Universidade Federal de Juiz de Fora
Programa de Mestrado em Modelagem Computacional

Análise de Ferramentas Computadorizadas para Suporte à Modelagem Computacional - Estudo de Caso no Domínio de Dinâmica dos Corpos Deformáveis

Por

Aldemon Lage Bonifácio

JUIZ DE FORA, MG - BRASIL
AGOSTO DE 2008

ANÁLISE DE FERRAMENTAS COMPUTADORIZADAS PARA SUPORTE A MODELAGEM COMPUTACIONAL - ESTUDO DE CASO NO DOMÍNIO DE DINÂMICA DOS CORPOS DEFORMÁVEIS

Aldemon Lage Bonifácio

DISSERTAÇÃO SUBMETIDA AO PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM COMPUTACIONAL DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS (M.SC.) EM MODELAGEM COMPUTACIONAL.

Aprovada por:

Prof. Ciro de Barros Barbosa, D.Sc.

(Orientador)

Prof. Flávio de Souza Barbosa, D.Sc.

(Co-orientador)

Prof[a]. Regina Maria Maciel Braga, D.Sc.

Prof. Roque Luiz da Silva Pitangueira, D.Sc.

Prof[a]. Patrícia Habib Hallack, D.Sc.

JUIZ DE FORA, MG - BRASIL

AGOSTO DE 2008

Bonifácio, Aldemon Lage

Análise de ferramentas computadorizadas para suporte a modelagem computacional - estudo de caso no domínio de dinâmica dos corpos deformáveis / Aldemon Lage Bonifácio. - 2008.

169 f.: il.

Dissertação (Mestrado em Modelagem Computacional) - Universidade Federal de Juiz de Fora, Juiz de Fora, 2008.

1. Ciência da computação. 2. Engenharia civil. Título.

CDU 681.3

A sabedoria da vida não está em
fazer aquilo que se gosta,
mas gostar daquilo que se faz.
Leonardo da Vinci

DEDICATÓRIA

Dedico este trabalho aos meus pais.

AGRADECIMENTOS

À minha esposa, Sandra, pela paciência e pelo apoio durante toda a jornada em busca do conhecimento.

Aos meus filhos, Eybraian e Aryanne, pelos momentos de interrupção durante os estudos para que eu não me esquecesse de dar-lhes a devida atenção.

À minha família, pela busca de uma boa formação e por depositarem em mim a confiança em todos os momentos.

Ao meu orientador, Prof. Ciro Barbosa, pelo companheirismo, atenção e incentivos dispensados. Pela paciência com meus erros e pelo entusiasmo com os meus acertos.

Ao meu co-orientador, Prof. Flávio Barbosa, por toda a atenção, estímulos e ensinamentos.

Ao amigo Samuel por ajudar-me nos momentos de dificuldades durante o mestrado.

Aos meus colegas de mestrado pela oportunidade de tê-los conhecido e convivido em momentos tão diversos.

Aos professores do curso de Mestrado em Modelagem Computacional, pelos conhecimentos passados e dedicação na sua tarefa de formar mestres.

À Natália, pela prestatividade e ajuda dispensada nos momentos de dúvidas nas questões burocráticas do mestrado.

Enfim, agradeço a todos que de forma direta ou indireta me apoiaram, auxiliaram e contribuíram não só na realização deste trabalho, mas em todas as atividades realizadas durante o mestrado, meus sinceros agradecimentos. Muito obrigado a todos!

Resumo da Dissertação apresentada à UFJF como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ANÁLISE DE FERRAMENTAS COMPUTADORIZADAS PARA SUPORTE A MODELAGEM COMPUTACIONAL - ESTUDO DE CASO NO DOMÍNIO DE DINÂMICA DOS CORPOS DEFORMÁVEIS

Aldemon Lage Bonifácio

Agosto/2008

Neste trabalho é feita uma análise de ferramentas computadorizadas para auxílio à atividade de modelagem computacional. Tais ferramentas visam solucionar dificuldades tais como a baixa reutilização de resultados obtidos, a integração das tarefas que compõem o experimento e o gerenciamento das várias etapas do processo de modelagem.

Dá-se o nome de *Workflow* Científico para o conjunto das tarefas que compõem um experimento em modelagem computacional. Ferramentas denominadas Sistemas de *Workflow* Científico são responsáveis por automatizar a execução desses processos.

Sistemas de *Workflow* Científico são ainda pouco difundidos. Isso se deve em parte ao esforço necessário para realização do aporte tecnológico e conceitual. Ferramentas dessa natureza disponibilizam um grande número de funcionalidades, muitas delas fazendo uso de tecnologias recentes e desconhecidas para grande parte dos pesquisadores em modelagem computacional. Este trabalho visa fazer um estudo detalhado dessas funcionalidades e uma avaliação da sua aplicabilidade para solução das dificuldades supracitadas. A abordagem usada para estudo das funcionalidades é a construção de uma taxonomia de conceitos e funções no contexto de Sistemas de *Workflow* Científico.

A avaliação da aplicabilidade desse tipo de ferramenta é feita através do desenvolvimento de um estudo de caso de modelagem computacional no domínio de Dinâmica dos Corpos Deformáveis usando o Sistema de *Workflow* Científico Kepler. Os critérios para escolha dessa ferramenta também são discutidos nesse trabalho.

Os principais aspectos de avaliação são a eficiência na definição e a performance

da execução dos experimentos. Para as funcionalidades com avaliação negativa são fornecidos guias ou artefatos de *software* que minimizem esse problema.

Orientador : Ciro de Barros Barbosa
Co-orientador : Flávio de Souza Barbosa
Programa: Mestrado em Modelagem Computacional

Abstract of Dissertation presented to UFJF as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

ANALYSIS OF COMPUTERIZED TOOLS FOR COMPUTATIONAL MODELING SUPPORT - CASE STUDY OF THE DYNAMICS SYSTEM DOMAIN

Aldemon Lage Bonifácio

Agosto/2008

In this work, computer tools are analyzed regarding their support to computational modeling activities. Those tools aim to solve problems such as the lack of reuse of obtained results, the integration of tasks that compound the experiments, and the management of the various steps of the modeling process.

The concept "Scientific Workflow" designates the collection of tasks that shapes an experiment in computational modeling. Tools called Scientific Workflow Systems are responsible for automating the execution of these processes.

Scientific Workflow Systems are not very spread yet. This is, in part, due to the effort necessary to learn the new techniques involved. Such tools implement a great number of functionalities, many of them using recent and unknown technologies to most researchers in computational modeling. This work is intended to provide a detailed study of such functionalities and an evaluation of their applicability to solve the difficulties we have mentioned. The approach to carry out the study of functionalities is to build a taxonomy of concepts and functions, in the context of Scientific Workflow Systems.

The evaluation of such tools is done by developing a computational modeling case study in the domain of Dynamic Systems, using the Scientific Workflow System named Kepler. The criteria used for choosing this tool will also be discussed.

The main aspects of the evaluation are the efficiency in the definition of models and the performance of their execution. To those functionalities that do not couple with our expectations, we will provide developing guides or software artifacts that will improve them.

Supervisor: Ciro de Barros Barbosa

Co-supervisor: Flávio de Souza Barbosa

Department: Computational Modelling

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Glossário

Ad hoc expressão latina que quer dizer "com este objetivo". Geralmente significa uma solução designada para um problema ou tarefa específicos, que não pode ser aplicada em outros casos. Um processo *ad hoc* consiste em um processo em que nenhuma técnica reconhecida é empregada e/ou cujas fases variam em cada aplicação do processo. 3, 110

Código aberto criado pelo OSI (*Open Source Iniciative*), a partir do texto orginal da *Debian Free Software Guidelines*, determina que um programa de código aberto deve garantir distribuição livre, código fonte, trabalhos derivados, integridade do autor do código fonte, distribuição da licença, entre outros. 4–6, 42, 54

Eclipse® IDE para desenvolvimento em linguagem Java/C/C++. 102

Framework estrutura de suporte definida para que um outro projeto de *software* possa ser organizado e desenvolvido. 30, 44, 55, 87, 101

GL graus de liberdade. 65–70, 72, 79

Grid Computação em Grid é um modelo computacional capaz de alcançar uma alta taxa de processamento e armazenamento dividindo as tarefas entre diversas máquinas através de uma rede, que formam uma máquina virtual.. 2, 3, 5, 6, 29, 40, 50, 51, 55, 58, 100, 110, 113

Hardware parte física do computador, ou seja, é o conjunto de componentes eletrônicos, circuitos integrados e placas, que se comunicam através de barramentos. 21, 31, 45, 51

HTML (*HyperText Markup Language*) linguagem de marcação para hipertexto com a qual as páginas Web são feitas. 23

HTTP (*Hypertext Transfer Protocol*) protocolo para envio e recebimento de dados pela internet. 44, 45

JDK (*Java Development Kit*) conjunto de programas para desenvolvimento de códigos em Java.. 103

JRE (*Java Runtime Environment*) conjunto de bibliotecas necessárias para executar códigos Java.. 53

MWC modelo de *workflow* científico. 2

Script linguagem interpretada que age de dentro de um programa e/ou de outras linguagens de programação, como forma de extensão. 9, 29, 40, 42, 43, 63

Software programa ou aplicativo de computador. 1–3, 6, 29–31, 42, 45, 54, 79, 80, 110

Sun Microsystems empresa desenvolvedora de tecnologias Java. 4

SWC sistema de *workflow* científico. 2

XML (*Extensible Markup Language*) linguagem de marcação extensível, utilizada para armazenamento de dados. 23, 40, 41, 44

Capítulo 1

Introdução

Esse é um trabalho multidisciplinar que investiga o uso de sistemas computacionais para apoio ao processo de modelagem computacional aplicado à engenharia, mais especificamente, a área de sistemas Dinâmicos dos Corpos Deformáveis.

No contexto dessa pesquisa é importante observar a modelagem computacional como sendo um dos passos de uma abordagem particular para o processo de experimentação científica. Nesse processo o fenômeno a ser estudado é simulado em um computador. Para esse tipo de experimentação foi definido o termo "*in silico*"[1].

Esse ambiente multidisciplinar envolve pelo menos três especialidades: o conhecimento e a compreensão dos fenômenos físicos que se deseja simular, a criação de modelos matemáticos para tais fenômenos e o conhecimento de sistemas computacionais.

Um cenário comum na pesquisa baseada em experimentos *in silico* é encontrar trabalhos deficientes em algumas das especialidades envolvidas. Algumas conseqüências desse cenário são relacionadas às deficiências na especialidade de sistemas computacionais. Essas deficiências são de interesse destacar nesse trabalho, e por esse motivo são listadas a seguir:

– a baixa reutilização de programas e sistemas já desenvolvidos, que compromete a eficiência do desenvolvimento dos *softwares* para simulação dos modelos;

[1] *In silico* é uma expressão comumente usada na simulação computacional e áreas correlatas para designar algo que ocorreu em ou através de uma simulação computacional.

– a dificuldade de integração dos sistemas computacionais usados nas diversas etapas do experimento *in silico*;

– e a dificuldade de gerenciamento dos vários processos envolvidos nas etapas de construção e execução de um projeto que envolva diversos sistemas computacionais.

O processo automatizado que gerencia processos e dados para criar soluções computacionais para um problema científico é chamado de *Workflow* Científico. Alguns exemplos de etapas desse processo são a definição dos dados do experimento, a simulação do fenômeno e a análise dos resultados.

Para um *Workflow* Científico definido com as particularidades de um experimento específico dá-se o nome de Modelo de *Workflow* Científico (MWC). Outro conceito importante no contexto desse trabalho é o conceito de Sistemas *Workflow* Científico (SWC), que é uma classe de ferramentas de *software* projetadas para auxiliar na definição e execução dos MWCs.

Relacionado ao ambiente de utilização de um SWC, existem diferentes tipos de usuários, a saber: usuários que definem e usam os recursos disponíveis em um SWC, usuários que apenas executam os MWCs e os usuários que criam componentes para seus MWCs com novas tarefas não fornecidas pelos SWCs.

Há também no mercado uma vasta variedade de SWCs disponíveis para uso, que fornecem um grande espectro de funcionalidades para apoio à definição e à execução de MWCs. Esses sistemas variam em grau de generalidade, quanto à área de aplicação, além de variarem quanto à facilidade de instalação e uso.

