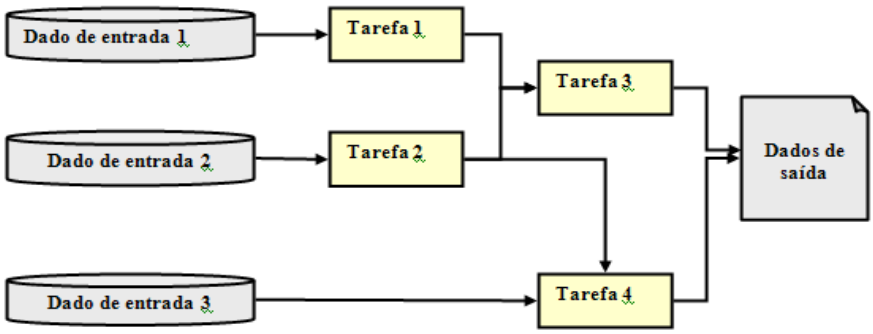

Figura 2.1- Representação de um workflow (MACHADO, 2007).

2.2 – Workflow Científico

Diversas comunidades científicas como físicos, astrônomos, biólogos, geólogos entre outras vêm desenvolvendo suas aplicações através da combinação de diferentes programas e dados formando os chamados workflows científicos (VIVACQUA, 2006).

O workflow científico é o principal recurso do experimento científico, pois ele representa a definição da orquestração de sequências de processos que manipulam dados de modo a construir uma simulação (MATTOSO, 2009). Eles guardam um correlação muito estreita com os experimentos científicos podendo até mesmo serem confundidos (MATTOS, 2008).

Para garantir recursos computacionais e humanos e otimização do tempo dos cientistas, os resultados produzidos pelo workflow devem ser armazenados. Através da análise dos mesmos, é possível confirmar as hipóteses postuladas no início do experimento ou até mesmo descobrir erros na execução total ou parcial do workflow. Além disso, é possível garantir a reprodutibilidade e reuso dos dados em outros experimentos (MATTOS, 2008).

O ciclo de vida de um workflow científico é representado pelo ciclo de vida de um experimento que está sendo reproduzido, confirmando suas semelhanças como dito anteriormente. Segundo (MOREAU, 2003), o ciclo de desenvolvimento de um workflow pode ser dividido em seis etapas.

1. Criação do Experimento: neste passo é construído o workflow que representa

o experimento.

2. Personalização: nesta etapa é registrado qualquer ponto relevante para o experimento.

3. Execução do Experimento: nesta fase ocorre a execução de cada uma das atividades do workflow. Esta execução é realizada por um sistema de workflow que será visto mais adiante.

4. Gerenciando o Experimento: nesta etapa é verificado o que ocorre durante a execução do experimento como disparo de avisos sobre os eventos e/ou geração de exceções.

5. Compartilhar Resultados: aqui ocorre a publicação do workflow gerado para que possa ser avaliado e utilizado em outros experimentos.

6. Descobrir e Reutilizar o Experimento: fase onde é descoberto um experimento previamente produzido e incorporá-lo a outro experimento.

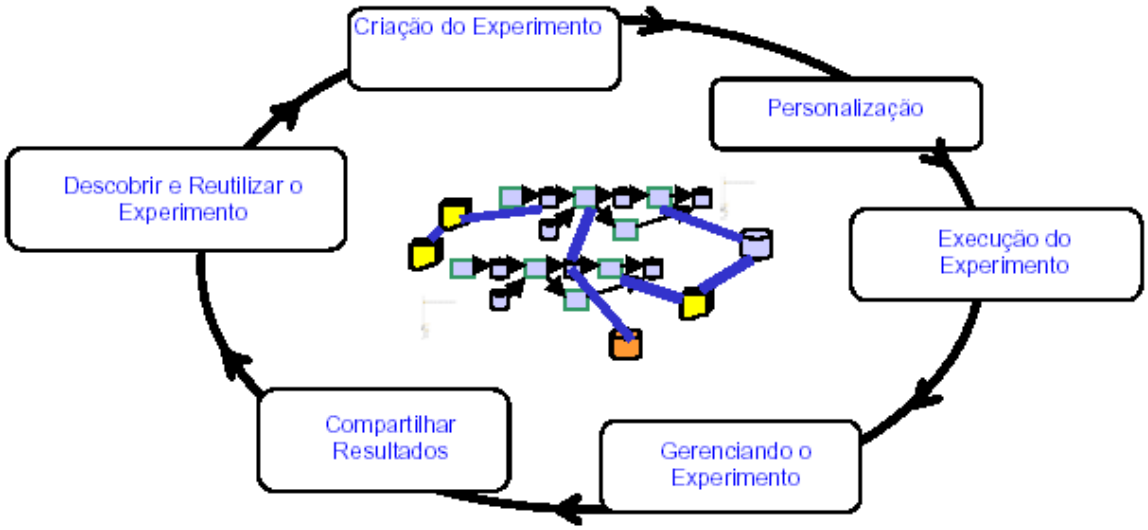

Figura 2.2 - Ciclo de vida de um workflow científico (MATTOS, 2008).

Dentre outros estudos sobre o ciclo de vida de um workflow científico, destaca-se o de (MATTOSO, 2009), que, para simplificar o ciclo, agrupou estas etapas em três fases: composição, execução e análise. Dessa forma é possível fazer uma analogia entre o experimento e um sistema computacional de forma mais genérica.

2.3 – Sistemas de Workflow Científico

Os SWC (Sistemas de Workflow Científico) são ferramentas computacionais que visam facilitar a criação, a execução e o gerenciamento de experimentos científicos. Segundo o WfMC, citado acima, é possível definir um sistema de workflow científico como (WfMC, 2008):

"*Sistemas para a definição, criação e gerência da execução de fluxos de trabalho através do uso de software, capazes de interpretar a definição de processos, interagir com seus participantes e, quando necessário invocar ferramentas e aplicações.*"

Todas as aplicações que rodam sobre os SWC são experimentos científicos que requerem alto poder computacional. Os SWC são responsáveis por invocar as aplicações (locais ou externas) que participam de um workflow científico, passando os dados pertinentes entre elas. Geralmente eles oferecem para os usuários interfaces gráficas que permitem não só a definição conceitual do workflow, como também permite o monitoramento de sua execução (MATTOS, 2008).

De acordo com (LEMOS, 2004), um SWC deve seguir uma lista de requisitos. Dentre estes, os que merecem destaque são:

1. Processos, Dados e Recursos. O SWC deve incluir os processos, dados e recursos normalmente usados e oferecer mecanismos de extensibilidade para acomodar novos processos, dados e recursos. Entre os processos estão os programas (locais ou remotos) que filtram/transformam os resultados de outros programas, além de mecanismos de controle de execução do workflow.

2. Definição. O SWC deve auxiliar os cientistas na definição e redefinição do workflow. A redefinição é um passo importante e sempre usado quando os resultados finais não forem considerados úteis ou interessantes pelos pesquisadores.

3. **Validação.** O SWC deve oferecer ferramentas de validação do workflow definido pelo cientista. Durante a validação, o sistema deve verificar se as entradas e saídas definidas pelos usuários para cada processo do workflow são consistentes, além de incluir, caso seja necessário, processos que façam conversão nos formatos dos dados e processos que verifiquem se os resultados gerados pelos programas são esperados ou não. Este requisito é importante, pois se uma entrada for enviada com valor diferente do esperado, possivelmente implicará no mau funcionamento de outros programas do workflow.

4. Otimização, Monitoramento e Execução. O SWC deve ser capaz de otimizar e executar o workflow definido pelo pesquisador, de acordo com a arquitetura que está sendo utilizada. A execução do workflow pode ser monitorada e deve permitir a intervenção do

pesquisador em qualquer ponto. A intervenção (interrupção temporária) é necessária caso ele queira avaliar os resultados intermediários para decidir se continua ou não a execução, ou para fazer alguma modificação na definição das próximas atividades do workflow.

5. Agendamento. O SWC deve oferecer agendamento da execução do workflow. O cientista ou membros de sua equipe podem desejar executar o workflow de tempos e tempos ou em uma única vez durante a semana.

6. Dados e Metadados - O SWC deve armazenar tanto os dados produzidos pelo workflow quanto os metadados. Metadado é comumente definido como "dado sobre dado", ou ainda, de forma mais completa, como uma informação sobre o dado que permite o acesso e gerenciamento deste dado de maneira eficiente e inteligente (SUMPTER, 1994). O sistema deve ser capaz de gerar estes metadados automaticamente e, sempre que possível, oferecer subsídios para que o cientista consulte-os e atualize-os.

Outra taxonomia para as funcionalidades de workflows pode ser encontrada em (BONIFÁCIO, 2008). Nas próximas seções veremos dois exemplos de SWC: Kepler e Taverna.

2.3.1 – Kepler

Kepler é um sistema de gerenciamento de workflows desenvolvido pelas Universidades da Califórnia de San Diego e Santa Bárbara. O sistema fornece uma interface gráfica que permite que o cientista defina e execute seu workflow. Kepler é um software open-source e roda nas plataformas Windows, Linux e Machintosh. O projeto Kepler conta com a participação dos seus usuários para auxiliar a adaptação dos projetos nas necessidades reais dos cientistas. Para instalar o Kepler é preciso ter o Java 1.5 ou superior instalado (KEPLER, 2009).

O sistema utiliza o conceito de ator (*actor*), que na interface gráfica do Kepler, é um componente que pode ser arrastado e incluído no workflow. Cada passo do experimento é representado pelo ator e ao conectá-los uns aos outros é formado o workflow, permitindo os cientistas analisarem os dados, modificar os parâmetros e caso necessário re-executar o experimento (KEPLER, 2009).

Existem também workflows aninhados denominados de atores compostos, ou seja, um workflow pode conter subworkflows para executar uma tarefa. Outro benefício do Kepler é a capacidade do reuso de parte ou de todo o workflow em um projeto diferente. No

Kepler é possível que o usuário modele, analise e mostre os dados por meio de uma interface simples. Através da inclusão ou exclusão de componentes na área de criação de workflows e a conexão dos componentes para construir um fluxo de dados, é criado seu próprio workflow (KEPLER, 2009).

O Kepler possui diversos componentes padrões incorporados, incluindo componentes matemáticos, estatísticos, processamento de sinais e componentes para inserção, manipulação e exibição dos dados. Estes componentes facilitam usuários iniciantes para a construção de workflows, pois eles podem ser modificados para atender a uma requisição específica, ou podem representar a exata necessidade do usuário no momento. Para cada workflow diferente que for aberto ou criado o Kepler abre uma nova janela, permitindo aos usuários trabalharem com múltiplos workflows simultaneamente e comparar, copiar e colar componentes entre eles (KEPLER, 2009).

A maioria dos atores são processos Java que rodam localmente, porém também podem ser usados serviços web. O Kepler provê acesso a repositórios de dados, recursos computacionais e bibliotecas de workflows distribuídos geograficamente (KEPLER, 2007). A partir da URL (*Uniform Resource Locator*) da descrição de um serviço web, um ator genérico de serviço web do Kepler é capaz de instanciar operações particulares especificadas na descrição do serviço. Depois da instanciação, este ator pode ser incorporado a um workflow científico como se ele fosse um componente local. Os dados de entrada e saída do serviço web, especificados na descrição WSDL (*Web Service Definition Language*) do serviço, são representados pelas portas de dados de entrada e de saída do ator (MACHADO, 2007).

Os principais componentes para a construção de um workflow no Kepler são: atores (*actors*), diretores (*directors*), parâmetros (*parameters*), relações (*relations*) e portas (*portas*) (KEPLER, 2009).

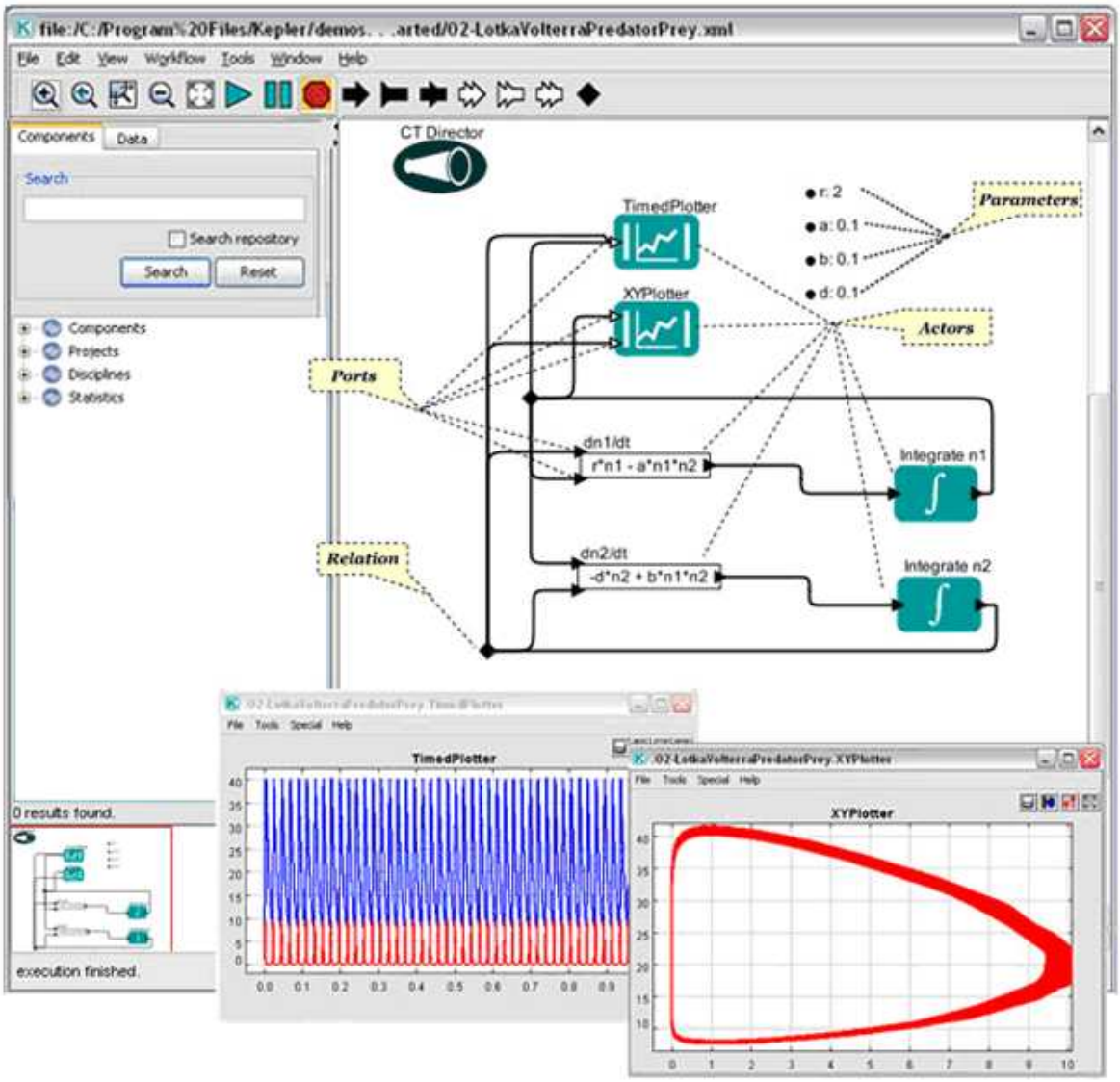

Figura 2.3 - Janela principal do Kepler com alguns dos principais componentes do fluxo de trabalho em destaque (KEPLER, 2009).