Os SWCs disponibilizam blocos de construção básicos para definição dos MWCs, que, eventualmente, podem não disponibilizar toda a funcionalidade necessária a um MWC específico. Assim, os SWCs também suportam mecanismos para extensão de suas funcionalidades, para o qual demandam maior ou menor especialização e atendem em maior ou menor grau as demandas das aplicações dos usuários.

Geralmente, os mecanismos para melhorar o desempenho da execução dos modelos demandam maior conhecimento da especialidade sistemas computacionais. Os SWCs encontrados na literatura apresentam recursos para facilitar o uso de tais mecanismos, como por exemplo, interfaces com Grids Computacionais e Paralelização

da execução das atividades dos MWCs.

1.1 Objetivo

O objetivo deste trabalho é avaliar a aplicabilidade de SWCs visando solucionar as dificuldades encontradas por grupos de modelagem computacional na área de sistemas Dinâmicos dos Corpos Deformáveis. Para atingir esse objetivo será realizado um estudo dos SWCs para identificar e documentar as possíveis funcionalidades e avaliar a sua aplicabilidade.

Com relação à eficiência no desenvolvimento dos MWCs, pretende-se avaliar a facilidade de um não especialista em sistemas computacionais, em compor novos MWCs, reutilizando componentes disponíveis no SWC ou partes de outros MWC desenvolvidos por outros pesquisadores.

Com relação a integração de sistemas computacionais pretende-se avaliar a facilidade de se integrar ao SWC, *software*s disponíveis no mercado, como o Matlab®[2] e o R®[3], e também programas *ad hoc* desenvolvidos por outros programadores, fazendo uso de tecnologias, tais como *web services* e Grid.

Com relação ao gerenciamento da construção e execução de um MWC, pretende-se investigar o suporte que é dado para a identificação de componentes disponíveis, a composição desses componentes e a depuração do MWC.

Outro objetivo dessa pesquisa é selecionar um SWC adequado, com uma justificativa da escolha, e oferecer exemplos práticos de sua utilização na área de sistemas Dinâmicos dos Corpos Deformáveis. Para as funcionalidades com avaliação negativa, quanto à facilidade de uso, serão fornecidos guias ou artefatos de *software* que minimizem esse problema.

[2]Matlab®: *software* interativo de alto desempenho voltado para o cálculo numérico. `http://www.mathworks.com`

[3]R®: programa gratuito para computação estatística e geração de gráficos. `http://www.r-project.org`

1.2 Estado da arte em Sistemas de *Workflow*

Esta seção apresenta algumas pesquisas relacionadas com o contexto dessa dissertação. Tais pesquisas incluem o desenvolvimento de ferramentas para implementação dos conceitos de *Workflow* Científico e classificações taxonômicas de tais conceitos.

Pesquisadores de várias áreas do conhecimento fazem uso dos SWCs e produzem documentos sobre esses sistemas, tutoriais de uso e diversas bibliotecas que agregam novas tarefas aos sistemas. Entre as áreas de conhecimento com relatos de uso de *Workflow* Científicos destacam-se a biologia [24, 46], ecologia [24, 46], química [46], geologia [24, 46], oceanografia [46], filogenia [46], entre outros [29, 55].

Existem atualmente inúmeros SWCs disponíveis para uso, entretanto por uma questão de objetividade foram selecionadas aqui algumas dessas ferramentas que ilustram satisfatoriamente os conceitos mais relevantes. As ferramentas selecionadas são: Kepler [18, 29], Triana [24, 53, 59], Taverna [24, 56, 57] e Vistrails [21, 54], detalhados a seguir.

A ferramenta Kepler é um SWC desenvolvido pela Universidade da Califórnia, Berkeley nos EUA, em 2005. O Kepler foi projetado para trabalhar com diversas áreas de pesquisa científica, dentre os quais cita-se a bioinformática, a ecoinformática e a geoinformática. Trata-se de uma ferramenta desenvolvida em Java®[4] e de código aberto, construída a partir do sistema Ptolemy [7, 14], que é um ambiente para estudos de modelos computacionais, desenvolvido pela mesma instituição. Atualmente a versão disponível deste sistema é a *beta3*.

O Kepler é uma ferramenta que tem uma boa estruturação de código e interface, além de suportar um grande número de funcionalidades relacionadas aos SWCs.

Atualmente é desenvolvido pela união de projetos colaborativos com participação de instituições como: SEEK[5], *SDM Center/SPA*[6], Ptolemy II, GEON[7], ROADNet[8],

[4] Java®: linguagem de programação orientada a objetos. Marca registrada da Sun Microsystems. [27, 28] `http://java.sun.com`

[5] *Science Environment for Ecological Knowledge*: repositório de dados e definições sobre ecoinformática

[6] *SDM Center/SPA*: ministério de energia dos Estados Unidos

[7] GEON: projeto colaborativo sobre geociência

[8] ROADNet: projeto da Universidade da Califórnia sobre ciência multidisciplinar nas áreas de

CIPRES[9], Kepler/CORE Project, entre outros.

Outro sistema voltado para o desenvolvimento de *workflows* científicos é o sistema Triana, mantido pela Universidade de Cardiff no Reino Unido. Essa ferramenta é de código aberto e desenvolvida utilizando a linguagem Java®. Atualmente a versão disponível deste sistema é a 3.2.3.

O Triana permite que seus usuários montem programas a partir de um conjunto de blocos de construção, que são selecionados para compor um MWC na área de trabalho, para o qual os blocos são arrastados. Entre os blocos disponíveis estão os que realizam processamentos de sinais, manipulam imagens e vídeos, fazem processamento de textos e os que fazem análises estatísticas.

Como descrito em seu guia na referência [58], o Triana é recomendado para executar tarefas automáticas e repetitivas, como por exemplo uma procura e substituição em vários arquivos textos.

O Triana faz parte da PPARC[10], fundada pelo projeto GridOneD[11], que tem o objetivo de criar aplicações para Grids feitas em Java® e integradas ao Triana. Além disso, o Triana também é utilizado como aplicação para interação no GridLab[12], que é um ambiente Grid para desenvolvimento de grandes projetos.

Outra opção para criação de MWCs é o sistema Taverna, que foi criado pelo projeto myGrid[13], do Reino Unido. Esse sistema foi planejado para trabalhar principalmente com projetos científicos do domínio de bioinformática, e teve sua primeira versão de testes disponibilizada em junho de 2003. O Taverna é um sistema de código aberto e escrito utilizando linguagem Java®. Atualmente a versão disponível deste sistema é a 1.7.0.

O Taverna fornece um mecanismo para validação do MWC que é definido na linguagem Scufl (*Simple Conceptual Unified Flow Language*) [47]. A linguagem Scufl permite que o usuário descreva uma tarefa conceitual através de uma entidade simples, além de ter um mecanismo de tratamento de falha básico.

---

geografia, oceanografia, ecologia e outros

[9] CIPRES: projeto colaborativo de pesquisa na área de filogenia

[10] *Pikes Peak Red Cross* - `http://www.pparc.org`

[11] GridOneD: ambiente Grid voltado para o Triana. `http://www.gridoned.org`

[12] GridLab: ambiente Grid. `http://www.gridlab.org`

[13] myGrid: ambiente Grid para área de bioinformática. `http://www.mygrid.org.uk`

Por fim, um último exemplo de SWC é o Vistrails [49], que provê suporte a exploração e visualização de dados. Esse sistema é desenvolvido na Universidade de Utah, nos EUA, utilizando a linguagem Python®[14]. O Vistrails, assim como os anteriores, também é um *software* de código aberto. Sua primeira versão foi liberada em janeiro de 2007. Atualmente (Fevereiro de 2008), a versão disponível deste sistema é a 1.0 *revision* 1024.

O Vistrails foi planejado para suprir as necessidades de aplicações científicas exploratórias, principalmente processamento de imagens. Como seus usuários tendem a gerar e avaliar as hipóteses sobre os dados estudados, uma série de diferentes MWC são criados enquanto o modelo é ajustado em um processo interativo. Por isso, o Vistrails foi construído para ser capaz de gerenciar os vários MWCs, nos quais há poucos processos repetitivos e algumas mudanças entre os modelos. Além dessa gerência, o Vistrails também organiza a criação, a execução, o compartilhamento de visualizaçãoes complexas e a mineração de dados. Vale citar que o sistema Vistrais foi o pioneiro na implementação do conceito de proveniência, discutido na Seção 2.4.

No estudo de caso apresentado no Capítulo 6, adotou-se o sistema Kepler, cuja escolha será justificada no Capítulo 4. Objetivou-se ilustrar no sistema Kepler os conceitos que serão apresentados no Capítulo 2, sobre SWCs, e de avaliar as funcionalidades comentadas na introdução. Essa ferramenta permitirá também ilustrar a criação de um MWC no domínio de Dinâmica dos Corpos Deformáveis.

Com relação a classificação de conceitos de SWCs, outros trabalhos também discutem a construção de taxonomias na área de *workflow* científico, tais como os citados nas referências [36] e [77]. No entanto, o primeiro desenvolve uma taxonomia das funcionalidades de *workflow* científico voltada apenas para características específicas da área de computação em ambiente Grid. O segundo, trata da construção de uma taxonomia para as características apenas relacionadas ao conceito de proveniência. Nesse trabalho é realizada uma classificação mais abrangente da funcionalidades dos SWCs, discutidas no Capítulo 2.

[14]Python®: linguagem de programação orientada a objetos. `http://www.python.org`

1.3 Organização do texto

Este trabalho é composto por 7 capítulos, incluindo este capítulo que dá uma visão geral sobre o trabalho desenvolvido.

O conceito de *Workflow* Científico é detalhado no Capítulo 2, bem como a apresentação da taxonomia.

No Capítulo 3 são apresentados conceitos que dão suporte tecnológico às categorias da taxonomia.

No Capítulo 4 é apresentado o sistema Kepler. Esse capítulo destaca a criação de MWCs especificamente nesse sistema, além de sua interface e componentes básicos. As categorias da taxonomia, suportadas pelo Kepler, também são comentadas.

No Capítulo 5 são expostos conceitos gerais a respeito da modelagem computacional de sistemas deformáveis, analisando-se as equações diferenciais ordinárias que regem o comportamento dinâmico desses sistemas em modelos de um ou mais graus de liberdade.

O estudo de caso é abordado no Capítulo 6, no qual são discutidos os MWCs criados, os algoritmos usados e as facilidades de uso do sistema Kepler.

Por fim, as conclusões e principais contribuições oferecidas por este trabalho, bem como algumas sugestões para futuros trabalhos de pesquisa envolvendo o tema discutido, são apresentadas no Capítulo 7.

Capítulo 2

Sistemas de *workflow* científico

A concepção formal de *workflow* iniciou-se no mundo dos negócios sob a forma de ferramentas e tecnologias para gerenciar e organizar os fluxos de trabalho propostos por empresas comerciais. A principal preocupação com os *workflows* de negócios se relaciona com a segurança e a integridade da seqüência das ações [33].

Em 1996, a WfMC (*Worflow Management Coalition*) definiu *workflow* como: "a automação de processos de negócios, no todo ou em partes, no qual documentos, informações ou tarefas são passadas de uma para outra ação, de acordo com o conjunto de regras de procedimentos" [36].

Uma definição mais recente de *workflow* denota a execução controlada de múltiplas tarefas em um ambiente de elementos processados de forma distribuída. *Workflows* representam um conjunto de atividades que podem ser executadas com suas relações interdependentes, suas entradas e suas saídas [41].

A referência [36] define *workflow* científico como uma automação de processos científicos no qual tarefas são estruturadas baseadas nas suas dependências de controle e dados. Ilkay Altintas [30] define *workflow* científico como um processo automatizado que combina processos e dados em um conjunto de passos estruturados para implementar soluções computacionais para um problema científico.

Um *workflow* científico de uma maneira geral necessita coletar, gerar e analisar uma grande quantidade de dados heterogêneos, que é, normalmente, a essência do seu trabalho. Assim, *workflow* científico tem como sua principal propriedade a

flexibilidade, que pode ser definida como a facilidade de se dar manutenção em MWCs, o uso de dados heterogêneos, o acompanhamento de processamentos e a comparação de resultados. Essas características são de grande importância para acelerar o trabalho do pesquisador.

Inicialmente, os *workflows* científicos eram criados através de *script* de automação das tarefas. Esses *script*s eram construídos utilizando-se linguagens textuais. Em versões mais recentes, a inclusão de sistemas gráficos facilitou muito a construção e manutenção de *workflows*.