- ***Actor* –** Os atores são as construções básicas de um workflow. Ele é projetado para executar uma tarefa específica dentro do fluxo de trabalho. Kepler vem com mais de 350 atores prontos para serem utilizados, porém, novos atores podem ser adicionados para uso exclusivo do usuário.
- ***Directors*** - O diretor controla a execução de um workflow, assim como em um filme, o diretor supervisiona seu elenco ou equipe. Para cada projeto deverá ter um diretor que controla a execução do workflow usando um determinado modelo de computação. Cada modelo de execução no Kepler é representado pelo seu próprio diretor.

- ***Parameters*** **–** Os parâmetros são valores configuráveis que podem ser conectados a um workflow ou aos atores e diretores individualmente. Os parâmetros do modelo podem ser ajustados em tempo de execução.
- ***Relations*** **-** As relações permitem os usuários ramificar um fluxo de dados, e assim enviar informações para dois lugares simultaneamente dentro do workflow.
- ***Portas*** **–** Cada ator em um fluxo de trabalho pode conter uma ou mais portas usadas para consumir ou produzir dados e se comunicar com outros atores inseridos no workflow. As portas podem ser classificadas em três tipos: *input port*, para os dados consumidos pelo ator; *output port*, para os dados produzidos pelo ator; *input/output port*, para os dados que são tantos consumidos e produzidos pelo ator.

2.3.2 – Taverna

Outra opção para a criação, execução e gerenciamento de workflow científico é o Taverna, que foi criado para o projeto *myGrid* no Reino Unido. Este sistema foi planejado para trabalhar principalmente com projetos científicos do domínio de bioinformática e, portanto, ele fornece um conjunto de ferramentas para composição e execução de workflows de bioinformática (TAVERNA, 2009).

O Taverna é um ambiente integrado com a linguagem SCUFL (*Simple Conceptual Unified Flow Language*) e com a máquina de execução FreeFluo. O FreeFluo é uma ferramenta Java para orquestração de workflows e é responsável pela invocação dos serviços e pela transferência de dados intermediários. O Taverna é composto por três painéis principais (FISHER, 2009).

- *Available Services Panel*: Exibe uma lista de serviços web para o usuário. Nessa lista contém os serviços default gerados pelo Taverna na sua inicialização e também podem ser adicionados outros serviços próprios. Cada um destes serviços podem ser incluídos no workflow de modo que uma tarefa possa ser alcançada. Estes serviços podem ser locais ou remotos.

- *Advanced Model Explorer*: É uma visão geral do workflow para o usuário em forma de árvore. Neste painel contém os serviços utilizados no workflow atual, incluindo as entradas, saídas e as ligações entre cada serviço.

- *Workflow Diagram*: Mostra uma representação gráfica do workflow construído. Pode ser adaptado para visualizar os diferentes aspectos do workflow, o usuário pode optar por mostrar todas as portas de todos os serviços, ou somente as portas que possuem conexão, e mudar o layout do workflow de retrato para paisagem. Além disso, auxilia na validação do workflow criado.

A figura 2.4, apresenta uma visão geral dos três painéis principais que constituem a interface gráfica do Taverna.

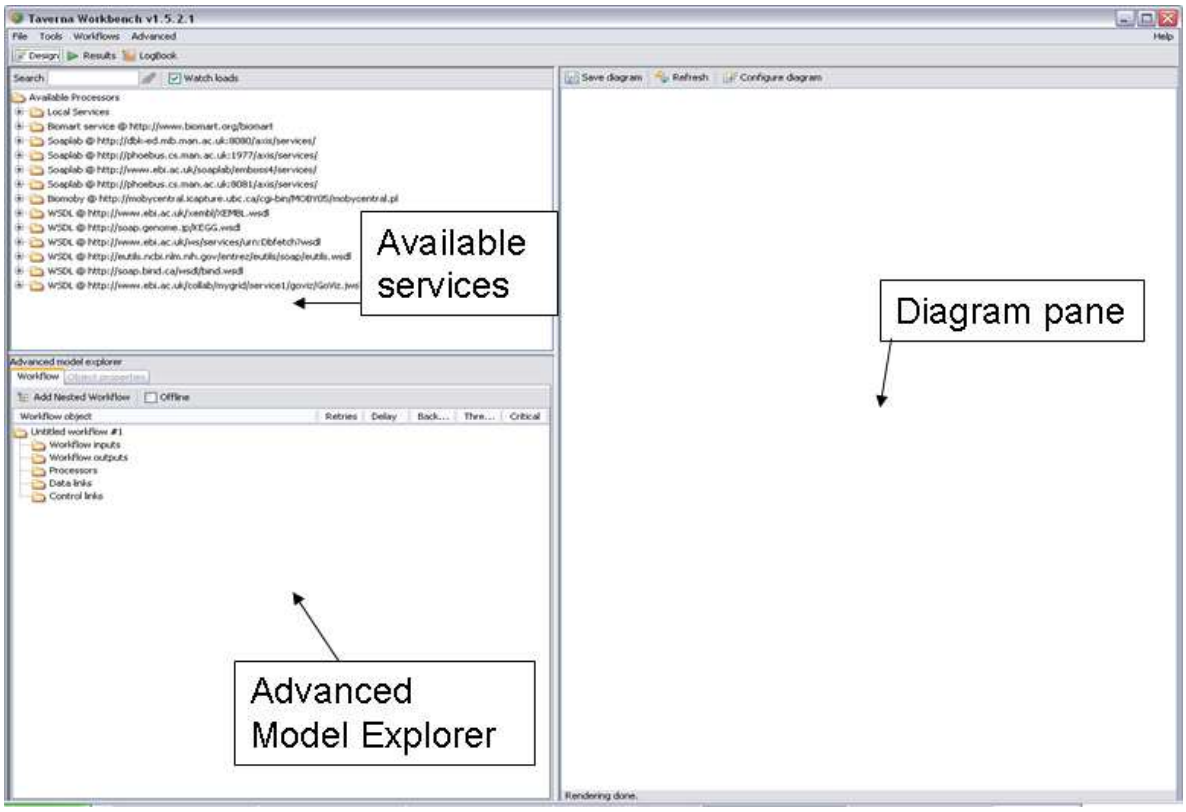

Figura 2.4 – Interface gráfica do Taverna (FISHER, 2009).

Outro painel relevante integrado com o Taverna é o *Enactor invocation panel.* Nele é possível visualizar o passo em que o workflow está sendo executado, se houve falha ao executar alguma etapa, além de apresentar o resultado final da execução do workflow e seus resultados intermediários (MATTOS, 2008).

Capítulo 3 - Tecnologias de Apoio para Gerência de Workflow

Esse capítulo apresenta duas tecnologias de apoio para a gerência de workflow relacionadas com o nosso trabalho, o uso de BPEL e Scripts.

3.1 - Business Process Execution Language – BPEL

BPEL (*Business Process Execution Language*) já é o padrão estabelecido como a linguagem para a especificação de processos de negócio, além disso, o BPEL vem ganhando espaço na especificação de processos científicos (Emmerich, 2006).

WS-BPEL é a sigla de abreviação para *Web Services Business Process Execution Language* que como o nome diz, é uma linguagem de especificação de processos de negócios que possuem seus comportamentos baseados em Web Services. Popularmente encontramos muito mais a abreviação da sigla: BPEL (LAWISCH, 2008). Neste trabalho será usado esta sigla para representar a linguagem.

O passo mais crítico no ciclo de vida de um processo de negócio é a especificação de um processo para uma plataforma que possa orquestrar um fluxo e executar as várias tarefas em um processo. Orquestrar um processo exige muitos requisitos técnicos, os quais incluem sistemas heterogêneos, padrões de mensagens síncronos e assíncronos, manipulação de dados, coordenação de fluxos e usuários, gerenciamento de exceções, eventos não determinados, gerenciamento de transações, versões, entre outros. O objetivo do padrão BPEL é prover uma abstração simples e rica para atender estes requisitos. O BPEL está rapidamente, tornando-se de fato um padrão na indústria para orquestração e execução de processos. (Building Flexible Enterprise Process Using Oracle Business Rules and BPEL Process Manager, 2005).

WS-BPEL teve a primeira especificação em julho de 2002, titulada de BPEL4WS 1.0 (*Business Process Execution Language for Web Services*) e proposta pela BEA, IBM e Microsoft, onde também foram implementadas idéias da mais antiga especificação de fluxos de processos de negócio até então usada, a WSFL (*Web Services Flow Language*) mantida pela IBM e outras idéias que vieram da XLANG da Microsoft (DIETER KÖNIG 1, 2006).

A especificação seguinte foi a BPEL4WS 1.1 que saiu em maio de 2003 e da qual teve a proposta revisada e submetida para a OASIS (*Organization for the Advancement of Structured Information Standards)*. Esta teve contribuições adicionais de empresas como a

SAP e a Siebel (LAWISCH, 2008). A versão 2.0 foi aprovada pela OASIS em abril de 2007 e é a que vigora até a presente data. Um ponto a favor da BPEL é o apoio recebido das indústrias: IBM, Microsoft, Oracle, BEA, SAP, Siebel e todas as outras que estão no consórcio OASIS (TADEU, 2009).

A última e atual especificação é a WS-BPEL 2.0 que está sob controle da OASIS, hoje possuidora de um comitê com 280 sócios e observadores onde cerca de 30 recebem convocações semanais (LAWISCH, 2008).

BPEL é uma linguagem orientada a gráficos, devido a sua natureza baseada em XML (*Extensible Markup Language*), permite que a lógica dos processos seja editada por ferramentas visuais. (NAKASHIMA, 2008).

Pode ser usado dentro de uma corporação (intranet) para padronizar a integração das aplicações corporativas e estender a integração de sistemas isolados. Ou pode ser usado entre as corporações (internet), possibilitando fácil integração entre parceiros (TADEU, 2009).

3.1.1 – Estrutura da linguagem BPEL

Abaixo serão mostrados os principais elementos e estruturas que são especificados no BPEL. Na figura 3.5 estão apresentados os elementos que serão abordados.

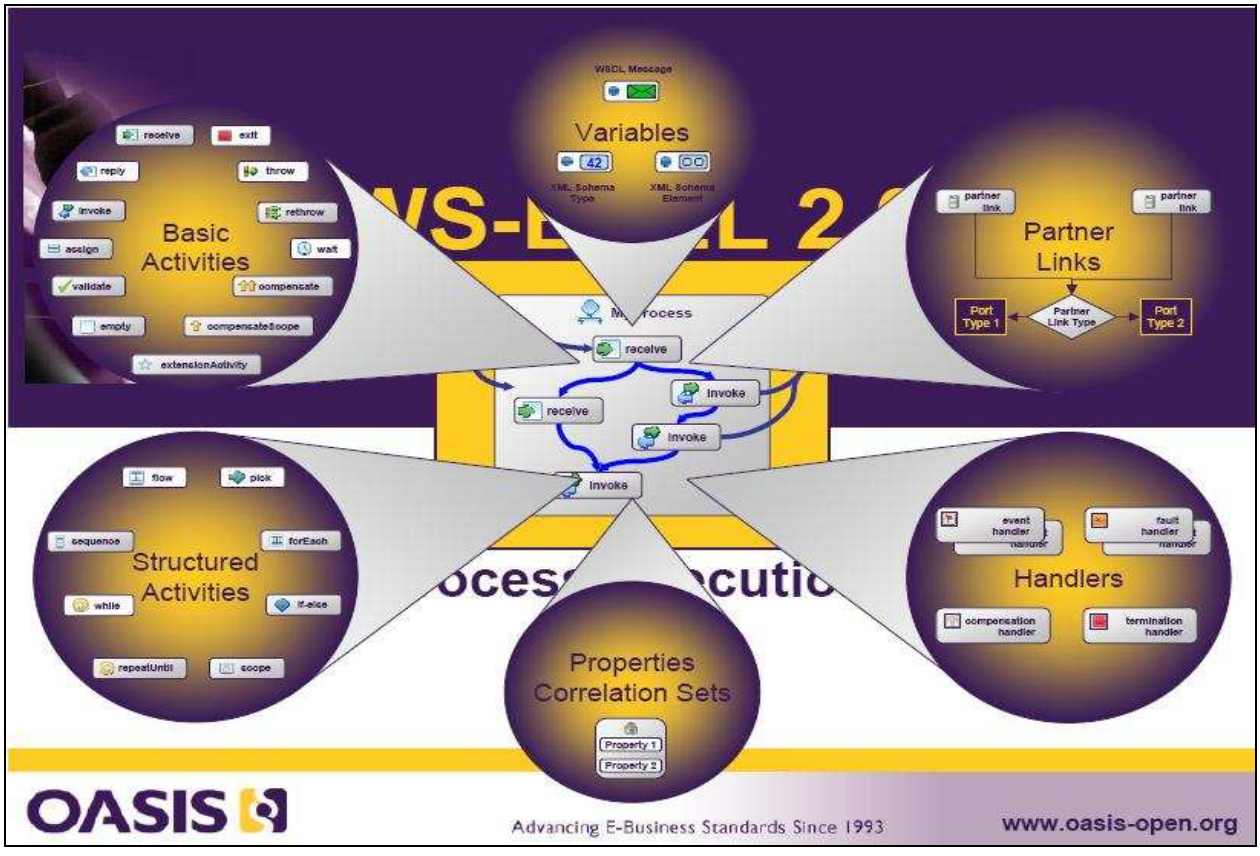

Figura 3.5 - Principais elementos do BPEL (König, 2007).

3.1.1.1 *Partner Links*

Um *Partner link* é um canal onde é realizada uma comunicação com um parceiro através de *peer-to-peer*, servindo como uma espécie de conector. Neles são definidos os roles (funções/cargos) participantes do processo que serão mapeados originalmente dos *PortTypes* declarados no WSDL (KÖNIG, 2007).

3.1.1.2 *Variables*

São utilizadas para troca de mensagens entre os Web Services e os processos BPEL. Elas representam os estados e resultados obtidos em cada passo do processo que poderão ser armazenadas num contexto global até o término do mesmo, estes serão mapeados a partir das mensagens definidas no WSDL (LAWISCH, 2008). São através das variáveis que os dados serão guardados e/ou transportados.

3.1.1.3 *Properties and correlation sets*

Durante o seu ciclo de vida, a instância de um processo de negócio pode estabelecer diversas conversações com os parceiros envoltos no processo. Em vários casos as conversações envolvem mais de duas partes, nestes casos é necessário que tenha um mecanismo para que mensagens e conversas casem com as instâncias de processos de negócio com as quais se destinam (TADEU, 2009). Usando `<correlation>` é possível transportar *tokens* que ajudam na identificação das mensagens e das conversas (BPEL, 2007).

3.1.1.4 Atividades Básicas e Estruturadas

As atividades são divididas em duas categorias: básicas e estruturadas. As atividades básicas são aquelas que descrevem os passos elementares do comportamento do processo, já as estruturadas codificam a lógica do fluxo. De acordo com a especificação de BPEL fornecida pelo OASIS, as atividades básicas são (BPEL, 2007):

- ***Invoke*** – A atividade `<invoke>` é usada para chamar um serviço web fornecido por um provedor de serviços.

- ***Receive*** – `<receive>` especifica o partnerLink usado para receber mensagens. Bloqueia e espera até que a mensagem adequada chegue.
- ***Reply*** – A atividade `<reply>` envia uma mensagem de resposta para alguma mensagem antes recebida.
- ***Assign*** – `<assign>` é utilizado para atualizar valores de variáveis e partnerLinks.
- ***Throw*** – A atividade `<throw>` gera um erro dentro de um processo de negócio**.**
- ***Wait*** - A atividade `<wait>` cria um *delay*, até que certo tempo tenha passado ou até uma data limite.
- ***Empty*** – `<empty>` é uma atividade que não faz nada. Pode ser usado como um ponto de sincronização ou para pegar uma excessão.
- ***ExtensionActivity*** – A atividade `<extensionActivity>` é o local onde podem ser incluídas novas atividades que não estão na especificação da linguagem, ou seja, é um envelope para extensão de linguagens.
- ***Rethrow*** – `<rethrow>` encaminha um erro para um tratamento de erro.
- ***Exit*** – A atividade `<exit>` é usada para parar imediatamente a instância do processo de negócio.

Figura 3.6 - Atividades Básicas do BPEL (König, 2007).

De acordo com a especificação de BPEL fornecida pelo OASIS, as atividades estruturadas são (BPEL, 2007):

- ***Flow*** – `<flow>` contém atividades que são executadas em paralelo.
- ***Sequence*** – A atividade `<sequence>` contém atividades que são executadas sequencialmente em ordem léxica.
- ***If*** – `<if>` e `<elseif>` provê um comportamento condicional ao processo, podendo decidir por um caminho ou outro.
- ***While*** – `<while>` permite uma atividade ser executada mais de uma vez, esta atividade é repetida até que uma `<condition>` seja verdadeira no início da iteração.
- ***RepeatUntil*** – A atividade `<repeatUntil>` permite uma atividade ser executada mais de uma vez, esta atividade é repetida até que uma `<condition>` seja verdadeira no fim da iteração.
- ***Pick*** – A atividade `<pick>` bloqueia e espera por uma mensagem adequada chegar (ou intervalo).
- ***ForEach*** – `<forEach>` contém atividades executadas sequencialmente ou em paralelo, controlado por uma variável contadora. O valor dessa variável é escolhido pelo desenvolvedor.
- ***Scope*** – `<scope>` contém atividades com suas variáveis locais, tratamentos de erros, manipulador de compensação, e eventos manipuladores.

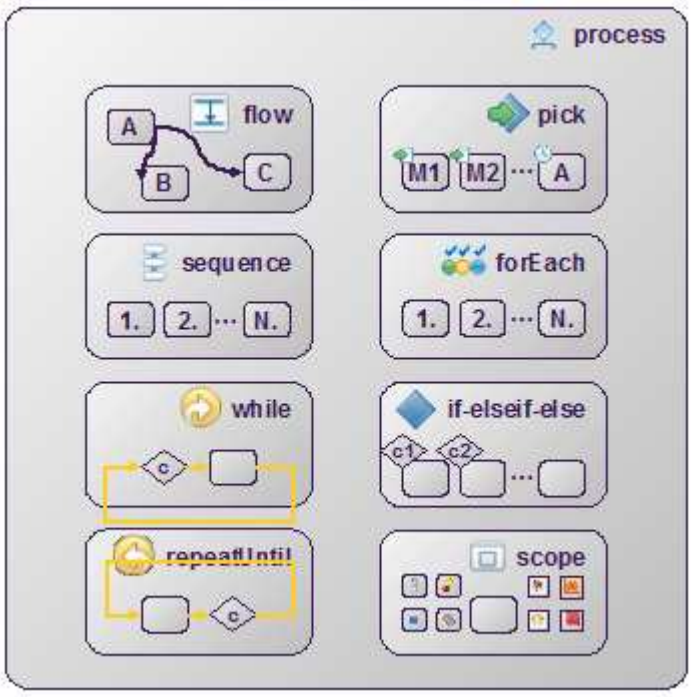

Figura 3.7 – Atividades Estruturadas do BPEL (König, 2007).

Na figura abaixo, temos uma visualização de como ficariam as atividades básicas e estruturadas construídas de forma aninhada.

```
<sequence>
   <receive .../>
   <flow>
      <sequence>
         <invoke .../>
         <while ... >
            <assign>...</assign>
         </while>
      </sequence>
      <sequence>
         <receive .../>
         <invoke ... />
      </sequence>
   </flow>
   <reply .../>
</sequence>
```

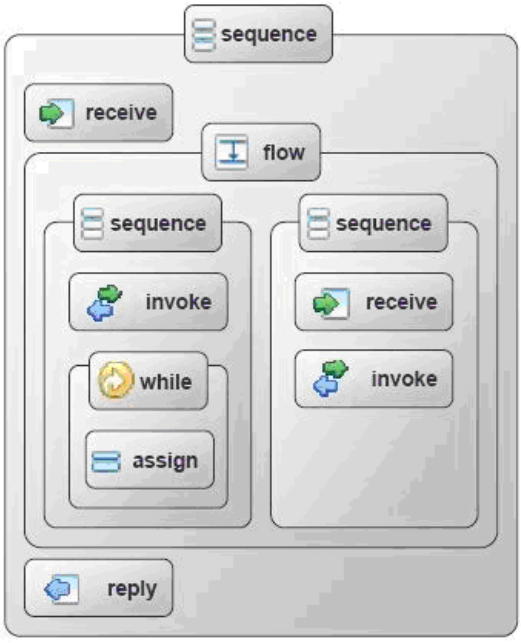

Figura 3.8 - Representação gráfica (BPMN) vs. Representação XML (BPEL) (König, 2007).

3.1.1.5 *Scopes and Handlers*

Os escopos fornecem o contexto que vai influenciar no comportamento das atividades inclusas. Este contexto comportamental inclui variáveis como *partner links, correlation sets, event handlers, message exchange handlers, fault handlers, compensation handler e termination handler* (BPEL, 2007).

Os *handlers* podem ser utilizados em relação aos escopos podendo realizar um tratamento final para lidar com finalizações forçadas, ou qualquer tipo de tratamento. Existem os seguintes tipos de tratamentos (LAWISCH, 2008):

- ***Event handlers:*** São utilizados para invocar eventos de mensagem ou eventos de relógio (prazo final ou duração).
- ***Fault handlers:*** São utilizados para tratar diferentes situações fora do comum (erros internos).
- ***Compensation handlers*:** Desfazem efeitos persistidos em atividades já finalizadas.
- ***Termination handlers*:** Utilizados para forçar o término de execução dentro de um escopo (erros externos).

A especificação WS-BPEL é muito complexa e extensa. Neste capítulo apresentamos somente as características principais da linguagem. Para uma referência melhor e mais completa dessa tecnologia pode ser encontrado na sua própria especificação em (BPEL, 2007).

3.1.2 – Engines de Construção e Execução BPEL

A execução de processos de negócios e dados, tanto para as empresas e a comunidade científica, é significativa na medida em que a partir de um modelo de processos de negócio possa gerar um conjunto de tarefas bem definidas. Utilizando uma ferramenta para a construção e execução de um workflow, conhecido também como *engine,* é possível automatizar algumas dessas tarefas.

As ferramentas que se destacam comercialmente e academicamente para a execução de workflow em BPEL são ActiveBPEL e Oracle BPEL Process Manager, sendo a primeira gratuita e a segunda comercial. Uma lista mais completa de ferramentas para

construção de workflow BPEL pode ser encontrado na wikipédia (Wikipedia, 2009). Abaixo será detalhada cada uma dessas *engines*.

3.1.2.1 ActiveBPEL

A *engine* ActiveBPEL é uma implementação open source que suporta toda a especificação BPEL 1.1. Ele funciona em qualquer container de Servlet padrão, como por exemplo o Apache Tomcat. Ele está disponibilizado sob a licença GNU GPL (*General Public License*), ou seja, ele não pode ser apoderado por outra pessoa (ActiveBPEL, 2009). O conjunto de ferramentas ActiveBPEL é formado pelo: ActiveBPEL Designer; ActiveBPEL Engine e Active Endpoints SOAP Clients.

ActiveBPEL Designer: é um ambiente para construção, teste e compilação de aplicações baseadas em WS-BPEL. Os serviços web são inseridos no processo WS-BPEL por meio de suas definições WSDL que podem ser importadas na ferramenta. A interface gráfica possui uma paleta com todas as atividades WS-BPEL de modo que é necessário apenas clicar e arrastar para usá-las. Após definição do processo WS-BPEL, pode-se simular a execução do processo adicionando-se valores-teste para as respostas dos serviços web. Ao final do desenvolvimento, pode-se publicar o processo gerado exportando o arquivo descritor de processo para o servidor WS-BPEL (FREITAS, 2008).

ActiveBPEL Engine: é responsável por receber requisições de uso de processos WS-BPEL, instanciar o processo, executar os comando contidos nos processos e retornar o processo ao requisitante (FREITAS, 2008).

Active Endpoints SOAP Clients: permite a configuração e o gerenciamento do ActiveBPEL Engine e todos artefatos contidos nele. Permite também o monitoramento dos gráficos de execução de um processo e analisar quais foram os passos executados neste processo (FREITAS, 2008).

3.1.2.2 Oracle BPEL Process Manager

Oracle BPEL Process Manager oferece uma abrangente e fácil infra-estrutura para criar, implantar e gerenciar processos de negócios BPEL. Oracle BPEL Process Manager é uma poderosa ferramenta de integração para a empresa. Sua capacidade de se conectar a sistemas externos e processo, misturado com o seu apoio para uma variedade de tecnologias

de apresentação faz dela uma ferramenta ideal para definir e implementar lógica de processos de negócios (ORACLE, 2008).

Oracle BPEL Process Manager tem suporte nativo ao BPEL, é compatível com a especificação BPEL4WS 1.1 e permite troca de mensagens síncronas e assíncronas. Além dessas existem outras características que tornam a ferramenta bastante robusta. A primeira delas é a "desidratação do contexto" de execução do workflow que é a persistência dos estados da execução do workflow. Neste caso se uma falha ocorrer na execução de algum processo, o workflow pode ser reiniciado a partir do ponto onde houve a falha. Outras características são a capacidade de manipular arquivos XML grandes e possuir um servidor UDDI (*Universal Description, Discovery and Integration*) para facilitar a localização de serviços Web disponíveis (MATTOS, 2008). A seguir estão os principais componentes do Oracle BPEL Process Manager.

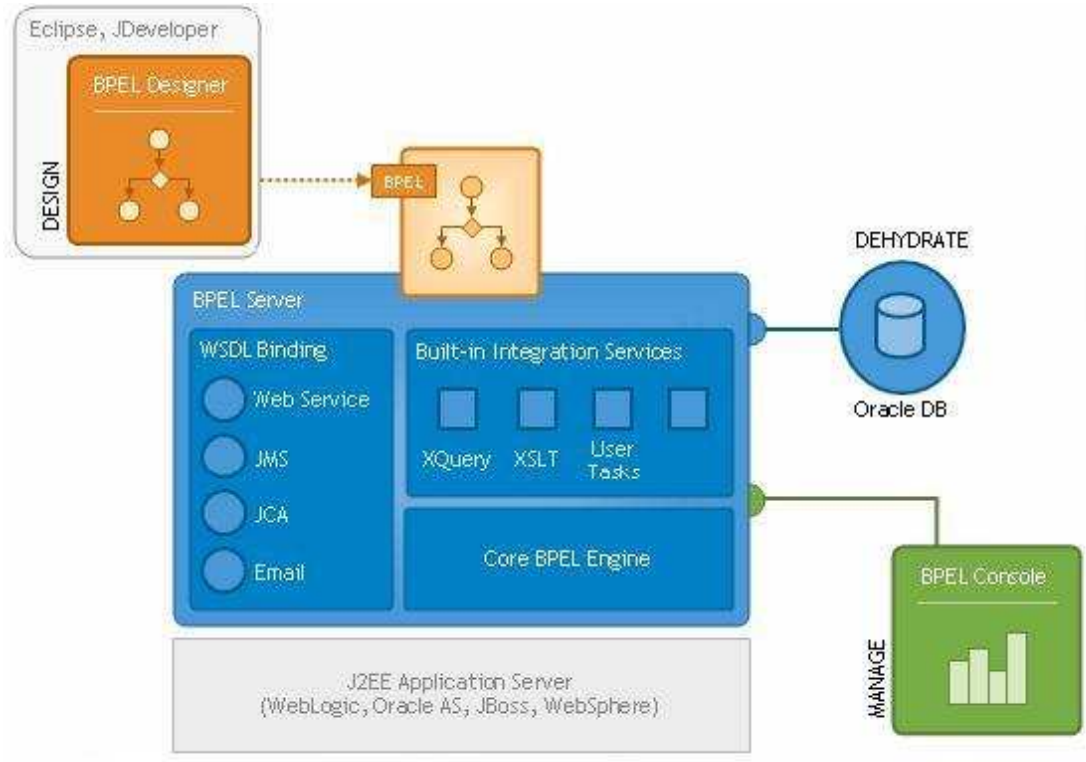

Figura 3.9 – Arquitetura do Oracle BPEL Process Manager (MATTOS, 2008)

O *BPEL Designer* fornece uma forma gráfica e amigável para construir processos BPEL; o Core *BPEL Engine* é o núcleo de execução do workflow; os serviços do "*Built-in*" facilitam a conectividade e as capacidades de transformação de dados dos processos BPEL; o componente "*WSDL binding"* permite a conectividade com outros

protocolos além do SOAP (*Simple Object Access Protocol*); e por fim a console BPEL que permite a gerência, a administração e a depuração dos processos e workflows disponibilizados no servidor (MATTOS, 2008).