Uma ilustração da interface gráfica de um SWC pode ser vista na Figura 2.1. No exemplo tem-se, à direita, um MWC, à esquerda, um conjunto de componentes disponíveis para serem adicionados ao modelo e uma caixa com o estado da execução do *workflow*.

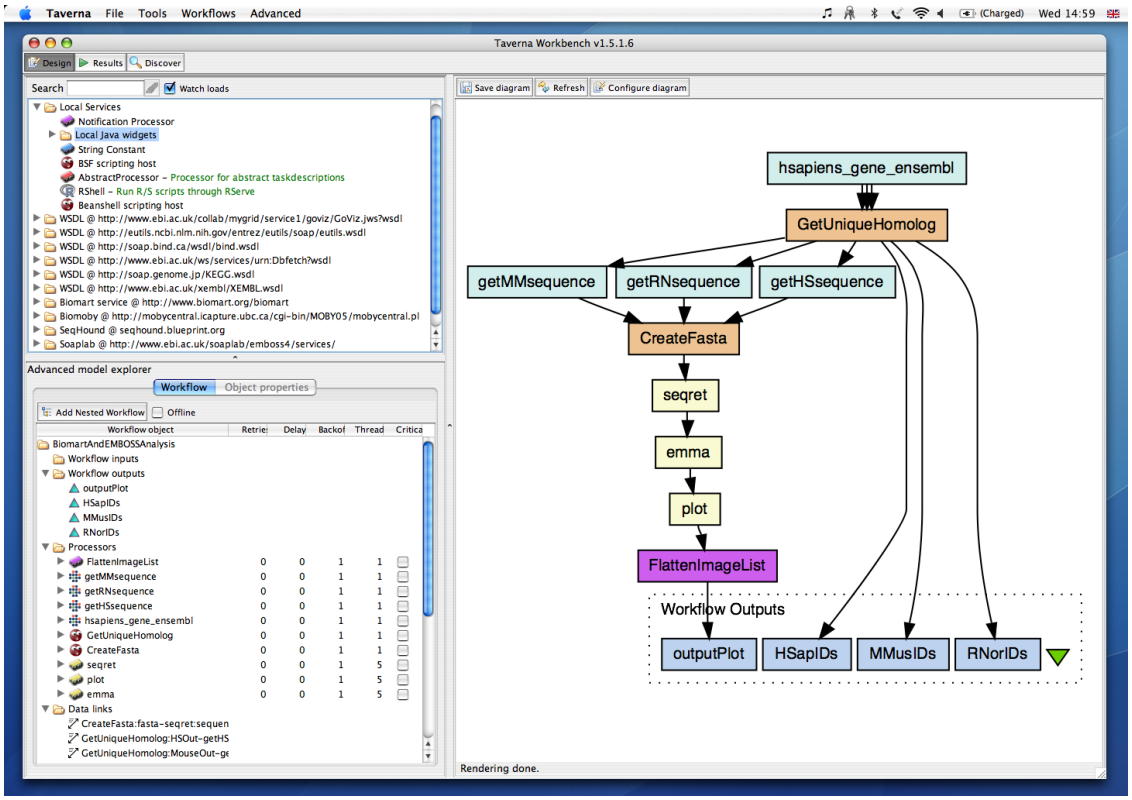

Figura 2.1: Interface gráfica do sistema de *workflow* científico Taverna. Extraído da referência [47].

Além da flexibilidade, pode-se citar como outras vantagens de se usar *workflow* em trabalhos científicos a divisão do trabalho e o controle avançado da execução. Por divisão do trabalho entende-se a execução de tarefas em máquinas remotas ou

até mesmo em *clusters*. O controle avançado da execução é o acompanhamento da computação da tarefa, com opções de visualização de variáveis, de pausa, resumo e parada da computação.

Um MWC é o projeto criado pelo usuário em que os componentes são organizados e o fluxo de processamento é definido. Outro exemplo de MWC é apresentado na Figura 2.2. Os retângulos representam as diversas tarefas a serem executadas e as setas definem as dependências entre elas. Os ícones ornamentando os retângulos ajudam a identificar as diferentes tarefas. As tarefas são identificadas também por nomes sugestivos de suas atividades. O texto descritivo é um dos artefatos possíveis no diagrama e tem papel de documentação.

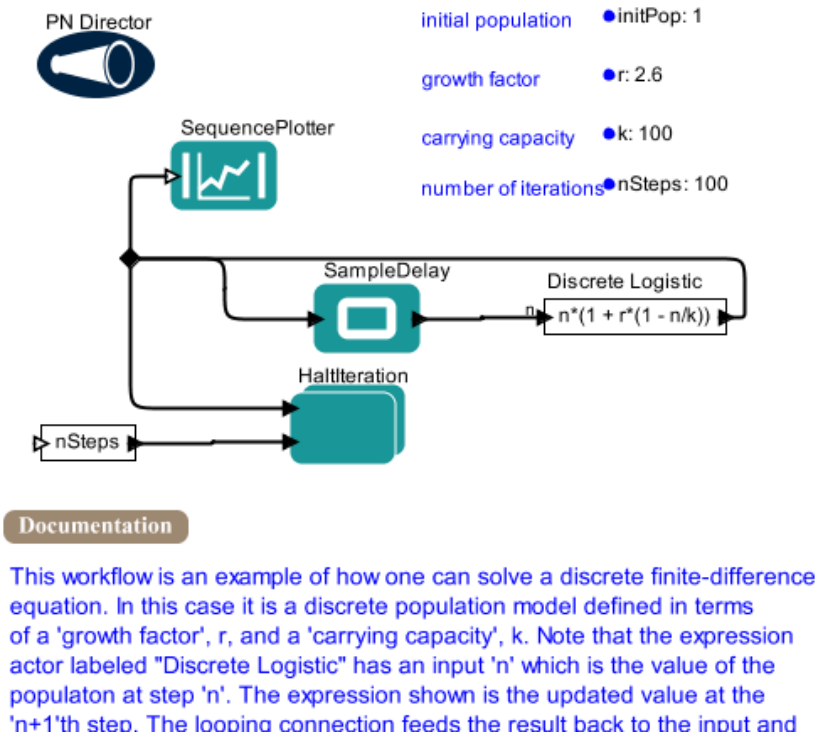

Figura 2.2: Exemplo de um modelo de *workflow* científico criado no Kepler. Extraído da referência [46].

Os usuários de um SWC são classificados de acordo com a especialidade e o uso na criação dos seus MWC:

usuário leigo é quem apenas executa os MWCs;

usuário intermediário é aquele que define os modelos e usa os recursos disponíveis em um SWC;

usuário avançado é capaz de criar novos componentes para seus MWCs que fornecem novas tarefas não encontradas nos SWCs.

2.1 A Taxonomia

A Taxonomia é conhecida como a ciência da identificação e classificação [62]. Assim, para facilitar o estudo dos SWCs é proposta nessa seção uma compilação dos diversos conceitos encontrados na literatura, em trabalhos que tratam do tema. Essa compilação é organizada sob forma de uma taxonomia.

Nesse sentido, os conceitos organizados nessa taxonomia se referem aos possíveis recursos encontrados em um SWC. Esses recursos englobam tanto os blocos de construção de um MWC como também as funcionalidades para sua manipulação.

A taxonomia criada neste trabalho foi desenvolvida a partir da leitura de trabalhos que apresentam taxonomias nesse domínio, estendendo-as com novos conceitos encontrados na análise das funcionalidades dos SWCs estudados: Kepler, Triana, Taverna e Vistrails.

A estrutura da taxonomia neste trabalho foi dividida inicialmente em três categorias: o Projeto, a Execução e a Proveniência. A categoria Projeto trata da montagem e organização de um MWC. A categoria Execução apresenta as funcionalidades do *workflow* em tempo de processamento. Por fim, a categoria Proveniência aborda as funcionalidades após execução de um MWC.

A Figura 2.3 mostra as categorias da taxonomia, e suas subcategorias, propostas por este trabalho.

2.2 A Categoria Projeto

O Projeto é uma etapa da construção de um MWC em que se especifica como suas tarefas são definidas e compostas. Tarefa é um procedimento utilizado para se solucionar um problema bem definido. Podem ser citados como exemplos de tarefa: o cálculo de uma raiz quadrada, a montagem de um gráfico, a computação de uma integral e a solução de uma equação.

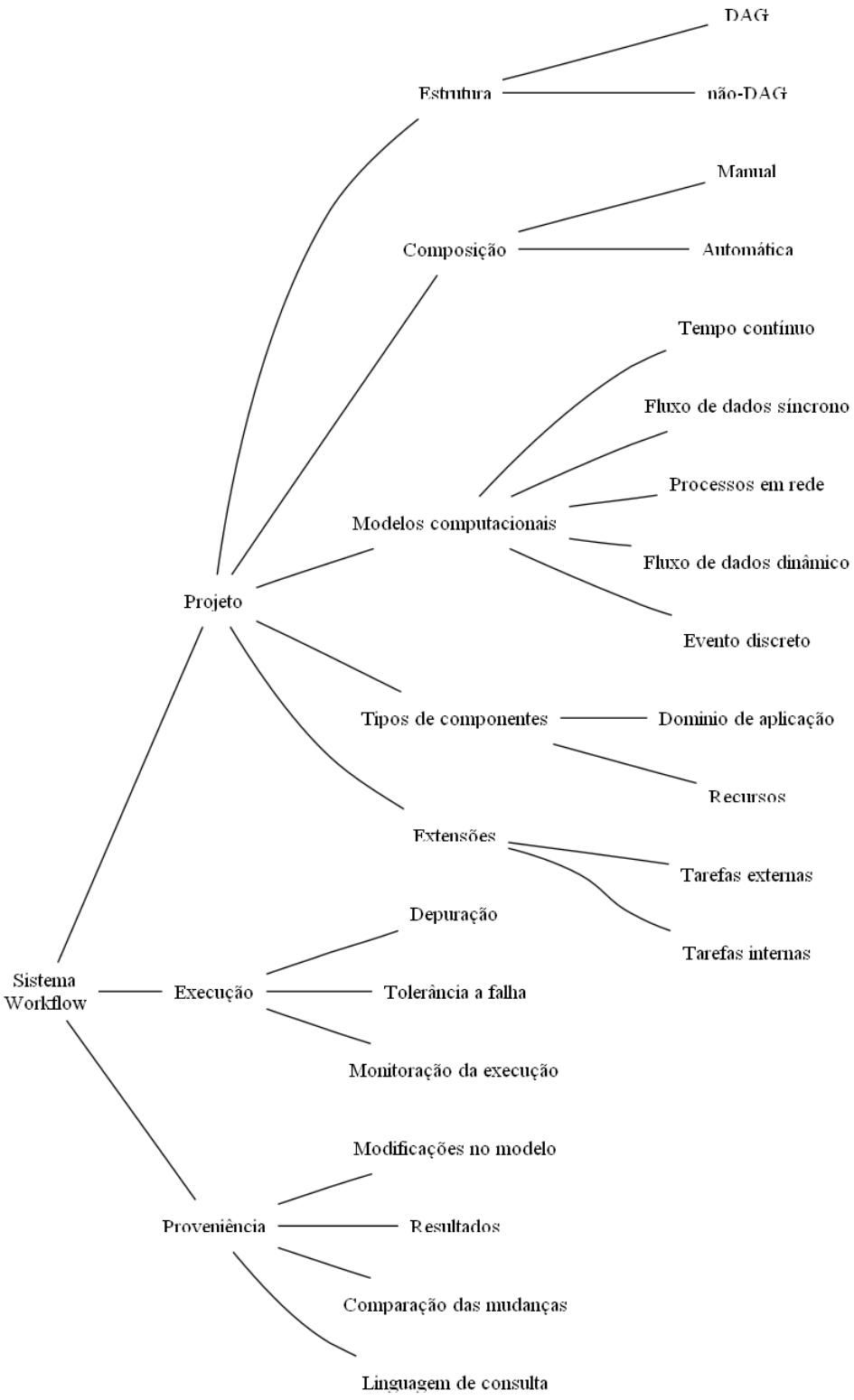

Figura 2.3: Árvore taxonômica dos conceitos de *workflow* científico (contribuição do autor).

Dentro do escopo da categoria Projeto de um *workflow* científico, encontram-se os seguintes conceitos: Estrutura (2.2.1), que identifica os tipos de organização da execução das tarefas; Composição (2.2.2), que aborda a interação do usuário na montagem do MWC; Modelos Computacionais (2.2.3), que descrevem como será o comportamento da execução de um MWC; Tipos de componentes (2.2.4), que agrega as tarefas disponíveis por padrão nos SWCs; e Extensões (2.2.5), que, por fim, apresenta as formas de adicionar novas tarefas aos MWCs. Esses conceitos são detalhados a seguir.

2.2.1 Estrutura

Um MWC é composto de várias tarefas conectadas de acordo com suas dependências, cuja estrutura indica a ordem de execução no tempo, de cada tarefa. Um MWC, geralmente, é organizado utilizando-se o modelo DAG (*Directed Acyclic Graph*) ou não-DAG. Um modelo DAG, na matemática, pode ser traduzido como um grafo dirigido sem ciclo [61], como ilustrado na Figura 2.4. Portanto, o modelo não-DAG é um grafo cíclico.

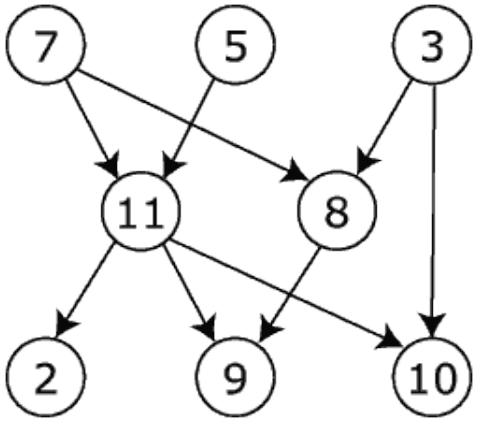

Figura 2.4: Exemplo de um grafo dirigido sem ciclo.

Um MWC baseado no modelo DAG pode conter estruturas do tipo Seqüencial, Paralela ou Seleção. A primeira define a execução em série das tarefas, ou seja, uma tarefa inicia seu processamento somente após a tarefa anterior ter finalizado seu trabalho. No tipo de estrutura Paralela, as tarefas executam suas atividades ao mesmo tempo. Por fim, o tipo de estrutura Seleção define a execução de uma tarefa somente quando a condição associada a essa tarefa for verdadeira.

Um MWC baseado no modelo não-DAG tem todos os tipos comentados no modelo DAG (seqüencial, paralelo e seleção) mais o tipo Laço. Esse se resume em um conjunto de tarefas que, em um bloco de iteração, pode se repetir várias vezes até uma condição ser satisfeita.

2.2.2 Composição

Os sistemas para composição de MWC são projetados para auxiliar o usuário a organizarem componentes no *workflow*. Um Componente é uma tarefa representada graficamente em um SWC. Os tipos de composição definidos na literatura são composição manual e composição automática.

Na Figura 2.5, pode-se visualizar um SWC com um MWC e seus componentes. Para a sua montagem, torna-se necessário, inicialmente, definir quais componentes serão necessários incluir no modelo, para, em momento posterior, serem escolhidas as ligações entre os componentes de acordo com seus tipos de dados de entrada e saída. Os dados de entrada normalmente ficam no lado esquerdo ou em cima do componente e os dados de saída na direita ou embaixo, essa disposição varia de um SWC para outro.

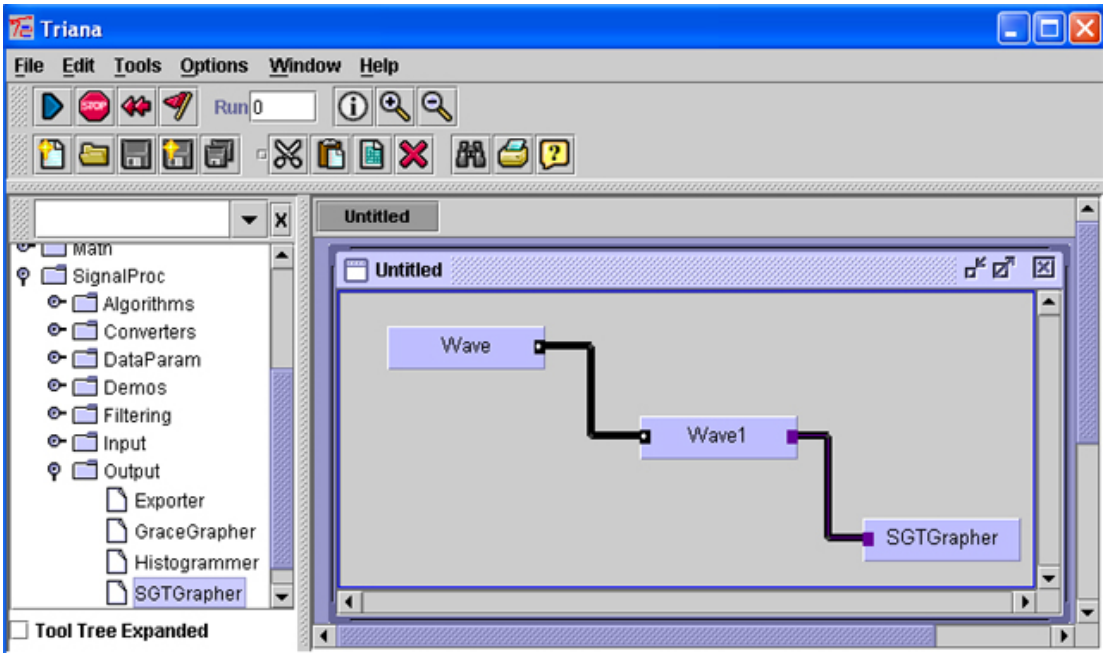

Figura 2.5: Paleta de componentes, componentes e interface gráfica do sistema Triana.

As ferramentas de composição manual geralmente são sistemas gráficos com representações intuitivas e podem ser utilizadas facilmente por usuários leigos.

Alguns SWCs possibilitam a importação de novos componentes a partir de repositórios na internet. Esses repositórios normalmente armazenam novas tarefas criadas pelos desenvolvedores do projeto ou, ainda, por outros usuários dos sistemas.

Outra variante da atividade de Composição é a composição automática, que consiste na geração de um MWC a partir de poucos requisitos, como dados esperados e parâmetros de entrada. Esse tipo de composição é recomendado para usuários que precisam criar uma grande quantidade de *workflows*. Em contra partida, há o desafio de encontrar os componentes corretos para comporem o MWC.

A composição automática é feita por um algoritmo que tem a função de selecionar e compor as tarefas num MWC. Essa automação pode ocorrer em diversos graus, variando desde a simples recomendação das tarefas até a composição de um MWC completo. As ferramentas baseadas em ontologia estão em fase de pesquisa para darem suporte à composição automática de MWC [16].

Outra interessante técnica de composição automática de *workflows* é disponibilizada pelo sistema Vistrails: por meio de informação, em alto nível de abstração, sobre os resultados semelhantes obtidos em outros *workflows* selecionados pelo usuário, é possível realizar a composição de um novo MWC por analogia [49].

2.2.3 Modelos computacionais

Em teoria da computação, um modelo computacional (*Model of Computing*) é a definição do conjunto de operações disponíveis para processamento de tarefas. O modelo computacional descreve como será o comportamento do sistema como um todo em função da combinação das suas operações elementares [52].

Os modelos computacionais podem ser agrupados em categorias, conforme as características principais encontradas em determinados domínios de aplicação. Por exemplo, determinadas aplicações possuem tarefas que ocorrem de forma contínua no tempo, como os circuitos analógicos. A modelagem de tais sistemas é normalmente feita através de sistemas de equações diferenciais ordinárias. Em outros sistemas, os eventos ocorrem de forma discreta, como a entrega de mensagens em redes de

comunicação de dados. Alguns desses casos têm como característica principal a noção de estados entre os quais o sistema pode alternar. Em sistemas com essa característica, a máquina de estados finitos é um conhecido modelo conceitual para sua modelagem.

Em outras ocasiões, é necessário modelar a possível ocorrência simultânea dos eventos de um sistema. Vários modelos conceituais têm sido propostos para modelar sistemas concorrentes, como por exemplo a linguagem formal CSP (*Communicating Sequential Processes*) [9].

Outro modelo importante é o Fluxo Síncrono de Execução de tarefas, encontrado em linguagens de programação. Esse modelo é, em muitos casos, suficiente para modelar as seqüências de atividades de um experimento científico, de particular interesse para o escopo desse trabalho.

Existe uma variedade de modelos computacionais que trabalham com concorrências e tempos de diferentes maneiras [7]. Esses modelos podem ser classificados como modelos computacionais determinantes, que conhecem todas as etapas da execução, e modelos computacionais estaticamente programados, que conhecem algumas etapas durante a execução.

A escolha de um modelo computacional apropriado para uma aplicação particular é muitas vezes difícil, por isso é essencial uma análise cuidadosa e uma seleção criteriosa antes da construção de fato no SWC.

Os SWCs que pretendam dar suporte a experimentos de diferentes domínios de aplicação devem possuir recursos que possibilitem a seleção do modelo computacional que será aplicado ao sistema que está sendo modelado.

O sistema Ptolemy, base do sistema Kepler, desenvolveu dezessete tipos de modelos computacionais diferentes. Entretanto, o Kepler incorporou em seu sistema um pequeno conjunto dos mais usados: Tempo contínuo, Fluxo de dados dinâmico, Evento discreto, Processos em rede e Fluxo de dados síncronos [46].

O estudo dos diferentes modelos computacionais e de como implementá-los não está entre os objetivos desse trabalho. Será feita, tão somente, a seguir, uma introdução sobre cinco dos principais modelos suportados nas ferramentas estudadas. Como se sabe, modelos computacionais podem ser combinados, gerando modelos

híbridos de comportamento mais complexos, discussão essa que também está fora do escopo desse trabalho.

Tempo contínuo (CT)

O modelo computacional tempo contínuo (CT - *continuous time*) foi implementado no Ptolemy por Jie Liu [42]. Nesse modelo, os componentes interagem via sinais em tempo contínuo e normalmente especificam relações algébricas ou diferenciais entre os dados de entrada e saída. O trabalho desse modelo computacional é encontrar um ponto fixo, ou seja, um conjunto de funções contínuas no tempo que satisfaça todas as relações.

Esse modelo é muito útil para modelagem de sistemas físicos com descrição de equações algébricas ou diferenciais lineares ou não lineares, como por exemplo, circuitos analógicos ou sistemas mecânicos.

A avaliação dos efeitos das mudanças dos parâmetros em MWCs que utilizam a modelagem CT é relacionada com o valor corrente ou avaliação das mudanças de outros parâmetros. Por exemplo, as mudanças na população de um predador e sua presa, sobre o tempo, (equação de Lotka-Volterra) pode ser calculada usando um *workflow* com o modelo computacional CT. A Figura 2.6 ilustra o exemplo citado.

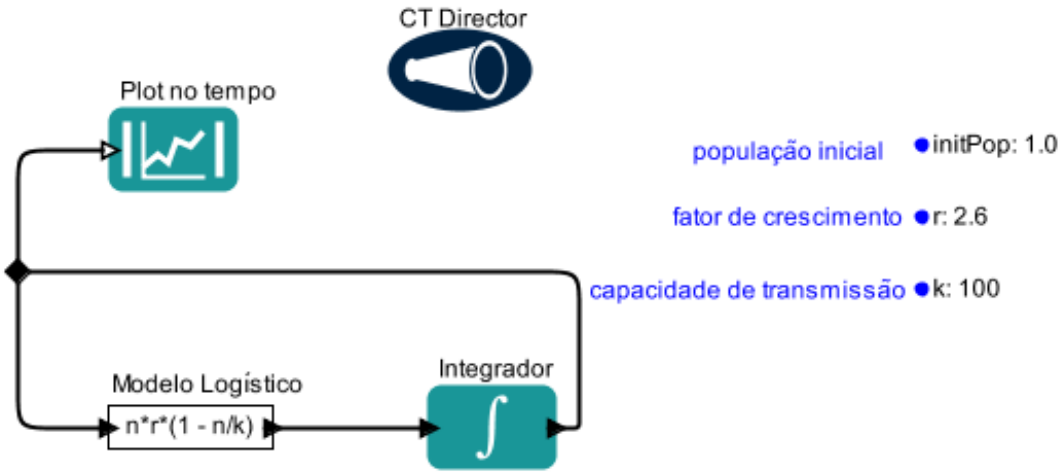

Figura 2.6: Modelo de *workflow* usando um modelo computacional tempo contínuo (CT) no sistema Kepler. Extraído da referência [46].