3.2 – Scripts (Beanshell)

Historicamente, o gerenciamento de workflows científicos foi tratado com o uso de linguagens scripts para automação das tarefas. Mais tarde os Sistemas de Workflow Científicos desenvolveram notações gráficas como as usadas no Kepler, por exemplo. Entretanto, por traz dessas notações gráficas, existe um aparato para persistência de modelos que é textual, normalmente baseada em XML, como a linguagem usada na ferramenta Taverna.

Algumas linguagens scripts se tornaram tão dependentes dos SWCs que não são recomendadas para uso fora do sistema. É o caso do XScufl (*XML Simple Conceptual Unified Flow*), linguagem descrita em XML usada pelo projeto Taverna para armazenar e recuperar as definições de workflow.

Além de gerenciar o workflow, alguns sistemas admitem a utilização de scripts, que são ativados automaticamente no início e final de execução de atividades, o que permite um grau adicional de automatismo associado à execução de cada atividade.

Ao utilizá-los, os usuários podem acrescentar tarefas que não estão disponíveis de forma simples e rápida nos SWCs, sem a necessidade de utilizar compiladores externos, pois eles são interpretados pelo sistema (MATTOS, 2008).

Beanshell é um interpretador de comandos escrito em Java. É uma linguagem de script simples, pequena e integrada à plataforma Java, assim é possível chamá-lo a partir de aplicações Java para executar código Java dinamicamente e em tempo de execução (BEANSHELL, 2009).

O Beanshell pode ser usado para realização de experimentos e testes, depuração, estender aplicações com novas funcionalidades, regras de negócios, sistemas embarcados, e até mesmo para iniciar programadores no Java, pois usando este script é possível obter um resultado rápido e eficiente (BEANSHELL, 2009).

No exemplo abaixo temos a representação de um código na linguagem Beanshell inserido no sistema Taverna. É realizada a concatenação das três variáveis seq1, seq2 e seq3 que são recebidas como dados de entrada. O resultado é atribuído na variável fasta.

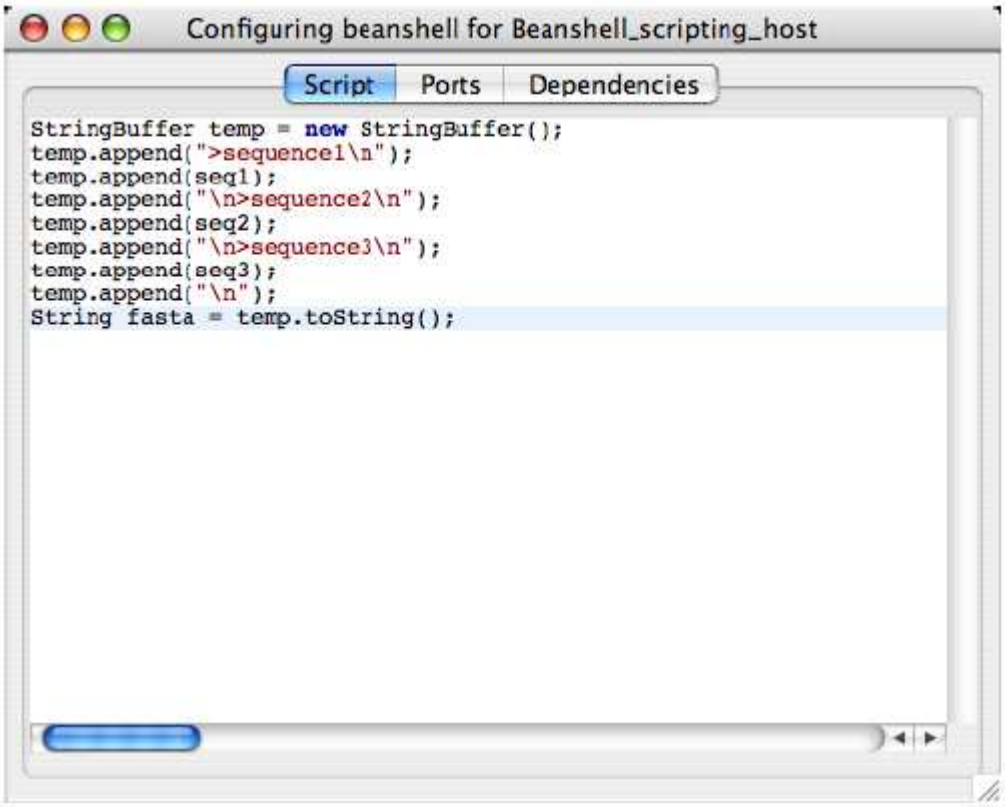

Figura 3.10 – Script Beanshell inserido no sistema Taverna (Taverna Project, 2007).

Capítulo 4 – Introdução às Ferramentas do Portal Fisiocomp

Nessa seção são apresentadas duas ferramentas para a integração com o Portal: AGOS (*API Generator for ODE Solution*) e JynaCoreAPI. Atualmente apenas o JynaCore API possui suporte com o Portal. O Portal desenvolvido é um ambiente similar ao Portal Fisiocomp, e será detalhado no próximo capítulo. Neste momento é importante apenas se preocupar com as características básicas das ferramentas para que no próximo capítulo fique clara a relação entre as ferramentas e o Portal.

4.1 – AGOS (*API Generator for ODE Solution*)

O AGOS é o ambiente científico do Fisiocomp propriamente dito. A ferramenta é um gerador de API (*Application Program Interface*) para solução de EDOs (Equações Diferenciais Ordinárias). Ele pode ser considerado uma ferramenta de transformação de modelos no padrão XML em código C++, ou seja, ele transforma um modelo matemático em um código executável correspondente. Recentemente, a comunidade de biologia computacional desenvolveu um padrão baseado em XML para a descrição de modelos celulares (CELLML, 2004). Um arquivo CellML inclui Content MathML para fornecer uma representação legível, tanto para seres humanos quanto para computadores, do relacionamento matemático entre componentes biológicos. Atualmente, a ferramenta AGOS funciona para qualquer PVI (Problema de Valor Inicial) baseado em sistemas não-lineares de EDOs de primeira ordem documentados no padrão MathML. Consequentemente, AGOS é uma ferramenta de transformação poderosa e útil que tem como objetivo auxiliar o desenvolvimento de vários problemas científicos nas mais diversas áreas de pesquisa (AMORIM, 2007).

O processo de transformação consiste primeiramente na submissão de um arquivo CellML completo ou apenas seu subconjunto MathML associado a um projeto novo ou já existente na árvore de projetos disponível em (FISIOCOMP, 2009). Uma vez submetido, o arquivo XML é armazenado e pode ser transformado em uma API caso seja escolhido a opção de executar. Na figura 4.10 e 4.11 podemos visualizar a interface da submissão do arquivo XML e a árvore de projetos com o mesmo arquivo armazenado respectivamente.

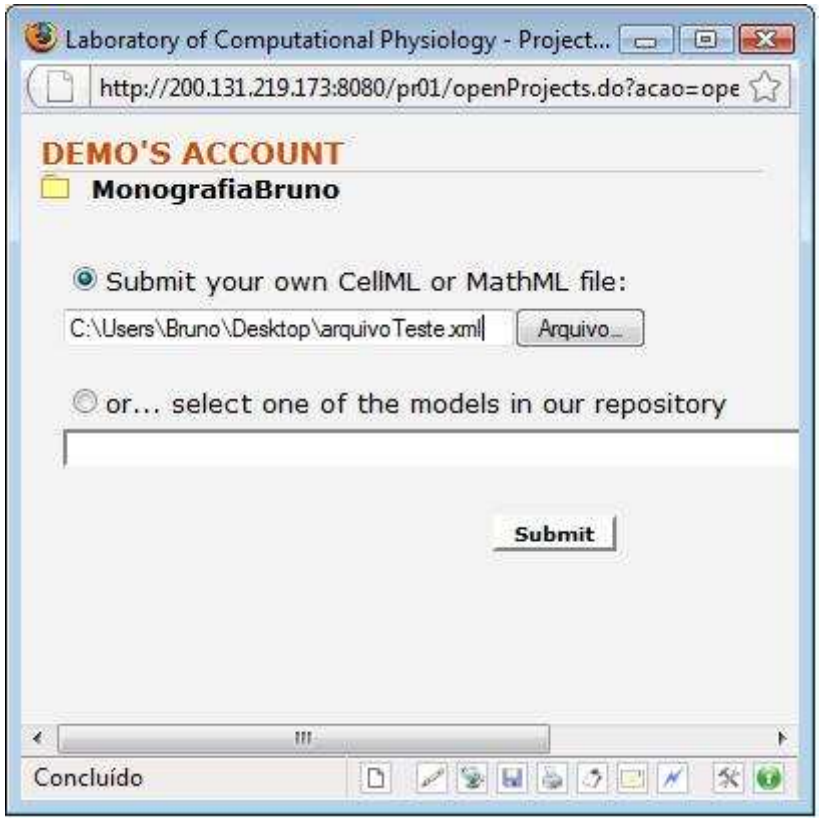

Figura 4.11 – Tela de submissão do arquivo XML.

Figura 4.12 – Árvore de projetos com o arquivo XML armazenado.

A ferramenta de tradução é composta de três componentes básicos: um Pré-processador para o formato XML, Analisador Sintático e Extrator, e o Gerador de Código. O Pré-processador lê o arquivo XML (MathML ou CellML) e extrai o conteúdo em uma árvore de estrutura de dados. A árvore XML é percorrida pelo Analisador Sintático e Extrator EDO

que testa a validade do arquivo XML de entrada, identifica os elementos da EDO e os armazena em estruturas de dados apropriadas. Finalmente, o Gerador de Código combina a informação extraída com um *template* de código e gera a API AGOS (AMORIM, 2007).

Como visto anteriormente, no segundo parágrafo desta seção, temos duas atividades principais dentro de toda a estrutura para a transformação do arquivo XML: a submissão do arquivo e a sua execução. A partir deste ponto de vista, foi elaborada a aplicação presente neste trabalho com o intuito de automatizar essas funções de modo a flexibilizar e agilizar o processo de transformação dos modelos científicos. Para um maior conhecimento sobre o AGOS, consultar (AMORIM, 2007).

4.2 – JynaCore API

JynaCore API é uma biblioteca desenvolvida para servir de base na construção de aplicações que utilizam técnicas de modelagem e simulação para estudos das causas e efeitos das dinâmicas encontradas em processos de desenvolvimento de software. JynaCore API é uma biblioteca de código livre, desenvolvida inteiramente em Java. Esta biblioteca captura o processo básico de simulação de modelos dinâmicos: descrição de modelos, simulação através de um método numérico e registro dos resultados (KNOP, 2009).

Atualmente, duas linguagens para construção de modelos dinâmicos são descritas e implementadas pela JynaCore API: os diagramas de estoque e fluxo, e os modelos da linguagem estendida de Dinâmica de Sistemas. A descrição detalhada dessas linguagens pode ser encontrada em (KNOP, 2009). Os usuários da biblioteca, cujo único interesse seja o resultado da simulação, podem simular os modelos em um nível mais elevado de abstração, não vinculando sua aplicação a uma linguagem específica (KNOP, 2009).

A arquitetura básica de uma simulação dentro JynaCore API envolve o processo de gerar um conjunto de dados a partir de um modelo e seus parâmetros. Na implementação se encontram as interfaces que descrevem o processo básico de simulação em alto nível de abstração. Os outros módulos estendem essas interfaces para implementar linguagens específicas como a dos diagramas de estoque e fluxo e modelos em linguagem estendida de Dinâmica de Sistemas. A funcionalidade central do processo de simulação é gerido pela interface *JynaSimulation*. Ela costura as demais interfaces envolvidas, atuando como mediadora (KNOP, 2009):

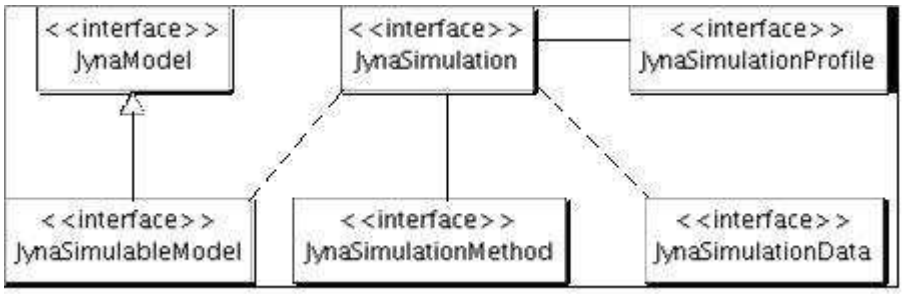

Figura 4.13 – Diagrama de classes dos elementos de um processo básico de simulação em JynaCore API (KNOP, 2009).

O *JynaModel* é a interface que descreve um modelo qualquer. Se este modelo puder ser simulado, ele também será um *JynaSimulableModel*. Atualmente, apenas modelos de estoque e fluxo de Dinâmica de Sistemas ou modelos de instância em metamodelos estendem *JynaSimulableModel*.

O *JynaSimulation* orquestra todos os passos de uma simulação a fim de que um *JynaSimulableModel* possa gerar um conjunto de dados. Este conjunto de dados é armazenado em *JynaSimulationData*, observa os elementos quantificáveis dos modelos e registra seus valores durante a simulação.

O *JynaSimulatonProfile* é um conjunto de parâmetros de configuração de uma simulação. Estes parâmetros são encontrados, comumente, como intervalos de tempo em que a simulação ocorre, o tamanho e quantidade de passos, etc. O *JynaSimulationMethod* encapsula o método numérico que resolve o sistema de equações diferenciais com condições iniciais, contido implicitamente nos modelos de estoque e fluxo e linguagem estendida de Dinâmica de Sistemas.

Nosso interesse nos detalhes dessa ferramenta se limita a identificar as funcionalidades que deverão ser gerenciadas pelo nosso interpretador. Os comandos a serem executado por nosso interpretador se constituirão, em muitos casos, em chamadas a métodos implementados nas classes descritas acima. Para um maior conhecimento sobre o JynaCore API, consultar (KNOP, 2009).

Capítulo 5 – Apresentação da Aplicação Desenvolvida

A aplicação foi desenvolvida com o objetivo de gerenciar as funcionalidades permitidas dentro do ambiente do Portal Fisiocomp. Assim, adicionaremos a esse ambiente de modelagem computacional essa importante característica de um workflow científico, na qual, através de uma linguagem script podemos definir uma sequência de processos a serem executados. A linguagem para definição de processos e todos os passos para a sua interpretação, constituem a implementação realizada para este trabalho. A ferramenta consiste em um interpretador que foi nomeado de SWI (*Scientific Workflow Interpreter*) e neste trabalho será usada esta sigla para representá-lo.

O Portal do Fisiocomp possui uma estrutura que permite abrigar diferentes ferramentas no contexto da modelagem computacional. Uma dessas ferramentas é o AGOS. As funcionalidades do AGOS, portanto, são exemplos de blocos de construção de um workflow cuja execução pretende-se orquestrar.

Na prática, entretanto, integrar nosso interpretador com o ambiente do Portal, manipulando as funcionalidades do AGOS, demonstrou ser uma tarefa difícil pelo porte do código de implementação envolvido. Em face dessa dificuldade, optou-se por realizar nosso teste de conceito em uma versão mais simplificada do Portal, hora mantida em função de um projeto de iniciação científica que visa acrescentar uma nova ferramenta ao Portal.

A nova ferramenta a ser acrescida no Portal é o Jynacore API (KNOP, 2009). Nossos testes então foram voltados para a automação das funcionalidades dessa nova ferramenta, executando nessa versão simplificada do Portal Fisiocomp.

A arquitetura do Portal, onde o SWI deve ser integrado é a Cliente-Servidor, baseada em tecnologia JAVA, para isso utilizamos como servidor Web o Tomcat. Para funcionamento do SWI é utilizado um Servlet no ambiente do servidor. O Servlet executa os comandos armazenados em um objeto gerado pelo interpretador. Estes comandos são executados de maneira síncrona com a operação realizada pelo browser no lado cliente. O Servlet é um aplicativo que permanece em execução no servidor aguardando por solicitações dos clientes e tem a capacidade de atender diversas solicitações simultâneas (OSSES, 2009). Na figura 4.10 temos a visualização dos módulos da aplicação.

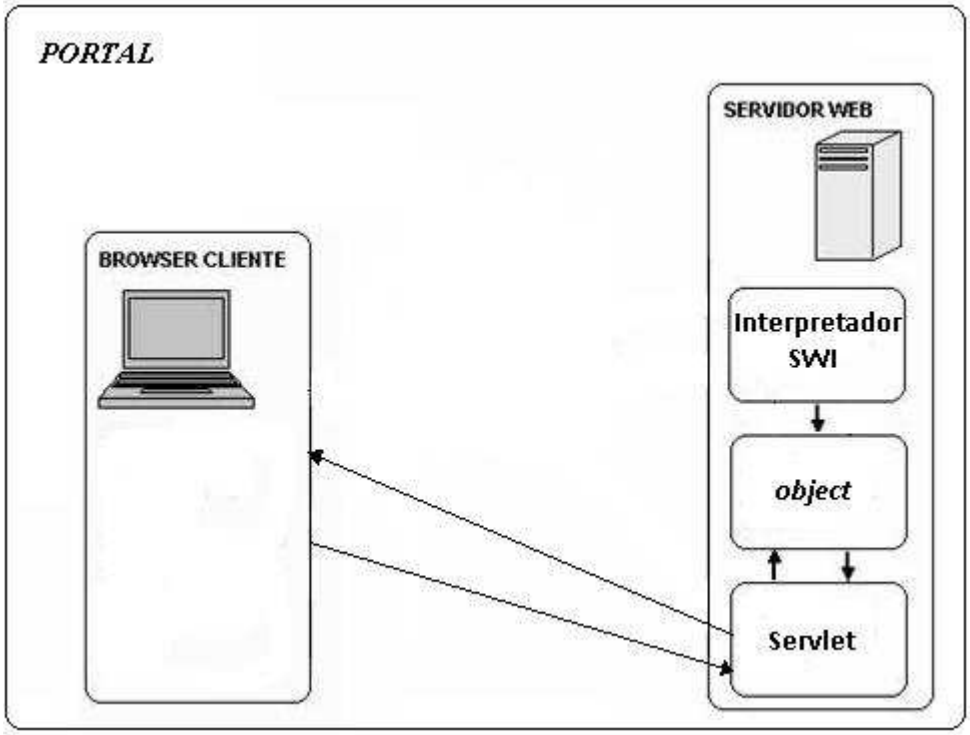

Figura 5.14 – Arquitetura da aplicação desenvolvida.

Dadas as complicações arquiteturais inerentes do ambiente operacional do Portal Fisiocomp, onde as funcionalidades possuem características heterogêneas, não sendo baseadas em uma única arquitetura de desenvolvimento como, por exemplo, Web-Services, optou-se por postergar a adoção de notações gráficas de modelagem, cuja implementação na arquitetura em questão demandaria muito tempo.

O objetivo desse trabalho é permitir a automação das atividades do Portal para ser utilizado de acordo como o que um Sistema de Workflow Científico propõe. Para isso, formas alternativas à de criação de uma linguagem para definição de processos foram pesquisadas, como por exemplo, o BPEL. No entanto, o uso do BPEL não atenderia o projeto, pelo fato do usuário ser obrigado a conhecer previamente a linguagem BPEL para construir sua sequência de serviços. Além disso, essa construção é feita dentro de uma IDE (*Integrated Developement Environment*), como por exemplo, o *NetBeans IDE*, mas o ambiente operacional em questão demanda que a modelagem seja realizada em um ambiente Web.

Existem algumas razões para a criação de uma linguagem para definição de processos específica para o projeto. Dentre elas podemos destacar controle sobre o processo de desenvolvimento, ou seja, é certo que o sistema irá funcionar sem nenhum tipo de surpresa negativa devida ao uso de frameworks. Além disso, o SWI, assim como o Servlet, são desenvolvidos em Java, logo, a aplicação é portável, podendo ser executada em diferentes

sistemas operacionais sem a necessidade de recodificação. Outra vantagem é que os Servlets rodam ao lado do servidor, herdando as características de segurança providas pelo Web Server e o tratamento das exceções.

5.1 – SWI (*Scientific Workflow Interpreter*)

Resumidamente, os interpretadores são programas de computador responsáveis por tomar ações efetivas conforme a orientação do usuário através de comunicação textual. Eles permitem aos usuários emitirem vários comandos, o que requer do usuário conhecer tais comandos e seus parâmetros, além da sintaxe da linguagem que será interpretada.

As partes iniciais de um interpretador são praticamente idênticas a de um compilador. Ambos fazem análise léxica, sintática e semântica. O compilador traduz código escrito em uma linguagem fonte para outra linguagem alvo. O papel do interpretador, por sua vez, não é gerar o programa alvo, mas sim executar as instruções do programa fonte. Outra diferença em relação ao modo como compiladores e interpretadores executam o programa fonte é que os interpretadores traduzem e executam linha por linha, enquanto o compilador precisa traduzir o código inteiro para poder executar.

A estrutura de classes do trabalho pode ser representada na interação entre as classes de dois projetos criados. O primeiro projeto contém as classes do SWI divididas em pacotes. O pacote *default* contém a classe *main.java* onde o programa inicia. No pacote *swi.principal* estão as classes responsáveis pelas análises léxicas, sintáticas e o gerenciamento dos erros. O pacote *swi.generation* é responsável por gerar e armazenar o objeto resultante do interpretador utilizado pelo Servlet. E finalmente o pacote *swi.tokens* define os *tokens* da linguagem criada. Esse primeiro projeto é compilado e disponibilizado através de uma biblioteca, *SWI – dist/SWI.jar*, que é inserida e utilizada no segundo projeto referente ao Servlet. Neste projeto o pacote *servlet.controller.project* implementa as classes *doGet* e *doPost* para responder as solicitações de um cliente e no pacote padrão estão os arquivos *.jsp*, *.css*, *.js*, para a construção do Portal (Figura 5.15).

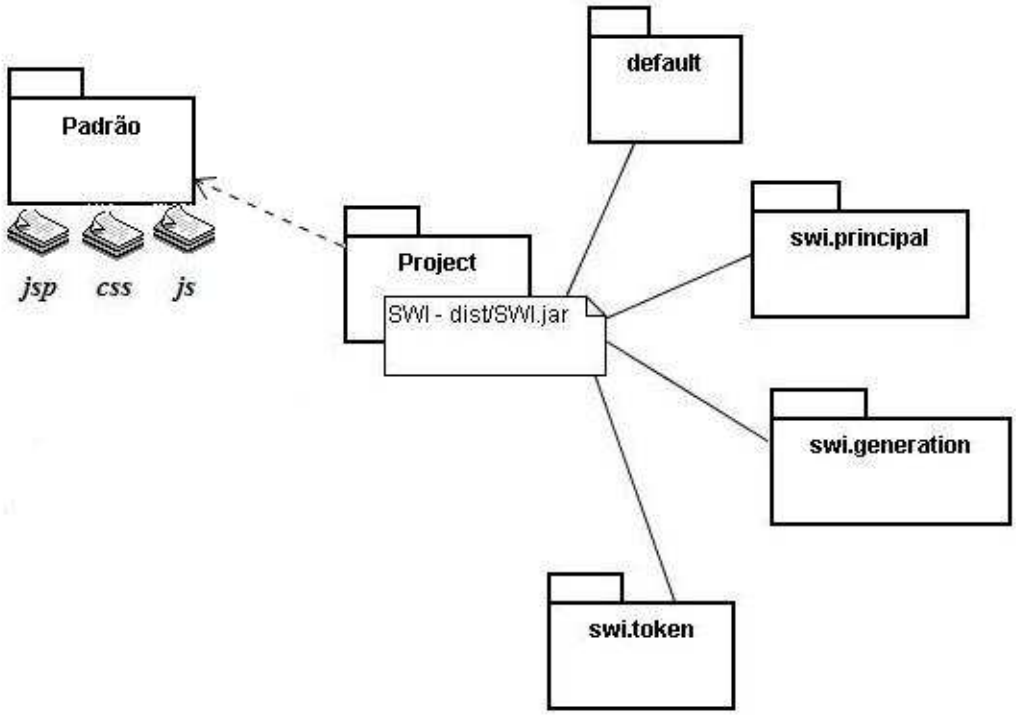

Figura 5.15 – Estrutura dos pacotes do projeto.

No SWI, os comandos e parâmetros lidos sem nenhum tipo de erro são guardados em uma lista que será usada pelo Servlet para a execução das funções, e os erros encontrados são concatenados em uma *String* e mostrados para o usuário na tela após a execução dos passos. Portanto, o SWI funciona inicialmente como um compilador, pois feito as análises, é gerado uma lista com os comandos que serão utilizados pelo Servlet. Essa lista pode ser considerada como um código intermediário.

O SWI é capaz de definir uma sequência de serviços a serem executados através de uma linguagem script. Atualmente os serviços disponíveis na linguagem são: upload de arquivos; execução de modelos com extensão *".jyna"* e a visualização dos resultados apresentados em um gráfico. Novas funcionalidades podem ser inseridas no SWI. O suporte a elas depende da alteração do código-fonte de algumas classes no interpretador e no Servlet. Além de novos serviços, o suporte para estruturas de controle, variáveis locais para guardar o retorno de execução, entre outras coisas, também podem ser acrescentados no interpretador para diversificar e agilizar a construção do script e a orquestração dos serviços.

Na próxima seção é mostrado como está definida a gramática da linguagem script e como é feito as análises léxica, sintática e semântica do mesmo.

5.2 – Análise da Linguagem Fonte

A análise dos dados de entrada é dividida em três partes. A análise léxica, na qual varre o arquivo da esquerda para direita e lê os caracteres individuais criando então uma sequência de *tokens*. Análise sintática, na qual apanha o fluxo de *tokens* e garante se a sintaxe está correta. E a análise semântica, na qual certas verificações são realizadas a fim de assegurar que os componentes de um programa se combinam de forma significativa.

Nas próximas seções será detalhado cada analisador da aplicação e os resultados obtidos da linguagem fonte desenvolvida para o SWI.

5.2.1 Análise Léxica

O Analisador Léxico é a primeira fase do interpretador. Sua tarefa principal é ler os caracteres de entrada e produzir uma sequência de *tokens* que é utilizado mais a frente pelo analisador sintático.