O modelo CT no sistema Kepler é projetado para operar em grupo com outros

modelos computacionais, tais como os eventos discretos (DE - *discrete events*), para obter uma modelagem de sinal mista, ou as máquinas de estado finito (FSM - *finite-state machine*), para criar modelos modais.

Fluxo de dados síncronos (SDF)

Codificado no PtolemyII por Steve Neuendorffer [7], o modelo computacional fluxo de dados síncronos (SDF - *Synchronous Dataflow*) serve à manipulação de computações regulares que operam em fluxo sequencial e capaz de determinar a ordem de execução das tarefas do MWC.

O modelo SDF é muito eficiente e não onera os recursos de sistema com dados extras, tendo em vista que, antes de iniciar a execução em si, uma pré-análise das execuções das tarefas é realizada. Entretanto, para alcançar a mencionada eficiência, certas condições deverão ser satisfeitas, como por exemplo, os valores dos dados consumidos e produzidos de cada componente devem ser constantes e declarados.

Todavia, MWCs que necessitam de execuções ou fluxos dinâmicos não podem usar esse tipo de modelo computacional. Além disso, o modelo SDF, a princípio, não reconhece a passagem do tempo e tarefas que dependem da noção do tempo não trabalham como esperado.

A Figura 2.7 apresenta um exemplo de MWC com o modelo computacional SDF, onde uma imagem é lida, rotacionada e apresentada para visualização. Esse exemplo é um MWC sequencial que necessita de um modelo computacional simples que assugure a computação de cada tarefa na ordem correta. Assim, o SDF assegurará que a imagem não será visualizada até que ela seja rotacionada, e, garante também, que a imagem não será processada até que seja carregada pela tarefa correspondente. Essa gerência deve-se ao fato que o SDF realiza uma pré-análise da execução do MWC antes de iniciar a computação da primeira tarefa.

Processos em rede (PN)

O modelo computacional processos em rede (PN - *Process Networks*) foi implementado no Ptolemy por Mudit Goel [43]. Nesse modelo, as tarefas são computadas em paralelo, geralmente implementadas como Java *threads*[27], e se comunicam pelo

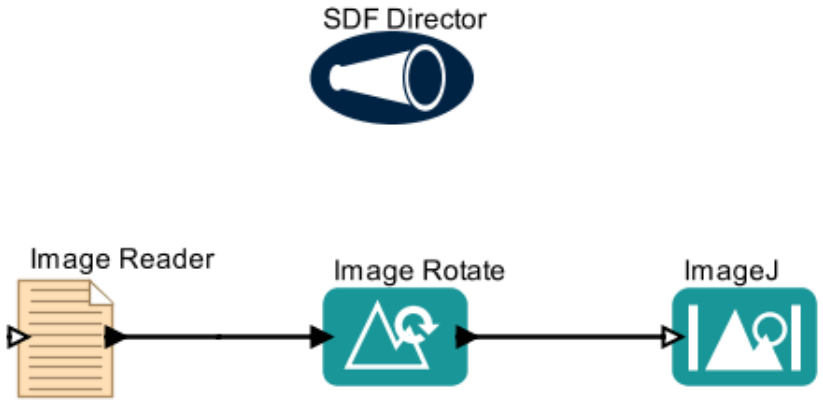

Figura 2.7: Modelo de *workflow* usando um modelo computacional de fluxo de dados síncrono (SDF) no sistema Kepler. Extraído da referência [46].

envio de mensagens através de canais que podem armazenar essas mensagens.

O estilo de comunicação usada é comumente conhecido como assíncrona, na qual o emissor da mensagem não precisa esperar que o receptor esteja pronto para recebê-la. Existem inúmeras variantes dessa técnica, mas o modelo PN utiliza uma técnica que assegura a computação, conhecida como *Kahn process networks* [22].

O modelo em análise, assim como o modelo computacional fluxo de dados síncrono (SDF), não tem noção de tempo. A diferença entre os dois modelos está na maneira que são executados, uma vez que o modelo PN não realiza uma pré-análise da execução das tarefas. Um MWC que utilize PN é guiado pelos dados disponíveis, ou seja, os dados são criados nas saídas das tarefas a qualquer momento, e estão disponíveis para as entradas da próxima tarefa para que possam computá-los. Os dados de saída são passados através dos canais de comunicação, onde ficarão armazenadas até que a próxima tarefa colete todos os dados de entradas necessários e possa iniciar sua execução.

Workflows que utilizem o modelo PN são eficazes, pois têm poucas restrições. Por outro lado, podem ser ineficazes porque o algoritmo de coordenação desse modelo deve observar as tarefas que tem dados suficientes para iniciar a execução.

Apesar das vantagens apresentadas acima, as tarefas do *workflow* utilizando o

modelo PN podem causar resultados inesperados. Por exemplo, um *workflow* pode recusar terminar a execução automaticamente porque sempre há dados sendo criados e disponíveis para as tarefas executarem. Outra falha pode ocorrer se uma tarefa gerar dados de saída muito mais rápido que outra consiga ler, assim, a área de armazenamento pode gerar estouro de memória e causar uma falha na execução do *workflow*.

Fluxo de dados dinâmico (DDF)

O modelo computacional fluxo de dados dinâmicos (DDF - *Dynamic Data Flow*) foi implementado no Ptolemy por Gang Zhou [23]. Este modelo é a união dos modelos fluxo de dados síncronos (SDF) e fluxo de dados booleano (BDF - *Boolean dataflow* [7]).

No modelo SDF, uma tarefa consome e produz um número fixo de dados por execução. Essa informação estática torna possível a avaliação da execução antes de executar de fato. No modelo em análise não há esse controle estático e, assim sendo, uma tarefa pode mudar a quantidade de informação produzida ou consumida a cada execução.

Os componentes que caracterizam o modelo DDF são *Select* e *Switch*, que consomem ou produzem dados nos diferentes canais de comunicação baseados nos dados recebidos.

Esse modelo é indicado para *workflows* que utilizem laços, ramificações ou outras estruturas de controle. Se o MWC tiver como requisito o processamento em paralelo, o modelo PN poderá ser usado. Em geral, o modelo DDF é uma ótima escolha para se gerenciar *workflows* que contenham o tipo de estrutura *if-then-else*, que a interação dependa dos dados ou que exista recursividade.

A Figura 2.8 apresenta um MWC que utiliza o componente *BooleanSwitch* para direcionar a entrada para o canal de saída *If* ou o canal *Else*, que dependerá do valor passado pelo canal de controle. Pelo fato da saída do componente *BooleanSwitch* não ser constante, esse *workflow* não pode ser executado por um modelo SDF, que necessita que a produção e consumo de seus dados sejam conhecidos.

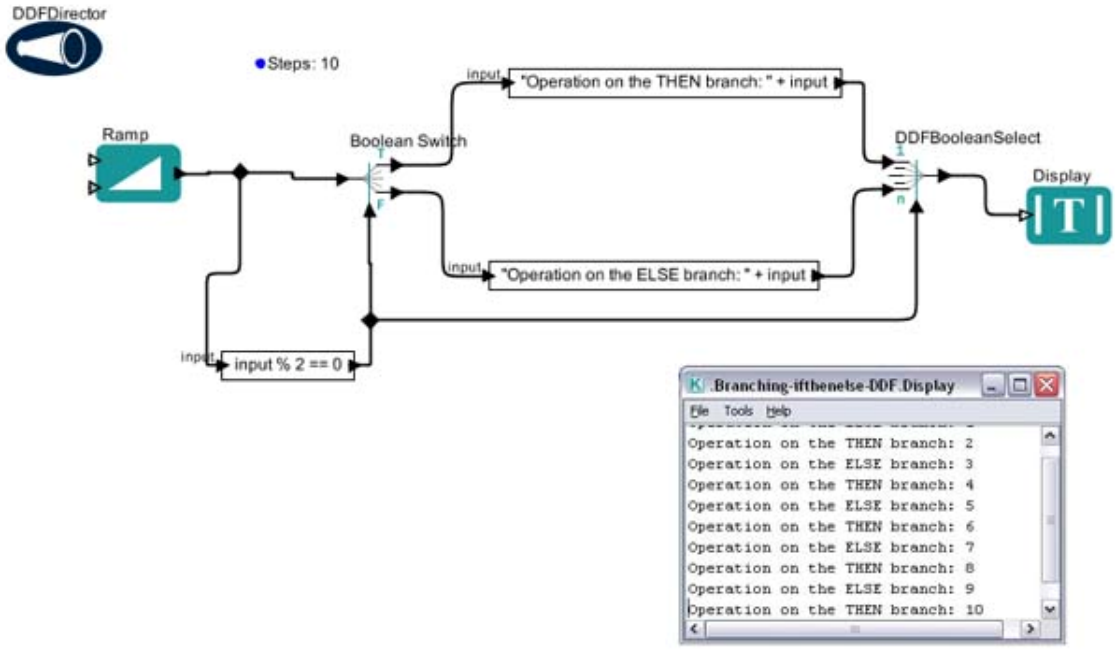

Figura 2.8: Uso do modelo DDF com um *workflow* que usa o tipo de estrutura *if-then-else*. Extraído da referência [46]

Evento discreto (DE)

O modelo computacional evento discreto (DE - *Discrete Event*) foi codificado no Ptolemy por Gang Zhou [40]. Nesse modelo, os processos se comunicam via seqüência de eventos, que é um valor e um ponto, ambos estabelecidos no tempo, junto com a linha de tempo real. Uma tarefa pode ser um processo que reage a eventos ou funções que executam quando novos eventos são fornecidos.

Esse modelo é comumente utilizado para a especificação de *hardware* digital, para a simulação de sistemas de telecomunicações e para um grande número de simulação de ambientes, de linguagens de descrição de *hardware* e de simulação de linguagens. É bastante empregado para modelagem de sistemas orientados no tempo, tais como sistemas de enfileiramento ou comunicação em redes.

Um problema clássico que pode ser modelado pelo DE é o da estação de ônibus e o passageiro, em que ônibus e passageiros chegam na estação aleatoriamente ou em taxas fixas e o diretor de trânsito deseja calcular, ou minimizar, o número de vezes que os passageiros ficam esperando.

O modelo DE implementa um simulador de evento discreto bastante sofisticado.

Os simuladores DE normalmente precisam manter uma fila global de eventos pendentes ordenados por tempo, conhecida como fila de prioridades. Ocorre que essa regra pode custar muitos recursos computacionais, pois precisa inserir um novo evento na ordem correta na fila.

Além disso, o modelo DE dá uma semântica determinística para eventos simultâneos, ao contrário da maioria das competições dos eventos discretos dos simuladores. Isso significa que para qualquer dois eventos com o mesmo tempo, a ordem na qual eles serão processados será deduzida da estrutura do modelo. Isso é feito a partir da análise da estrutura gráfica do MWC para os dados precedentes.

Escolha do modelo computacional

Todo MWC requer um modelo computacional, que tem uma maneira única de instruir as tarefas de um *workflow*.

A escolha do modelo computacional a ser utilizado deve ser feita durante os estágios iniciais da construção do *workflow*. À medida que o usuário descreve os passos do *workflow* e pensa sobre os tipos de tarefas que serão executadas, é preciso ter as seguintes perguntas em mente: O *workflow* depende do tempo? Ele executa uma transformação de dados simples com taxas de produção e consumo constantes? O modelo *workflow* é descrito por uma equação diferencial? As respostas a essas questões geralmente indicarão qual é o melhor modelo computacional que se deve usar [[46]].

A seguir, será discutido quais modelos computacionais devem ser usados, de acordo com as respostas das perguntas apresentadas acima.

Questão 1: O *workflow* depende explicitamente do tempo?

Para diversas tarefas computadas num *workflow*, a passagem do tempo é indiferente. Um *workflow* que aplica uma formatação a um arquivo de dados estático em outro tipo precisa ser capaz de ler o formato de entrada e saber como traduzi-lo, mas não necessita saber quantos segundos são necessários entre o tempo de início e término da execução. Igualmente, para um *workflow* que verifica uma série de moléculas e compara (compara ou modela, etc.) suas estruturas com outra série, é dispensável o cômputo do transcurso do tempo. O modelo computacional que esses

workflows usam deve conhecer como ordenar os eventos, ou seja, em que momento cada tarefa do *workflow* deve ser executado. Entretanto, não é necessário que as ações das tarefas sejam executadas em tempos específicos.