Quando se fala sobre a análise léxica, usamos os termos "*token*" e "*lexema*" com significados específicos. Em geral, existe um conjunto de cadeias de entrada para os quais o mesmo *token* é produzido como saída. Um *lexema* é um conjunto de caracteres no programa-fonte que é reconhecido pelo padrão de algum *token*. Tendo como exemplo a linguagem utilizada para a elaboração da aplicação desenvolvida, podemos ver que os lexemas

upload (modelo.jyna) ; execute (modelo.jyna) ; view (resultado.png) ;

poderiam ser agrupados nos seguintes *tokens*:

1. Identificador (ID) *upload*
2. Abre parênteses "("
3. Identificador (ID) *modelo.jyna*
4. Fecha parênteses ")"
5. Ponto e vírgula ";"
6. Identificador (ID) *execute*
7. Abre parênteses "("
8. Identificador (ID) *modelo.jyna*
9. Fecha parênteses ")"
10. Ponto e vírgula ";"
11. Identificador (ID) *view*

12. Abre parênteses “(“
13. Identificador (ID) *resultado.jpg*
14. Fecha parênteses “)”
15. Ponto e vírgula “;”

Como o Analisador Léxico é a parte do interpretador que lê o texto-fonte, também pode realizar algumas tarefas secundárias ao nível da interface com o usuário. Uma delas é a de remover do programa fonte os comentários, espaços em branco, tabulações e caracteres de avanço de linha.

Seguindo as definições para os *tokens* dadas acima, a passagem do analisador léxico pelos lexemas acima resultaria na seguinte sequência de tokens:

ID (ID) ; ID (ID) ; ID (ID) ;

Mas como *upload*, *execute* e *view* são palavras reservadas na gramática e são inseridas em uma tabela de símbolos, temos então:

upload (ID) ; execute (ID) ; view (ID) ;

Esta sequência de tokens é então utilizada como entrada para o analisador sintático como veremos na próxima seção.

5.2.2 Análise Sintática

O Analisador Sintático obtém uma cadeia de *tokens* proveniente do Analisador Léxico e verifica se a mesma pode ser gerada pela gramática da linguagem-fonte. É esperado de um Analisador Sintático que ele relate quaisquer erros de sintaxe de uma forma inteligível. Ele deve também se recuperar de erros que ocorram mais comumente, a fim de poder continuar processando o resto de sua entrada.

Cada linguagem de programação possui as regras que descrevem a estrutura sintática dos programas bem-formados. Em Pascal, por exemplo, um programa é constituído por blocos, um bloco por comandos, um comando por expressões, uma expressão por *tokens* e assim por diante. A sintaxe das construções de uma linguagem pode ser descrita por

gramáticas livres de contexto. Usualmente, as frases gramaticais do programa fonte são representadas por uma árvore gramatical (AHO, 1986).

Uma gramática descreve naturalmente a estrutura hierárquica de muitas construções das linguagens de programação. Por exemplo, um comando *if-else*, em C, possui a forma :

if (expressão) comando **else** comando

Usando-se a variável *expr* a fim de denotar uma expressão e a variável *cmd* para um comando, esta regra de estruturação pode ser expressa como

cmd -> **if (** expr **)** cmd **else** cmd

onde a seta deve ser lida como “pode ter a forma”. Tal regra é chamada de produção. Em uma produção, os elementos léxicos, como a palavra reservada *if* e os parênteses, são chamados de *tokens*. As variáveis como *expr* e *cmd* representam sequências de *tokens* e são chamados de não-terminais (AHO, 1986).

O papel do Analisador Sintático é, dada uma cadeia de *tokens*, identificar se esta pertence a uma à linguagem descrita pela gramática livre de contexto. Isto é feito tentando encontrar uma derivação, a partir da gramática livre de contexto, que consuma toda a cadeia de entrada. Caso não exista uma derivação, conclui-se que esta cadeia não pertence à linguagem descrita pela gramática, e um erro é reportado pelo Analisador Sintático. Abaixo temos a gramática descrita utilizada no desenvolvimento da aplicação.

Program -> Stmt_List
Stmt_List -> Statement Stmt_List **|** ***Epsilon***
Statement -> ***execute (ID) ;*** **|** ***upload (ID) ;*** **|** ***view (ID) ;***

O exemplo abaixo mostra uma cadeia de *tokens* aceita pela linguagem e sua árvore de derivação:

execute (ID) ; upload (ID) ; view (ID) ;

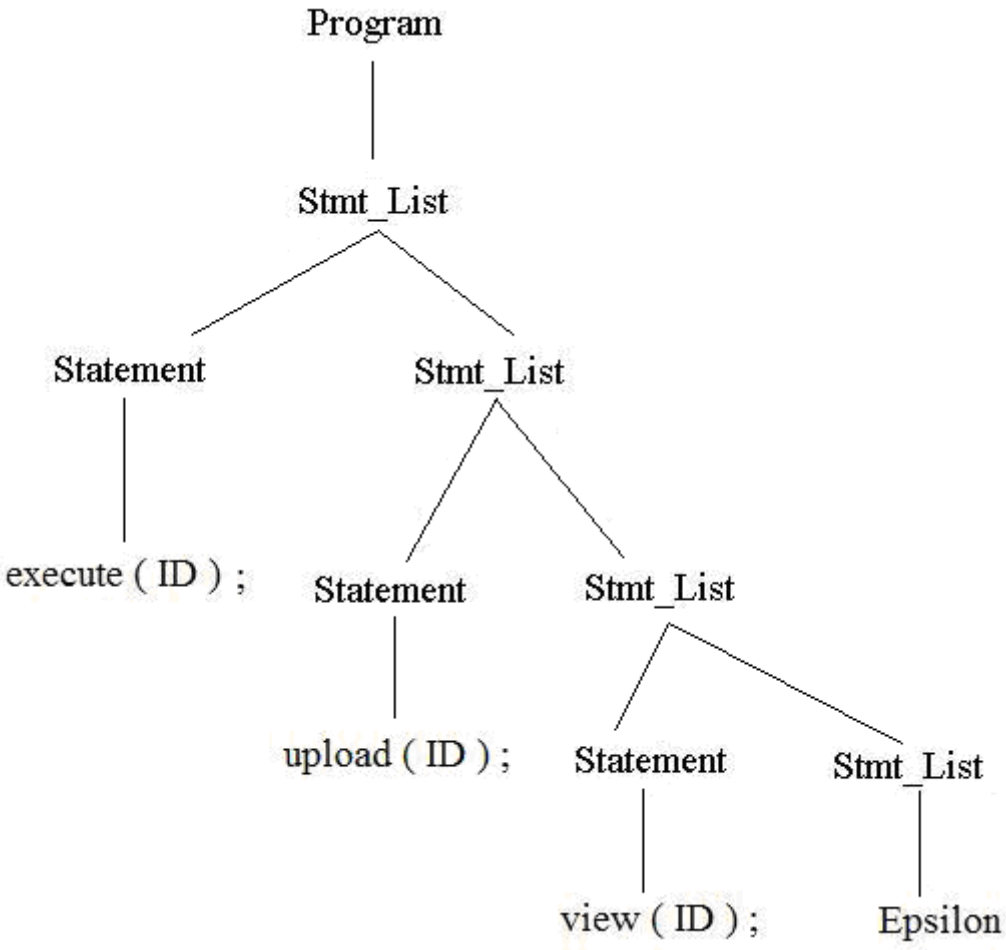

Figura 5.16 – Árvore de derivação da cadeia de *tokens* descrita.

5.2.3 Análise Semântica

O interpretador precisa verificar se o programa segue as convenções sintáticas e semânticas da linguagem fonte. Esta checagem assegura que certos tipos de erros de programas serão detectados e reportados. Alguns exemplos de verificação incluem: verificação de tipos; verificação de fluxo de controle; verificações de unicidade; verificações relacionadas aos nomes.

Na aplicação focalizamos a verificação relacionada ao tipo de arquivo. A execução dos simuladores suportados no Portal depende da extensão do arquivo parametrizado na função *execute.* Se uma extensão desconhecida for lida, é esperada uma mensagem de erro. Mas outros tipos de verificações podem ser implementados no SWI, como por exemplo, a verificação da consistência de arquivos XML, investigar se o arquivo passado como parâmetro existe, entre outros.

5.3 – Exemplos para Definição de Processos no SWI

Esta exemplificação tem por finalidade mostrar o funcionamento do SWI juntamente com o Portal passo a passo. Consiste em dois tipos de exemplos, sendo o primeiro sem erros, ou seja, o script foi escrito de forma correta. E o segundo contendo um erro sintático. O ambiente usado foi o Portal simplificado, similar ao Portal Fisiocomp e foram utilizados modelos de Sistemas Dinâmicos simulados no JynaCore API. O funcionamento interno do simulador não é relevante para a aplicação, pois o Servlet irá apenas realizar chamadas a métodos implementados nas classes do simulador.

O primeiro passo é criar um novo projeto para a exemplificação. Quando um novo projeto é criado, uma nova pasta é criada no servidor com o mesmo nome do projeto para guardar os arquivos que posteriormente estarão neste projeto. Nessa versão simplificada do Portal apenas projetos para a ferramenta JynaCore API são suportados. É criado o projeto com o nome “projeto_simulacao” conforme mostrado abaixo (Figura 5.17).

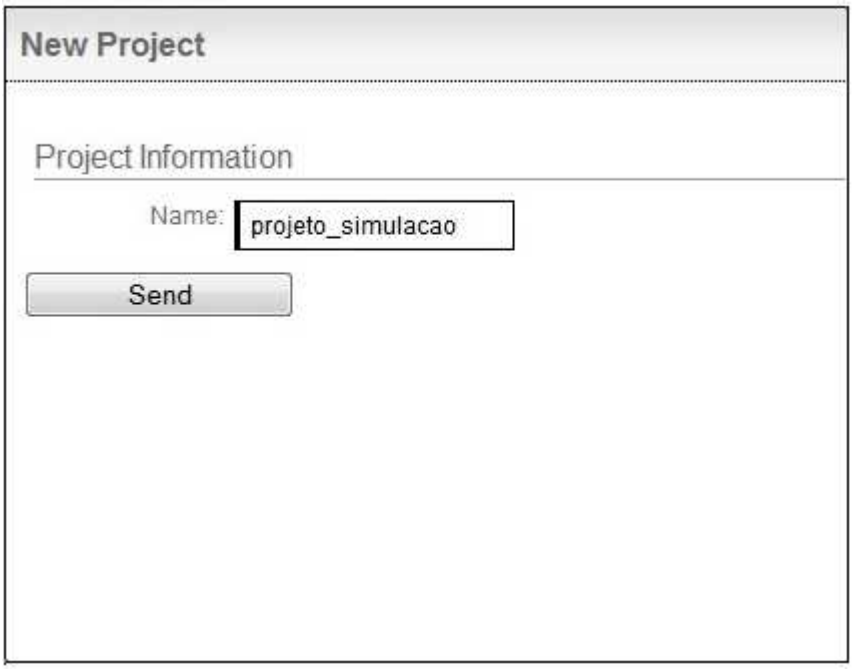

Figura 5.17 – Tela de criação de um projeto novo no Portal.

O projeto fica então armazenado em uma árvore, listando todos os projetos. Neste exemplo temos apenas esse novo projeto criado como mostra a figura 5.17.

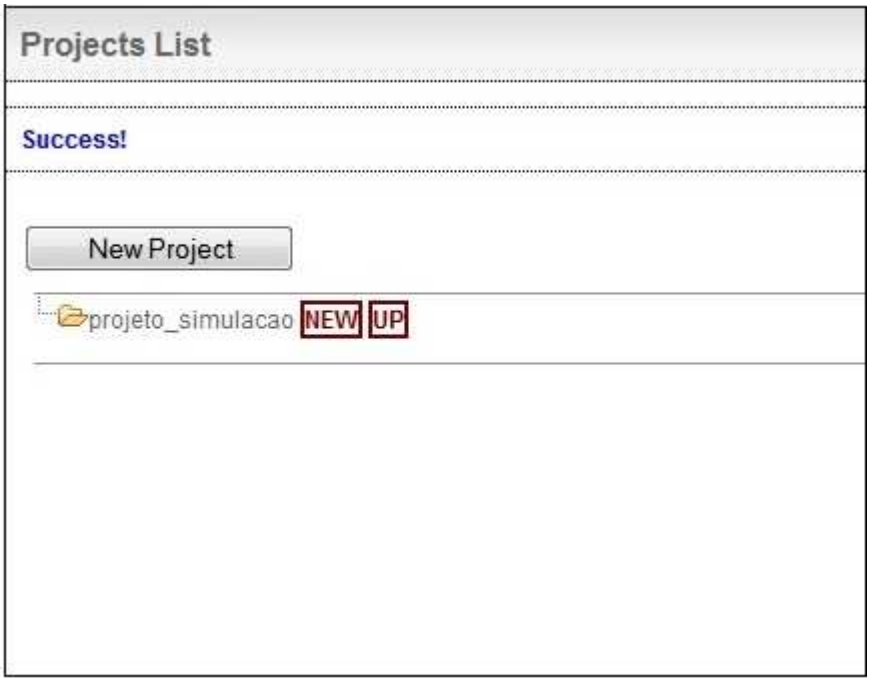

Figura 5.18 – Projeto apresentado na árvore.

Com o projeto criado, o próximo passo é construir o script contendo a linguagem definida para a orquestração dos processos. Esse script pode ser criado no próprio Portal na opção "*new*" ou pode ser realizado o *upload* de algum arquivo no disco local na opção "*up*". Ambas as opções ficam em frente ao projeto criado, como mostra a figura 5.18. Neste primeiro exemplo criamos um script escrito de forma correta com a seguinte sintaxe: "*upload(fullOscilatory.jyna); execute(fullOscilatory.jyna); view(fullOscilatory.png);"*. O arquivo ficará armazenado na pasta do projeto no servidor e será mostrado na árvore dentro do diretório do projeto.

Figura 5.19 – Tela de criação do script escrito de forma correta.

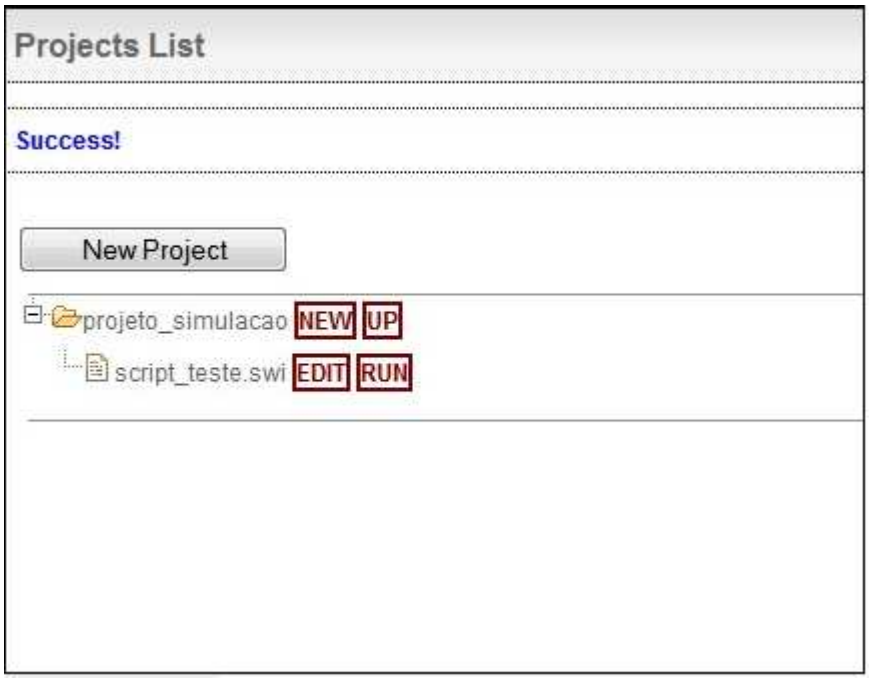

Figura 5.20 – Script apresentado na árvore dentro do diretório do projeto.

Por fim é necessário executar o script, para isso basta clicar no botão *"run"*, em frente ao arquivo, na árvore. Feito isto, a interface do Portal aciona o Servlet, que aciona o SWI, que analisa o script lexicamente e sintaticamente, e gera uma lista com os comandos para serem executados. Neste caso nenhum erro é concatenado na *String* de mensagens, e todas as três funções são aceitas e enviadas para o Servlet interpretar e executar. O Servlet então executa uma a uma as funções começando pela primeira, o *upload*.

Figura 5.21 – Tela de *upload* do modelo a ser simulado.

Enviado o modelo *fullOscilatory.jyna* o Servlet interpreta o comando *execute* e o simulador JynaCore API é chamado para rodar o modelo. Neste ponto é feita uma análise semântica com relação ao tipo do arquivo a ser executado. Como o arquivo é do tipo "*.jyna*', o simulador JynaCore API é chamado. Quando o modelo é executado, um arquivo de dados é gerado e o mesmo é representado em forma de gráfico, através do Gnuplot. Após o *execute,* o último comando *view* é lido e executado.

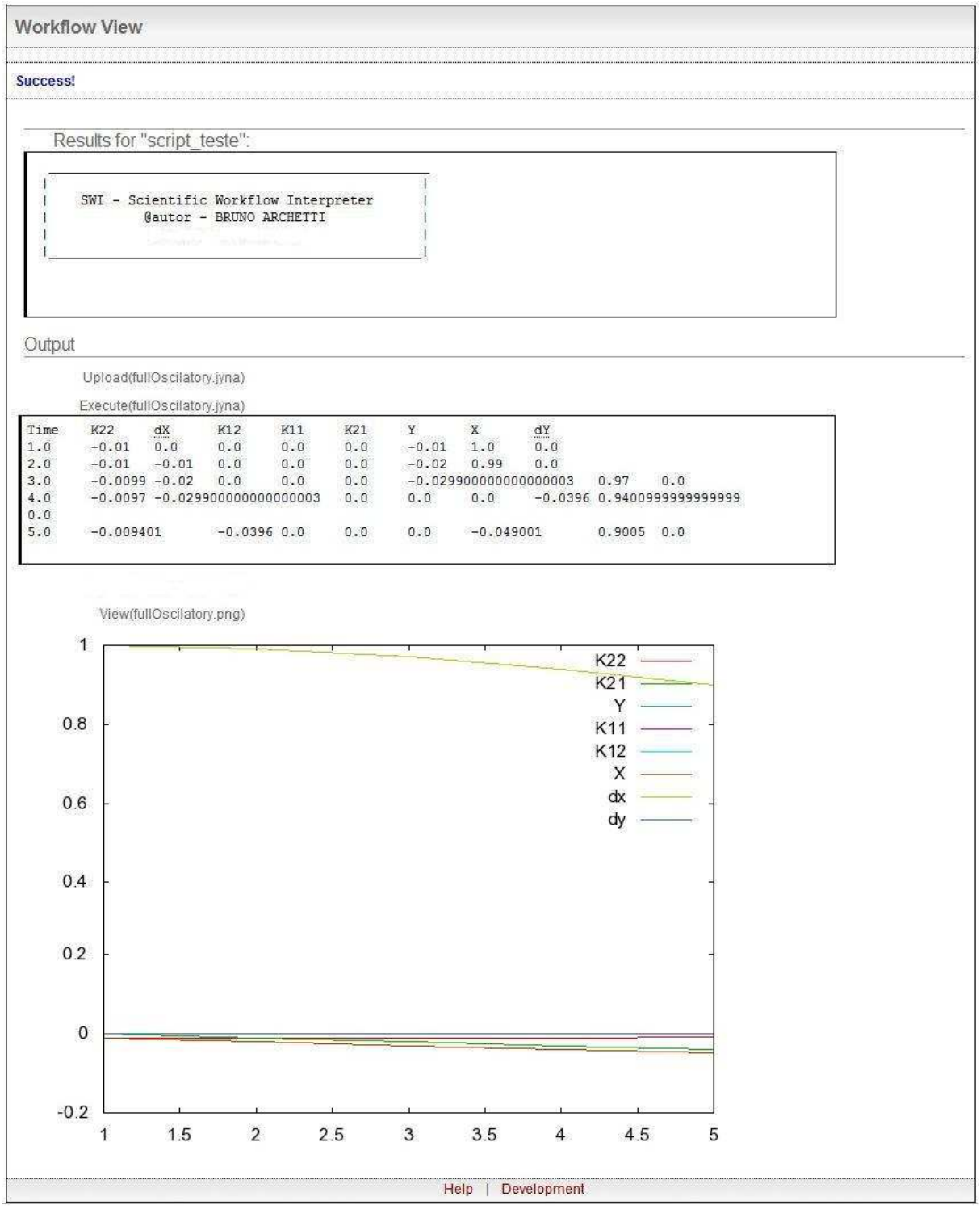

Figura 5.22 – Tela de saída com o resultado do script executado.

Fazendo uma análise da figura 5.22, temos na saída do programa os dados do modelo executado e abaixo o gráfico gerado a partir desses dados.

A figura 5.23 mostra como ficou o diretório do projeto após a execução de todas as funções. Podemos perceber que todos os arquivos utilizados são salvos na pasta do respectivo projeto criado.

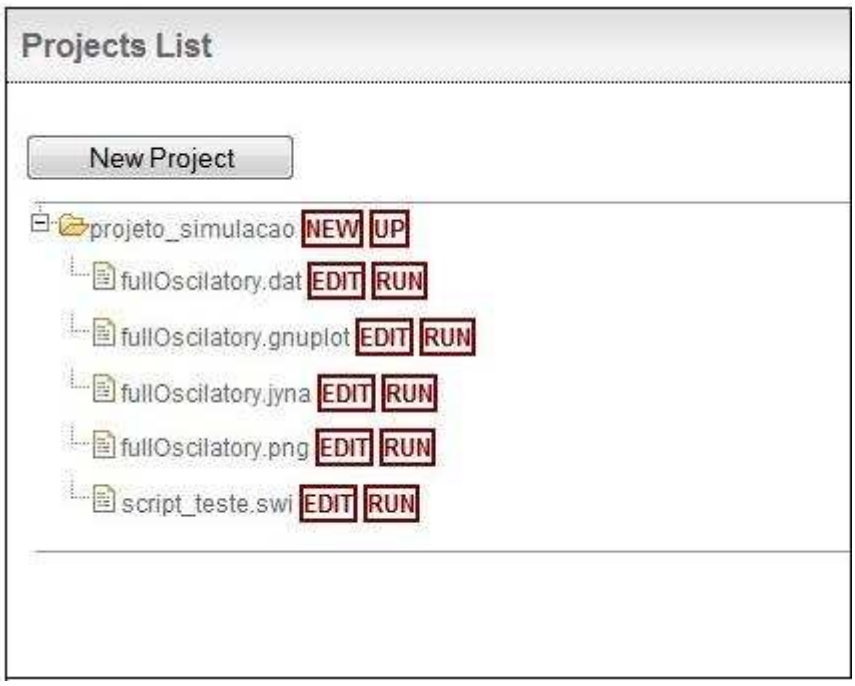

Figura 5.23 – Arquivos utilizados no projeto visualizados na árvore.

Além dos arquivos submetidos: *script_test.swi*; *fullOscilatory.jyna*, outros três novos arquivos são criados com a execução do script. São eles: *fullOscilatory.dat*, arquivo gerado pelo simulador JynaCore API e armazena os dados do modelo; o *fullOscilatory.gnuplot*, script para gerar o gráfico; e o *fullOscilatory.png*, arquivo gerado pelo script Gnuplot para representar os dados em forma de gráfico.

Vamos agora demonstrar um script contendo um erro sintático. Criamos então um novo projeto chamado de *projeto_simulacao_erro*. Dentro deste projeto, inserimos o script sintaticamente incorreto com a seguinte sintaxe: *"upload(fullOscilatory.jyna); execute( ); view(fullOscilatory.png);"* (Figura 5.24).

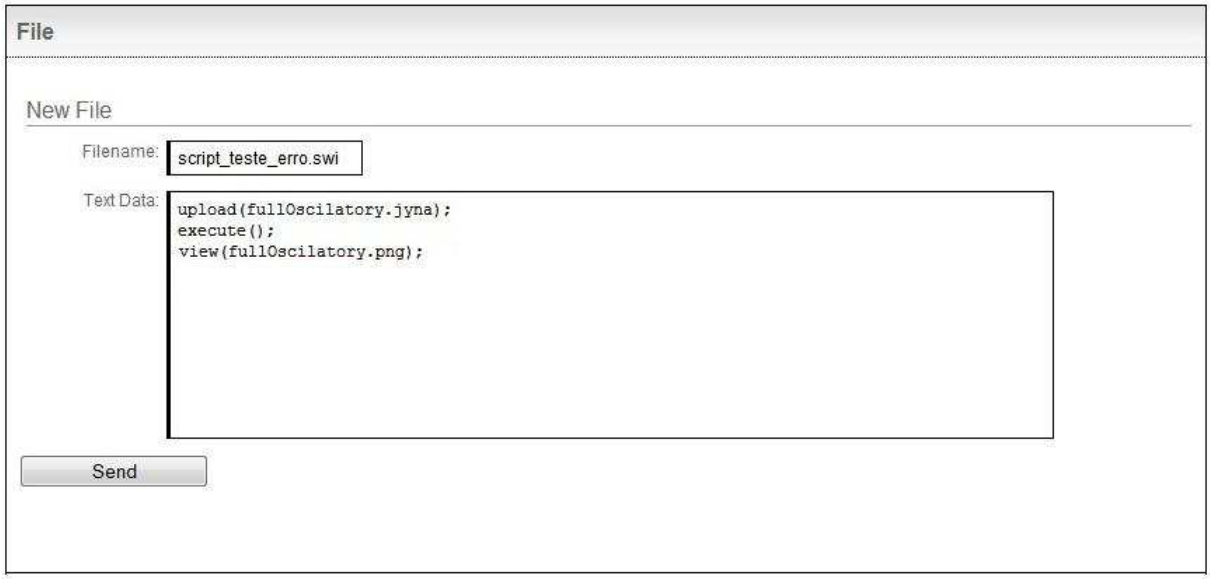

Figura 5.24 – Tela de criação do script escrito de forma incorreta.

Podemos ver que o erro está no segundo comando. A linguagem definida não permite que a função *execute* seja escrita sem nenhum parâmetro. Portanto um erro sintático é esperado. Rodando esse script temos a tela de *upload* sendo executada (Figura 5.25).

Figura 5.25 – Tela de upload do modelo a ser simulado do exemplo sintaticamente incorreto.

O próximo passo seria executar o comando *execute*, mas o mesmo não foi enviado ao Servlet, pois possui um erro em sua construção. Portanto, a próxima função interpretada e

executada é a *view*. O erro sintático é concatenado em uma *String* e mostrado no Portal com a identificação da linha e coluna de onde o erro ocorreu (Figura 5.26).

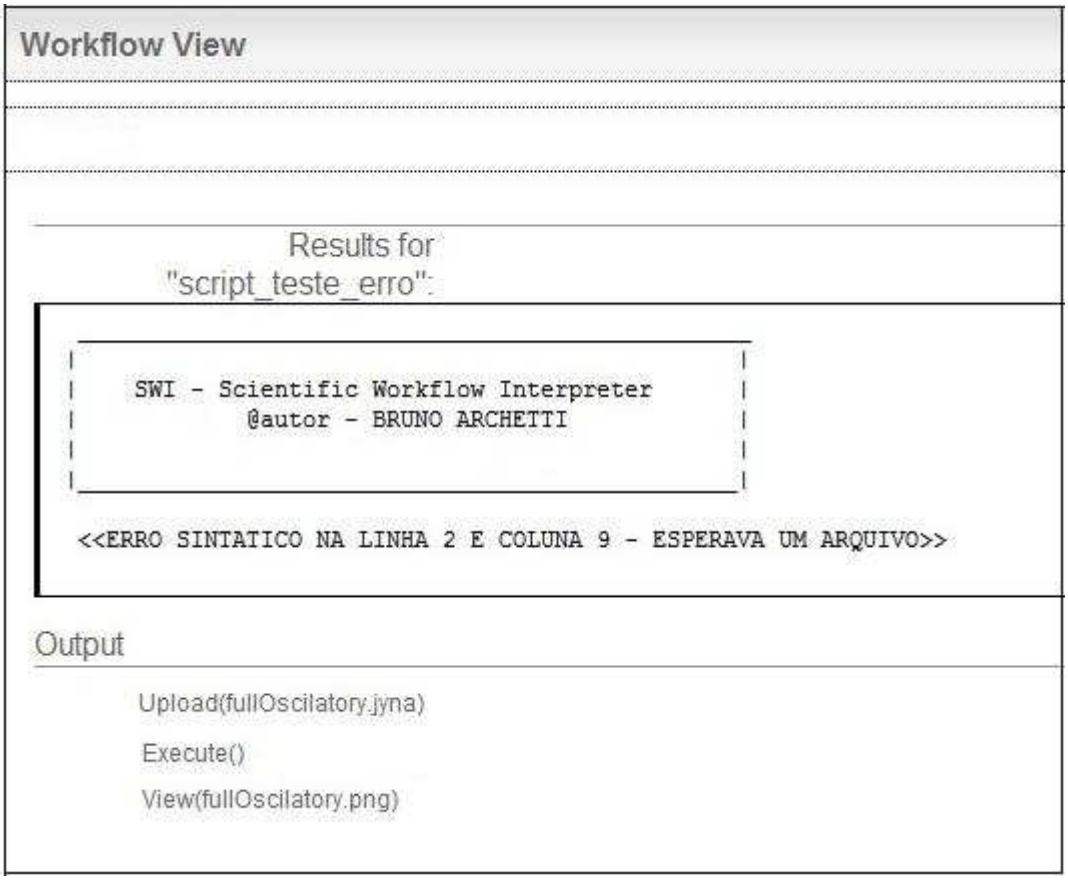

Figura 5.26 – Resultado da execução de todos os passos do script incorreto.