Por outro lado, alguns *workflows* exigem o conhecimento da noção do tempo. Num *workflow* que descreva o crescimento da população com recursos limitados, onde a população é uma função do tempo e a taxa da população muda, é preciso usar o tempo para fazer os cálculos de previsão do crescimento populacional. Do mesmo modo, um *workflow* que modela eventos que ocorrem em tempos discretos também requer que se tenha noção do tempo. Note-se, entretanto, que o tempo do modelo e o tempo real podem ser diferentes. Por exemplo, uma análise pode gastar apenas alguns segundos no tempo real para ser feita, enquanto o tempo do modelo avançou por várias horas.

Alguns modelos computacionais são melhor empregados em *workflows* com dependência de tempo e outros para *workflows* com independência de tempo. Geralmente, se um *workflow* necessita ter noção do tempo, usa-se os modelos CT ou DE. Se o *workflow* não requer que se tenha noção do tempo, utiliza-se os modelos SDF, PN ou DDF.

Questão 2: O *workflow* executa uma transformação de dados simples com taxas de produção e consumo constantes?

Inicialmente, destaca-se que, se o modelo *workflow* exige o conhecimento do tempo empregado, não se faz necessário responder a essa questão de número dois. Deve-se, pois, passar para a questão 3.

Uma transformação simples de dados não faz uso de uma lógica complexa, de uma computação distribuída ou de múltiplos processos executando em paralelo. Entre os exemplos de transformação de dados simples estão a conversão de tipo de dados: uma série de itens para um vetor, a tradução do formato de um arquivo para outro: arquivo XML (*Extensible Markup Language*) para arquivo HTML (*HyperText Markup Language*); o cálculo da média de uma série de valores; e a leitura de um arquivo de dados e apresentação de uma linha ou valor específico.

Uma taxa constante significa que todas as tarefas de um *workflow* consumirão e produzirão um número pré-determinado de dados a cada vez que o *workflow* iterage. Apesar de um vetor ou matriz consistir em múltiplos itens, ele é reconhecido como

um único tipo de dado e é passado da saída de uma tarefa para a entrada de outra via canal de comunicação.

O modelo computacional recomendado para *workflows* que façam transformações de dados simples e que tenham taxas constantes de produção e consumo é o SDF. Se o *workflow* envolve uma lógica complexa, computação distribuída ou produção/consumo de dados inconsistentes, é recomendado o uso dos modelos DDF ou PN.

Questão 3: O modelo *workflow* é descrito por uma equação diferencial?

Equações diferenciais geralmente são usadas por *workflows* que descrevem sistemas dinâmicos, sistemas que dependem de um parâmetro que varia no tempo continuamente, tais como crescimento da população de predadores e suas presas, no tempo, ou *workflows* que são usados para executarem integrações numéricas. Esses MWCs deverão usar o modelo computacional CT, que foi desenvolvido para trabalhar com equações diferenciais ordinárias.

Os *workflows* orientados por tempo e que não estão envolvidos com equações diferenciais deverão usar o modelo computacional DE para executar os eventos nos tempos especificados ou para simulação de programação de eventos.

A Figura 2.9 apresenta um guia de referência rápido muito útil para se escolher o modelo computacional.

2.2.4 Tipos de componentes

Os SWCs trazem por padrão uma vasta variedade de componentes que facilitam a criação dos MWCs, pois o usuário não precisa codificar tarefas para trabalhos básicos ou avançados. Esses componentes vão desde operações matemáticas simples até processamentos de imagens, processamentos de sons, entre outros.

Os componentes foram classificados em dois grupos ortogonais: Domínio de aplicação e Recursos. Comumente, os componentes de Domínio de aplicação são apresentados como a dimensão horizontal da ortogonalidade, enquanto os componentes de Recurso, nesse caso, são a componente vertical. A subcategoria Domínio de

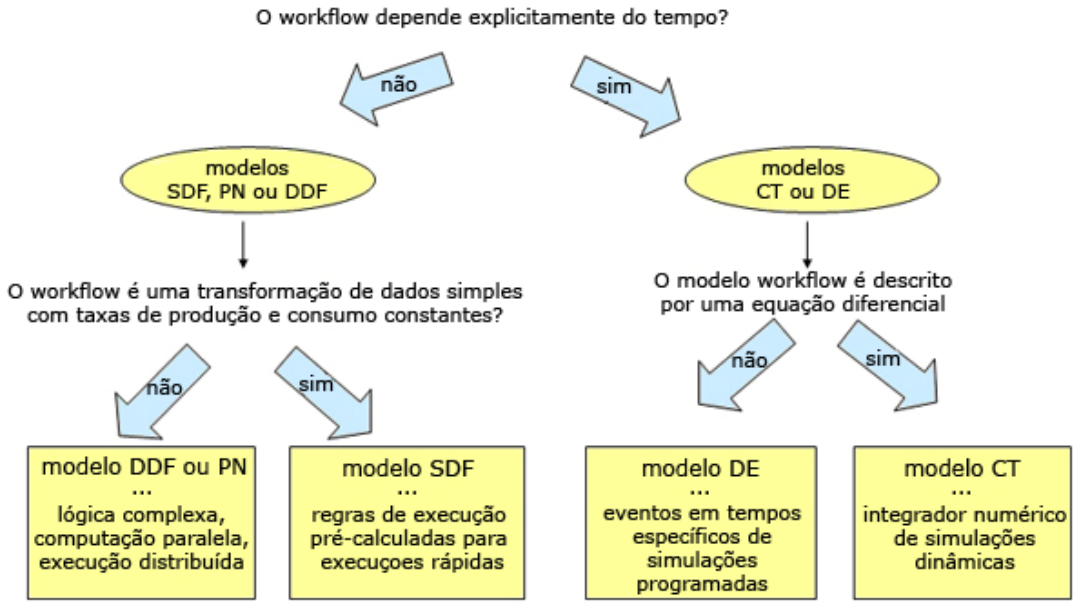

Figura 2.9: Guia de referência rápido para escolha do modelo computacional. Extraído da referência [46]

aplicação agrega componentes que executam tarefas intrínsecas ao domínio do experimento que será executado. Os componentes dessa subcategoria foram agrupados pelas áreas de conhecimento nas quais os componentes se aplicam. A categoria Recursos agrega os componentes que implementam as funcionalidades da ferramenta de *workflow*, que se aplicam a MWCs de qualquer área de conhecimento. A Figura 2.10 apresenta um exemplo de classificação de componentes.

A Figura 2.10 mostra a paleta de componentes do SWC Kepler, que sugere uma classificação dos tipos de componentes. A árvore mostra itens e subitens que agrupam componentes. Por exemplo *Components* e *Disciplines* equivalem às subcategorias Recursos e Domínio de Aplicação da taxonomia, respectivamente.

Domínio de aplicação

Os SWCs disponibilizam para os usuários componentes para computação de dados de áreas de domínio como computação, matemática, estatística, biologia, química, etc. Dentre os vários grupos disponíveis podemos destacar:

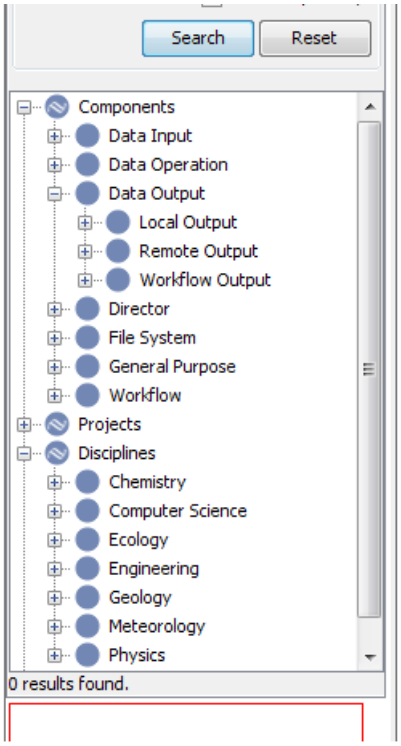

Figura 2.10: Paleta de componentes do sistema Kepler agrupados por funcionalidade ou domínio de aplicação.

Computação este grupo traz componentes específicos da computação, como execução de tarefas de conversão de tipo de dado, leitura e escrita de arquivos, etc;

Matemática este grupo contém componentes desde os básicos como adição, subtração até cálculo de integrais e derivadas;

Estatística os componentes estatísticos fornecem tarefas que computam dados de distribuições, análises temporais, desenho de gráficos, etc. Alguns SWCs integram-se com o R®, que é uma linguagem e ambiente para cálculos e gráficos estatísticos;

Biologia este grupo contém componentes específicos de várias áreas da biologia, como por exemplo, a genética;

Química os componentes deste grupo disponibilizam a computação de dados químicos, como computação de experimentos GAMESS;

Processamento de imagens este grupo fornece componentes para leitura e escrita de imagens, manipulação tais como rotação e redimensionamento, efeitos,

etc;

Processamento de sinais os componentes deste grupo têm funções como filtros de freqüência e de tempo, função FFT, função *shift*, conversores entre outros;

Processamento de som este grupo contém componentes para leitura e escrita de arquivos de som, processamento de conversões como mono para estéreo, distorções *Fuzz*, etc.

Além dos domínios citados acima há exemplos de estudos nas áreas de conhecimento de geologia, de ecologia e de oceanografia [29].

Recursos

Os componentes classificados como recursos fornecem tarefas que podem trabalhar em conjunto com qualquer área do conhecimento. Essa subcategoria agrega os componentes que implementam as características funcionais de SWC descritas na taxonomia, como, por exemplo, estrutura de controle da execução e execução de tarefas externas.

Essa subcategoria agrega, ainda, componentes para algumas funcionalidades não contempladas na taxonomia, quais sejam:

Repositório de recursos alguns SWCs permitem acessar e buscar componentes e MWCs disponíveis em servidores que mantem uma grande quantidade desses recursos disponíveis para uso em experimentos científicos;

Documentação do *workflow* os SWCs disponibilizam componentes para acrescentar anotações ao MWC. Esses componentes não executam uma tarefa, apenas estão no MWC para fins de documentação;

Documentação do componente em alguns SWCs, cada componente disponível para uso tem uma documentação que explica qual é a tarefa a ser realizada e quais são os parâmetros de entrada e saída necessários para a computação;

Encapsulamento este permite que você agrupe um conjunto de tarefas e represente graficamente como um único componente. Entre as vantagens está a

organização visual para melhorar a leitura do modelo. Outra vantagem que merece destaque é a reutilização, onde cria-se um *workflow*, transforma-o em um componente e disponibiliza-o para ser reutilizado em outros *workflows*. Essa funcionalidade é muito útil para MWCs complexos e extensos, possibilitando a segmentação da modelagem em níveis de abstração. Um exemplo desse tipo de funcionalidade pode ser vista na Figura 2.11.

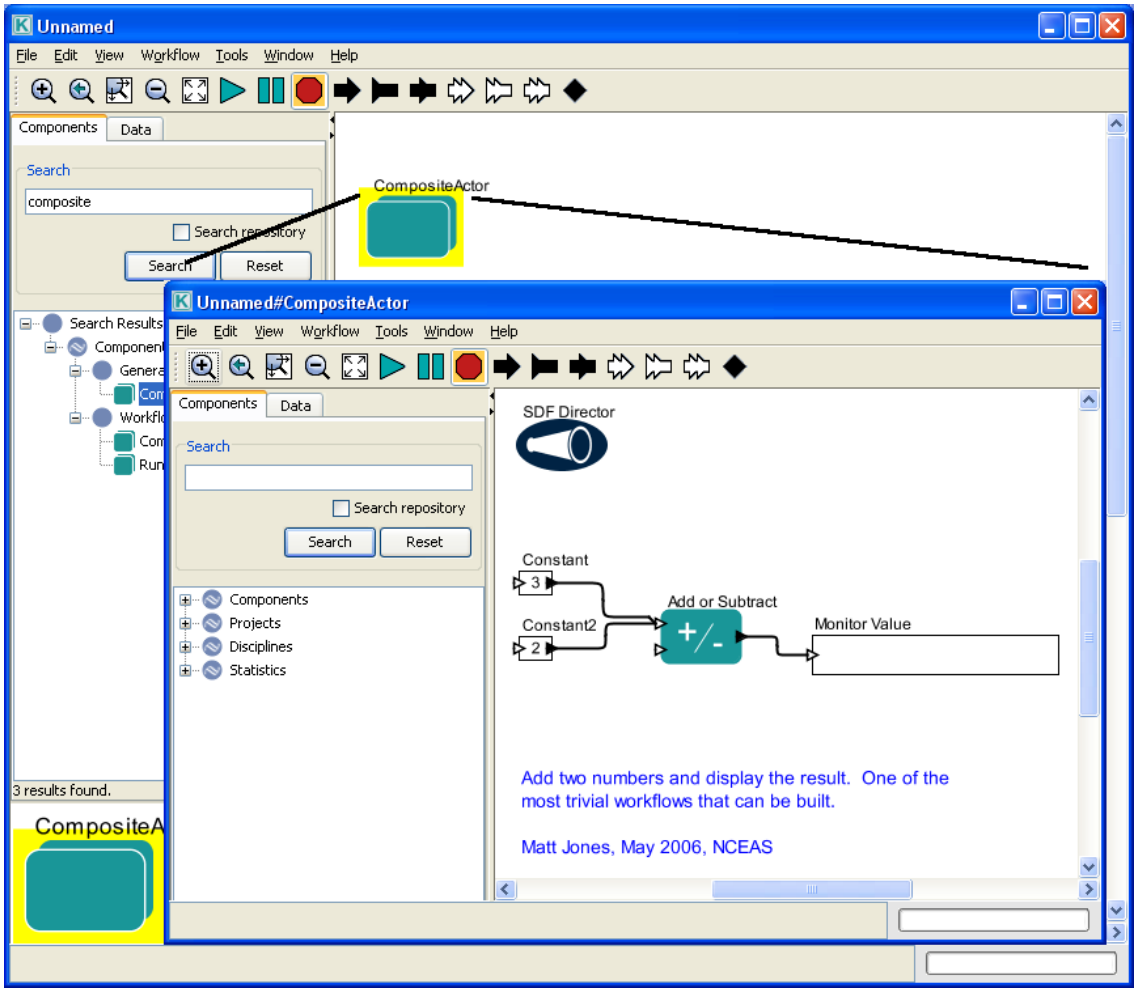

Figura 2.11: Encapsulamento de um grupo de componentes em um único componente. Modelo de *workflow* criado no Kepler. Extraído da referência [46].

A Figura 2.11 apresenta um exemplo de encapsulamento, onde um MWC simples, que realiza o somatório de dois números e apresenta o resultado, é agrupado em um único componente e utilizado pelo MWC mais complexo.

2.2.5 Extensões

Além dos componentes já existentes nos SWC, a subcategoria Extensões agrega componentes que permitem adicionar novas tarefas que executam trabalhos específicos do usuário, ainda não disponíveis em nenhum componente do sistema. As duas subcategorias identificadas foram a interação com aplicativos externos ao sistema e a implementação de novos componentes, denominadas neste trabalho como Tarefas externas e Tarefas internas, respectivamente.

Os componentes criados podem ser incorporados ao SWC para que os pesquisadores na área de modelagem possam facilmente disponibilizar seus trabalhos para o restante do grupo, ou até mesmo reutilizá-los em outros MWC. Alguns SWC fornecem maneiras de disponibilização através de repositórios na internet, que facilita também a busca e utilização por outros usuários interessados.

Uma dificuldade encontrada nesse momento, da construção de um novo componente, é saber quando dividir uma tarefa maior em várias tarefas menores. A vantagem da divisão está na possibilidade de reutilização, mas por outro lado, essa divisão pode comprometer o desempenho da execução do MWC.

Tarefas externas

Os SWCs permitem adicionar tarefas externas ao sistema principal de maneira facilitada, através de mecanismos de integração que dispensam o conhecimento do código fonte do SWC.

Em todas as ferramentas estudadas são encontradas maneiras de adicionar componentes que automatizam e facilitam o uso de aplicações externas, serviços web ou ambientes Grid.

Entre as aplicações encontradas que são suportadas pelos SWCs estão a ferramenta Matlab®, a ferramenta R®, programas de linha de comando do sistema operacional, entre outros. Essas extensões facilitam o reaproveitamento de tarefas criadas em outros *software*s que não valeriam a pena, num primeiro momento, serem incorporados ao SWC.

Além destas, os SWC também fornecem formas de utilização de tarefas disponí-

veis por meio de serviços web. Para computações que necessitam de alto desempenho há componentes para utilização de Grid.

Todavia, as integrações disponíveis nas ferramentas estudadas têm limitação quanto à linguagem de programação e formas de interação. Alguns exemplos de formas de interaçao e linguagens de programação são apresentados no Capítulo 3 (Suporte Tecnológico), nas Seções 3.2 (*Script*s), 3.3 (Serviço web) e 3.5 (Interfaces com Grids), especificamente.

Este trabalho faz uso de uma Tarefa Externa por meio de aplicação externa, utilizando Matlab® e interagindo com serviço web. A discussão detalhada de como esses componentes foram usados neste trabalho é feita nas Seções 6.2.1 e 6.2.2.

Tarefas internas

As documentações dos SWCs sugerem a criação de extensões via alteração do código fonte. A intenção é criar novos componentes, com as tarefas específicas do domínio do usuário, que ainda não são encontradas nos sistemas.

Saliente-se que esta opção de extensão é a de mais difícil implementação, pois pode requerer conhecimento do código fonte do SWC escolhido pelo usuário. Embora, para os profissionais da área de computação, alguns recursos da engenharia de *software* possam amenizar as complicações desse tipo de extensão, isso não é válido para profissionais de outras áreas. O domínio da linguagem de programação em que o sistema é implementado também é um fator de impacto.

Caso esse tipo de extensão esteja disponível em um SWC, ela torna a ferramenta em um *framework*[1] para novos desenvolvimentos na área de experimentos *in silico*. O uso desse *framework* desempenha importante papel não só no aspecto de eficiência do desenvolvimento, mas também no aumento do reuso dos trabalhos desenvolvidos. A ferramenta Kepler, por exemplo, permite disponibilizar os novos componentes desenvolvidos na paleta de seu sistema para usuários locais ou disponibilizá-los na internet via repositório de componentes.

Através do estudo de caso desenvolvido nesse trabalho ilustra-se o processo de

[1] *Framework*: estrutura de suporte definida para que um outro projeto de *software* possa ser organizado e desenvolvido.

extensão, via programação de componentes, no SWC Kepler. Baseado nesse estudo será realizada uma avaliação do grau de dificuldade de execução dessa tarefa. Para conhecer detalhes sobre o uso de Tarefas Internas neste trabalho, veja a Seção 6.2.3.

2.3 A Categoria Execução

A execução de um MWC se dá a partir do momento em que o usuário está com seu modelo composto no SWC. Nesta etapa o usuário inicia o processo de computação dos dados e obtém o resultado gerado pelo seu MWC.

Nos SWCs estudados, há várias funcionalidades que tornam o processo de execução mais fácil de acompanhar, a saber: informar a situação da computação das tarefas, informar a progressão por tarefa e por modelo, informar quais tarefas foram finalizadas com sucesso ou falha e enviar mensagens de notificação para email ou celular.

Existem, ainda, funcionalidades de depuração, como em processos de desenvolvimento de *software*s, que foram incorporadas aos SWCs para facilitar a resolução de problemas.

Além dessas, há funcionalidades para implementação de tolerância à falha, que tomam decisões automaticamente para correção de eventuais problemas de acordo com parâmetros definidos pelo usuário.

A monitoração da execução também é muito útil para o usuário leigo, que deseja acompanhar e conhecer quando determinadas situações ocorrem durante a computação.

2.3.1 Depuração

A etapa de depuração, também conhecida como *debugging*, é o processo de encontrar falhas em um aplicativo de *software* ou em *hardware* [67]. Uma falha (*bug*) é um erro no funcionamento de um programa de computador que impossibilita a execução de uma ação ou a utilização do mesmo [63].

Como em toda criação de código de computador, os MWCs também podem ser

criados com falhas. Assim, a depuração em um SWC é uma funcionalidade muito importante, pois facilita a descoberta dos problemas.

As funcionalidades disponíveis para depuração são pausa, retomar e parada, que interagem diretamente com o fluxo de execução do *workflow*; e componentes para acompanhamento dos valores que uma variável recebe durante a computação.

Outra funcionalidade de depuração é o acompanhamento da computação de um componente através de textos, que registram o que este está fazendo. Esses textos, adicionados no código fonte do componente pelo usuário que o desenvolveu, são úteis ao ajudar na identificação de possíveis erros da tarefa ou do próprio MWC. A Figura 2.12 mostra um exemplo dessa funcionalidade.

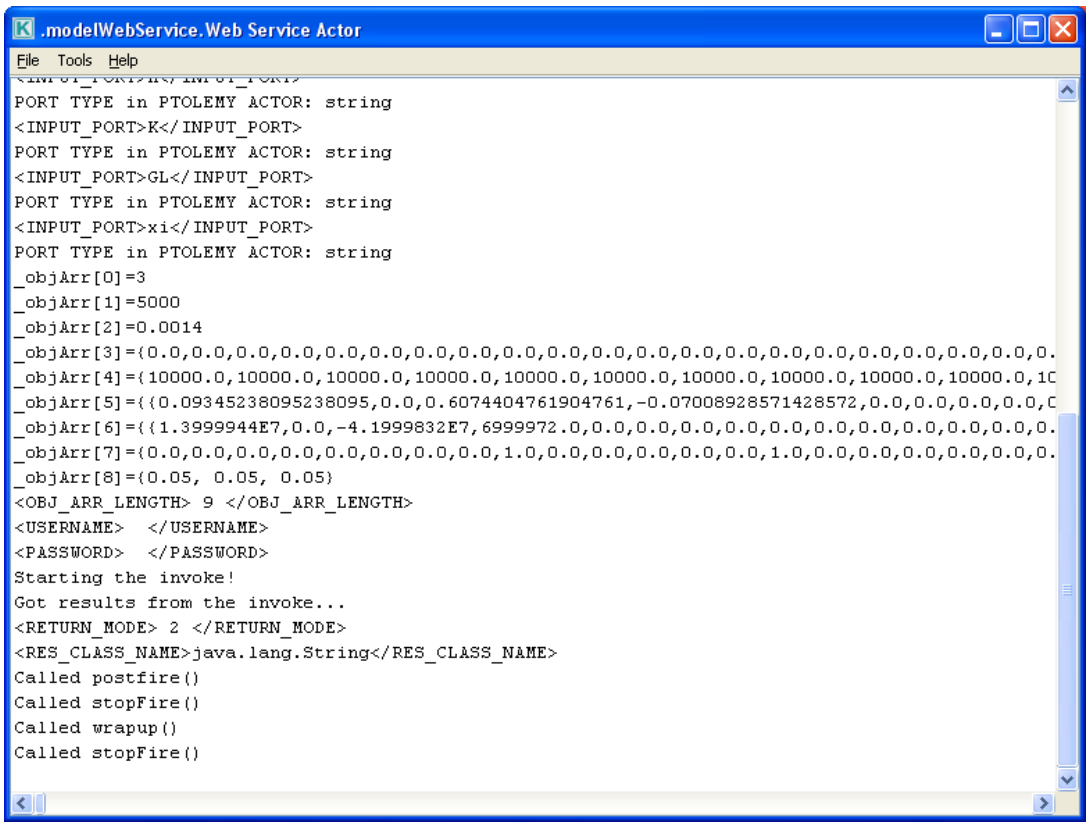

Figura 2.12: Depuração por meio de textos com informações sobre a computação de um componente *Web Service*. Extraído do sistema Kepler.

Outra funcionalidade relacionada a depuração é a verificação dos tipos de dados transmitidos pelos componenentes através dos canais de comunicação, que apresenta uma alerta quando encontra divergência nesses tipos.

2.3.2 Tolerância a falha

Pode ocorrer na execução de alguma tarefa uma falha, ou o SWC não conseguir se comunicar com um componente. Em situações normais, a execução do *workflow* é abortada e perde-se o trabalho computado até o momento.

Para auxiliar no controle dessas falhas, foram incorporadas aos SWCs algumas opções de tolerância à falha na execução dos MWCs, a fim de minimizar o problema ou automatizar as escolhas para acelerar o processo.