Evidentemente que na figura 5.25 não temos a visualização dos dados do modelo e o respectivo gráfico, porque o modelo não foi executado e o arquivo solicitado por parâmetro na função *view* não existe no servidor, pois não foi criado.

A figura 5.26 mostra como ficou o diretório do projeto após a execução de todas as funções. Podemos perceber que apenas o arquivo "*script_teste_erro.swi"* e o modelo *"fullOscilatory.jyna"* estão no diretório dentro do projeto. Os outros não chegaram a ser criados devido à identificação do erro no comando *execute.*

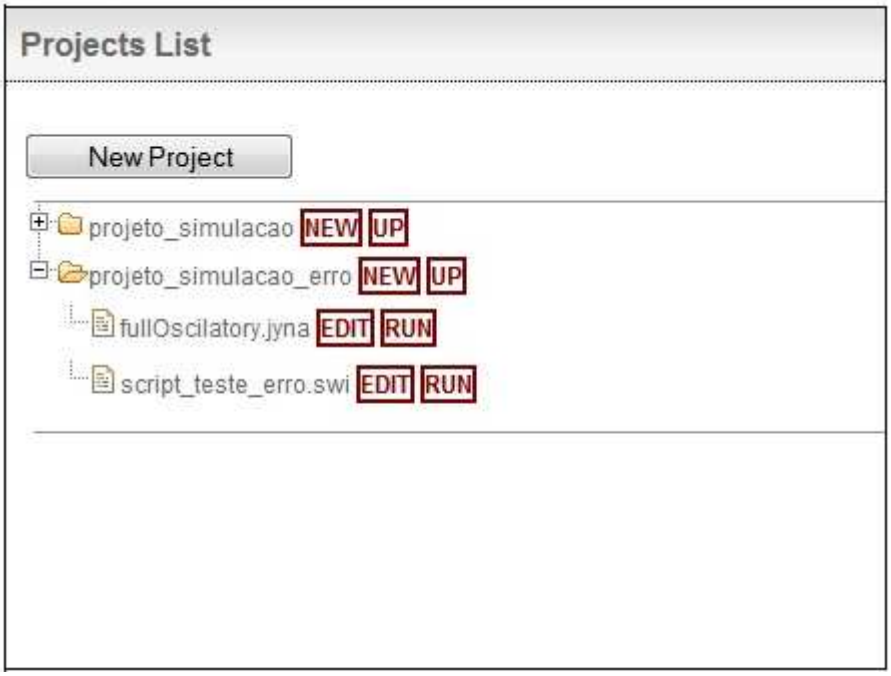

Figura 5.27 – Arquivos utilizados no projeto visualizados na árvore.

5.4 – Limitações da Arquitetura Proposta

Existem algumas limitações na arquitetura adotada para a construção do SWI, por ele atuar em parte como um compilador. O script definido pelo usuário deve ser lido totalmente para gerar o código intermediário interpretado pelo Servlet, logo o processo de interpretação fica defasado caso a linguagem fonte passe a aceitar situações em que o fluxo precise de uma determinada condição para evoluir ou não. Pois como a interpretação já foi processada totalmente, não é possível receber respostas em tempo real pra interferir na interpretação do código pelo SWI e verificar qual decisão a ser tomada. A solução para o problema é deixar o SWI funcionando totalmente como um interpretador. Nesse caso, o código intermediário não seria mais gerado, e ao invés de gravar os comandos na lista, eles seriam executados um a um interagindo com o Servlet de forma sequencial. Feito isso, o retorno de uma determinada condição poderia ser analisado e decidido se executa ou não o próximo passo.

Outro ponto desfavorável é a forte relação entre o SWI e o Servlet. Atualmente é o Servlet quem executa as funções do sistema, e caso o SWI fosse modificado para aceitar mais alguma funcionalidade, o Servlet deveria ser adaptado para executar essa nova alteração. Um exemplo é a análise semântica realizada pelo Servlet. É nele que a verificação da

extensão do arquivo a ser executado ocorre. Se um modelo do tipo *".jyna"* é identificado, o Servlet executa o simulador JynaCore API. Caso então um novo simulador seja inserido, como por exemplo o AGOS, o Servlet deve ser adaptado para identificar um modelo do tipo *".xml"* e executar o AGOS. Portanto essa dependência entre o Servlet e o SWI é prejudicial para a evolução do ambiente e para o suporte a diversos tipos de modelos. O ideal é que todas as execuções dos simuladores fossem realizadas no SWI. E o Servlet ficasse apenas com a responsabilidade de rodar o arquivo contendo a linguagem fonte.

Capítulo 6 – Conclusão

Os recursos computacionais, aplicados às atividades de pesquisa científica, estão colaborando para o crescimento da produção científica em diversas áreas de conhecimento. Os experimentos científicos são desenvolvidos em diversos componentes que são combinados para gerar um resultado satisfatório. Para definir essa orquestração de seqüências de processos são utilizados os Sistemas de Workflows Científicos, que são considerados por alguns autores o principal recurso do experimento científico.

Na sua estruturação o trabalho distingue três abordagens para as questões relacionadas aos workflows científicos, são elas: a natureza do problema, o ferramental teórico para a formalização dos workflows e as principais *engines* para a execução dos workflows científicos, além da linguagem construída para definição dos processos. Desta forma, no capítulo 2 são apresentados conceitos relacionados com os workflows científicos e ferramentas para a construção e execução dos mesmos. No capítulo 3, é apresentada a linguagem BPEL para definição de workflows e são detalhadas as suas principais construções o que permite um melhor entendimento de como os workflows são definidos e modelados. São mostradas também as principais *engines* para construção e execução do BPEL. No capítulo 4, o AGOS e o JynaCore API, ferramentas de uso no Portal, são brevemente detalhadas. Por fim, no capítulo 5 apresentamos a contribuição deste trabalho, onde a aplicação desenvolvida para automatizar, gerenciar e apoiar os experimentos científicos realizados no Portal Fisiocomp é detalhada e exemplificada.

A aplicação desenvolvida, detalhada no capítulo 5, pode ser facilmente expandida pela contribuição de outros alunos nos sentido de aumentar as funcionalidades dos workflows do Portal Fisiocomp que podem ser gerenciadas.

Algumas melhorias que podem ser feitas no SWI, a título de trabalhos futuros são a expansão da linguagem script para que suporte outras estruturas de controle tais como “while” e “foreach”, como suportadas por BPEL; a definição de variáveis para composição de condições de controle; a definição da execução em paralelo das atividades e a interação com Grids Computacionais.

A integração definitiva do SWI com a versão em produção do Portal Fisiocomp, embora facilitada pela documentação disponibilizada nessa monografia, é deixada para trabalhos futuros.

Uma linha de investigação no sentido de desenvolver uma solução alternativa para

a que foi apresentada aqui seria a de adaptar o ambiente do Portal para que se pudesse aplicar as ferramentas que fazem uso da linguagem BPEL, herdando assim todos os recursos já disponíveis para o uso dessa linguagem.

A revisão de conceitos apresentada nesta monografia propicia um melhor entendimento do contexto de workflow científico e podem ser estudados e aplicados na realização de pesquisas reais. Além disso, as ferramentas apresentadas e detalhadas podem ser utilizadas para atender às necessidades de grupos de pesquisa.

Referências

AHO, A., SETHI, R., ULLMAN, J., Zhao, J.; "**Compiladores Princípios, Técnicas e Ferramentas**", AT & T Bell Laboratories Murray Hill, New Jersey, 1986.

AMORIM, Ronan, **Ferramenta de Transformação para Equações Diferenciais Ordinárias descritas em XML e sua aplicação em Modelagem Cardíaca**. 2007.

ARAUJO, Renata Mendes de; BORGES, Marcos Roberto da Silva. **Sistemas de Workflow**. 2001. Disponível em chord.nce.ufrj.br/cursos/teesi/textos/apostilaJai2001div.pdf Acesso em: 30 de setembro de 2009.

BEANSHELL in**:** http://www.beanshell.org, 2009.

BONIFÁCIO, Aldemon, **Análise de Ferramentas Computadorizadas para Suporte à Modelagem Computacional - Estudo de Caso no Domínio de Dinâmica dos Corpos Deformáveis**. 2008.

BPEL, **Web Services Business Process Execution Language Version 2.0**. 2007. Disponível em: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html Acesso em: 08 de setembro de 2009.

Building Flexible Enterprise Process Using Oracle Business Rules and BPEL Process Manager, White Paper, 2005. Disponível em: <http://www.oracle.com/technologies/soa/oracle_business_integration.pdf>. Acessado em 08 de setembro de 2009.

CELLML, 2009. **CellML Portal**. Disponível em: <http://www.cellml.org/>. Acesso em: 16 de novembro de 2009.

DIGIAMPIETRI, Luciano Antônio. **Gerenciamento de *workflows* científicos em bioinformática**. Campinas, 2007. 112f. Tese (Doutorado em Ciência da Computação) – Instituto de Computação, Universidade Estadual de Campinas.

EMMERICH, W.; Butchart, B.; Chen, L.; Wassermann,B.; and Price, S.; "**Grid ServiceOrchestration using the Business Process Execution Lenguage (BPEL)**" in: http://www.cs.ucl.ac.uk/staff/w.emmerich/publications/JoGC/Workflow/bpel.pdf, 2006.

FISHER, Paul. **A user guide to the Taverna workflow workbench**. 2007. Disponível em: <workflows.mygrid.org.uk/repository/myGrid/PaulFisher/**Taverna**_User_**Guide**.ppt>. Acesso em: 08 novembro 2009.

FISIOCOMP, 2009. **FISIOCOMP: Laboratory of Computational Physiology**. Disponível em: <www.fisiocomp.ufjf.br>. Acesso em: 16 de novembro de 2009.

GREENWOOD, M., Goble, C., Stevens, R., Zhao, J., Addis, M., Marvin, D., Moreau, L. e inn, T., "**Provenance of e-Science Experiments - experience from Bioinformatics**", Proceedings of the UK OST e-Science 2nd AHM, 2003.

KEPLER Project. **Kepler Project**. 2007. Disponível em: <http://kepler-project.org/>. Acesso em: 05 de outubro de 2009.

KNOP, Igor, **Modelagem e Simulação de Sistemas Dinâmicos: Uma ferramenta baseada em Dinâmica de Sistemas aplicada a Gerenciamento de Processos de Software.** 2009.

KÖNIG, Dieter. (2006). **WS-BPEL – Standards Roadmap** . Disponível em: <www.citt-online.com/downloads/exp4/Dieter%20Koenig%20-%20WS-BPEL%20Standards.pdf>. Acessado em 08 de setembro de 2009.

LAWISCH, Eduardo André, **BPEL: Uma Padronização para especificação técnica de processos.** 2008. Disponível em: www.inf.unisc.br/repotcs/arquivos_tc/TCII_Eduardo_Andre_Lawisch.doc Acesso em: 08 de setembro de 2009.

LEMOS, M. "**Workflow para Bioinformática**". Tese de Doutorado – PUC-Rio, Rio de Janeiro, Brasil, 2004

MACHADO, Larissa Aparecida. Monografia de conclusão de curso de Ciência da Computação**, Workflow Científicos**. 2007.

MATTOS, A.; Coutinho, F; Ruberg, N.; Manuel, S., **Gerência de Workflows Científicos: Uma Análise Crítica No Contexto da Bioinformática.** 2008

MATTOSO Marta; Werner, Cláudia; Travassos, Guilherme Horta; Braganholo, Vanessa; Murta, Leonardo; Ogasawara, Eduardo; Oliveira, Frederico; Martinho, Wallace. **Desafios no apoio à composição de experimentos científicos em larga escala**, 2009.

MEDEIROS, C., VOSSEN, G., WESKE, G., 1995, "**WASA: A Workflow-Based Architecture to Support Scientific Database Applications**", In: Proceedings of the 6th*DEXAConference* 1995, pp. 574-583, London, England, September.

MORO, M., "**Workflow**". Disponível em http://www.inf.ufrgs.br/~mirella/workflow/work.html Acesso em: 30 de setembro de 2009.

NAKASHIMA, Caio, **BPEL: Construindo um padrão Business Processo baseado em Web Services.** 2008. Disponível em: www.dainf.cefetpr.br/~caio/aula/ws/BPELOverviewCaio.ppt Acesso em: 01 de junho de 2009.

OSSES, J.; CARVALHO, J.; CÂMARA, G. **Arquiteturas Cliente-Servidor para Bibliotecas Geográficas Digitais**. Disponível em: http://www.dpi.inpe.br/nsf-cnpq/cliente_servidor.pdf. Acessado em: 23 de novembro de 2009.

SUMPTER, R., Whitepaper on Data Management, Lawrence Livermore National Laboratory. The IEEE Metadata Workshop, 1994.

TAVERNA Project. **Taverna 1.7 Manual**, Dezembro 2007.

TADEU, Tiago. Monografia de conclusão de curso de Ciência da Computação, **Arquitetura Orientada a Serviços e Workflow**. 2009.

VIVACQUA, Luiz, **Estratégias para o escalonamento dinâmico de workflows em Grid**, 2006.

WfMC, WfMC – Workflow Management Coalition "**The workflow referece model**". Disponível em http://www.wfmc.org. 2008.

WIKIPEDIA **BPEL** in**:** http://en.wikipedia.org/wiki/BPEL, 2009.

WRM, Workflow Management Coalition. **The Workflow Reference Model**. 1995. Disponível em: http://www.wfmc.org/standards/docs/tc003v11.pdf Acesso em: 30 de setembro de 2009.