Nesse sentido, os controles de falha podem agir sobre uma única tarefa, um conjunto de tarefas ou sobre o MWC como um todo.

Um controle de falha de repetição permite ao usuário escolher quantas vezes uma tarefa ou *workflow* irá repetir a computação; quanto tempo irá aguardar até a computação ser reiniciada após uma falha; e se existirá um tempo adicionado ao tempo de espera a cada repetição.

Controles de tarefas alternativas são úteis no caso do SWC não conseguir se comunicar com uma tarefa, sendo a alternativa, assim, acionada e a execução prossegue naturalmente.

Tolerância a falha é muito importante em qualquer ambiente, principalmente utilizando-se um grande número de máquinas e processos.

2.3.3 Monitoração da execução

A monitoração é um recurso de interface gráfica que facilita o acompanhamento da execução principalmente pelo usuário leigo. O usuário pode realizar configurações para acompanhar quantas vezes um componente é usado, a progressão da execução ou a situação de um componente.

O objetivo é diminuir a complexidade do fluxo de execução das tarefas através de elementos visuais chamados ícones. Os ícones de progressão transmitem para o usuário quanto já foi computado e quanto ainda falta. Os ícones de semáforo permitem acompanhar a situação dos componentes através de cores associadas a cada situação. Por fim, os ícones contadores permitem acompanhar quantas vezes

um componente recebeu ou retornou alguma informação.

Um exemplo de ícones de monitoração pode ser visto na Figura 2.13.

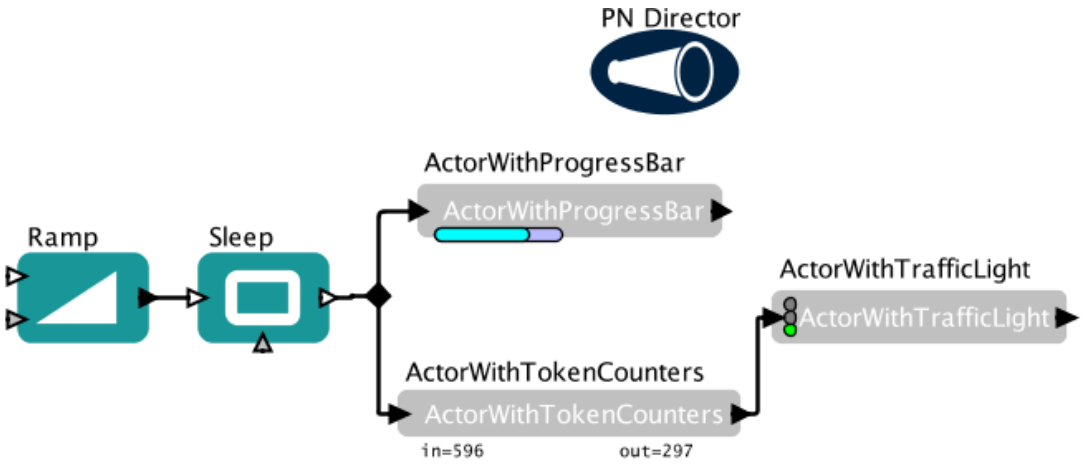

Figura 2.13: Ícones de progressão, de semáforo e de contador que facilitam o acompanhamento da execução de um modelo de *workflow* no sistema Kepler. Extraído da referência [17].

2.4 A Categoria Proveniência

O termo original, do inglês, para essa categoria, é *provenance*, que pode ser traduzido como rastreabilidade ou controle de versões. A rastreabilidade, de acordo com a norma NBR ISO8402/1994, é a capacidade de recuperação do histórico de uma entidade por meio de identificações registradas. O controle de versão tem como função a gerência das diferentes revisões (histórico) no desenvolvimento de um documento qualquer [71].

O conceito de proveniência, entretanto, engloba as funcionalidades dos dois termos citados, quais sejam, rastreabilidade e controle de versões, além de adicionar outras funcionalidades que serão comentadas durante esta seção. Portanto, o termo proveniência será utilizado para melhor destacar todas as suas funcionalidades.

A proveniência nos SWCs fornece um local de armazenamento da informação original e de suas derivações na execução de um MWC. Ela é muito útil na recriação de resultados antigos e para prover um processo de validação que apresenta os passos do resultado quando estes foram gerados. Essa funcionalidade também auxilia o

usuário a identificar como uma execução ocorreu e quais parâmetros e dados de entrada foram utilizados [20].

A seguir serão discutidos os seguintes tópicos relacionados à proveniência: controle de mudanças no modelo e resultados obtidos, que gerenciam e armazenam as informações de um modelo e os resultados das execuções do *workflow*; a comparação de mudanças, que fornece ferramentas úteis para visualizar as modificações realizadas no modelo; e por fim, a linguagem de consulta, que permite realizar buscas avançadas nas informações arquivadas.

2.4.1 Controle das modificações no modelo e resultados obtidos

O ponto fundamental da proveniência está no armazenamento dos modelos, dos parâmetros e dos resultados obtidos a partir de uma execução.

Um sistema que provê esse tipo de funcionalidade deve ser capaz de criar e manter associações entre entradas e saídas do *workflow* além dos dados intermediários.

Nesse sentido, cada informação armazenada é associada a uma identificação única chamada ID. Esse ID é usado para futuras buscas e verificação no *cache*, que, como se sabe, é um dispositivo de acesso rápido, interno a um sistema, utilizado como um operador intermediário de um processo e um dispositivo de armazenamento [65].

Uma das funcionalidades mais interessantes da proveniência consiste em diminuir o tempo de execução de um *workflow* em que apenas alguns componentes tenham sido modificados. Por componente modificado entende-se tanto alteração de dados de entrada quanto substituição de componentes.

Essa técnica de execução consiste em computar apenas os componentes modificados e os que dependem desses. Os demais têm seus dados de saída recuperados a partir do *cache* da proveniência. Assim, essa se torna uma ferramenta muito útil para pesquisadores que necessitam fazer vários testes com poucas modificações em um modelo.

2.4.2 Comparação das mudanças

As cópias dos MWCs e seus resultados armazenados pelos SWC permitem que o usuário veja as diferenças entre as mudanças realizadas.

O sistema permite recuperar as mudanças de um modelo armazenado e compará-los para visualizar as diferenças entre eles. Esse procedimento torna-se muito útil para pesquisadores que precisam descrever todos os passos até chegarem ao resultado final. A Figura 2.14 demonstra uma comparação de mudanças realizada no sistema Vistrails [49]. As cores dos componentes e dos canais de transmissão dos dados facilitam a identificação das informações adicionadas em cada modificação, identificadas na legenda por *Version 'z-space'* e *Version 'textureMapper'*; das informações que não foram modificadas, identificadas como *Shared*; e dos dados de entrada e saída modificados, identificados como *Parameter Changes*.

2.4.3 Linguagem de consulta

O uso da proveniência pelos SWCs faz com que eles manipulem e armazenem diversos MWCs. Para facilitar a busca desses modelos foi criada uma linguagem de consulta rápida [49].

Existem dois tipos de métodos para consulta: busca pelo exemplo e busca textual. A primeira permite ao usuário criar modelos de consulta e buscar por estruturas e dados de entrada ou saída similares. Entretanto, esta tem uma sintaxe de pesquisa direta, que possibilita informar um texto desejado ou utilizar critérios como data, autor, etc.

2.5 Funcionalidades dos Sistemas de Workflow Científico

O suporte dado por cada SWC estudado neste trabalho em relação à taxonomia será discutido nessa seção. Suas funcionalidades foram extraídas a partir do uso de cada ferramenta, de documentos disponibilizados e de artigos relacionadas, tais

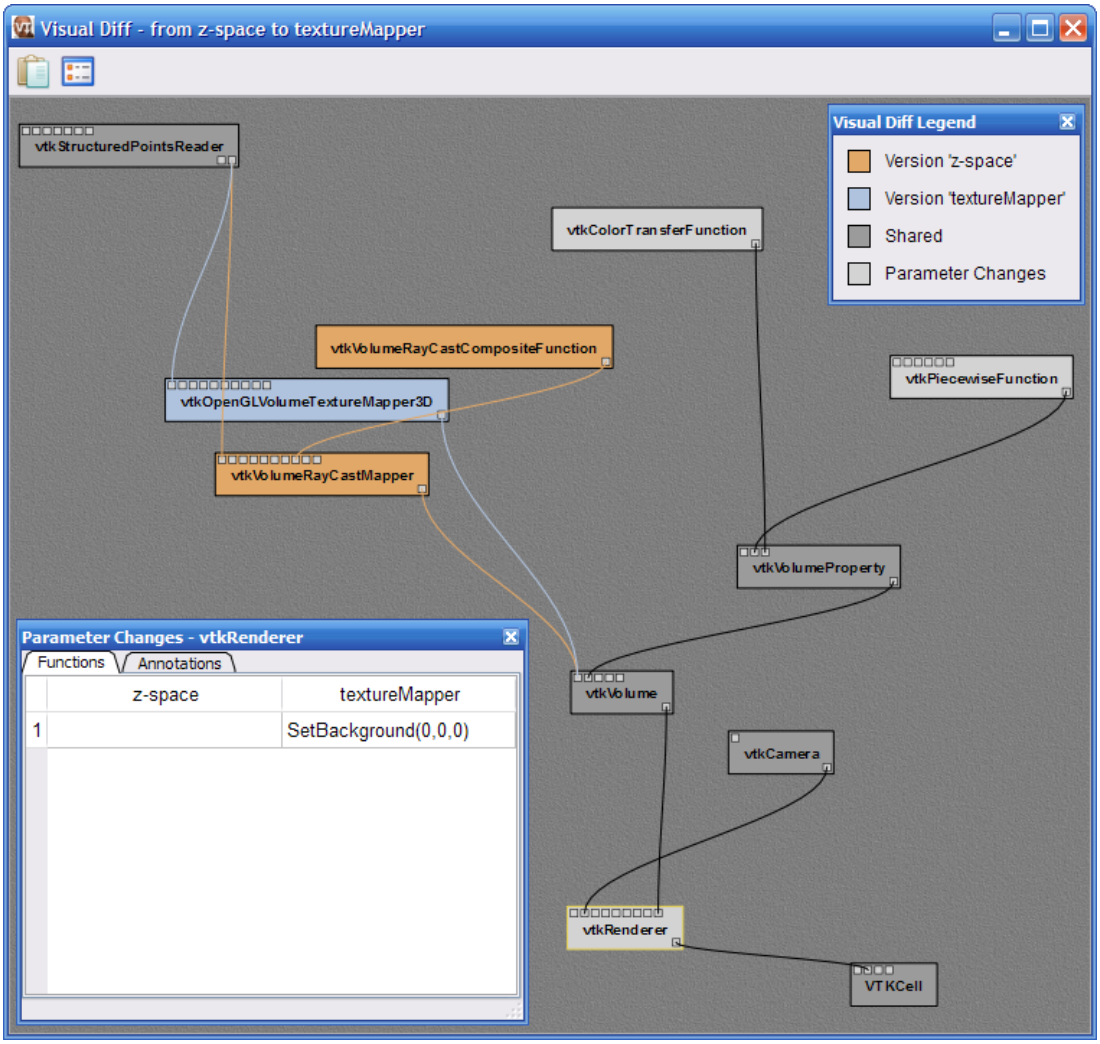

Figura 2.14: Comparação de mudanças num modelo de *workflow* no sistema Vistrails. Extraído da referência [49].

como as referências [46], [49], [58], [47], [35], [39], entre outras.

As funcionalidades de cada SWC quanto à categoria Projeto são destacadas na Tabela 2.1. Como apresentado na tabela, todos os SWC suportam a Estrutura do tipo DAG, a Composição Manual e as Extensões por Integração Interna e Externa. O sistema Vistrails é o único a não relatar suporte à Estrutura do tipo não-DAG e Tipos de Componentes classificados como Recursos; além de ser o único a ter suporte a Composição Automática. O sistema Kepler é o único a suportar Modelos Computacionais e Tipos de Componenentes classificados por Domínio de Aplicação.

As funcionalidades suportadas pelos SWCs em relação a categoria Execução são apresentadas na Tabela 2.2. Como destacados na tabela, os SWCs que dão suporte

		Kepler	Triana	Taverna	Vistrails
Estrutura	DAG	✓	✓	✓	✓
	não-DAG	✓	✓	✓	
Composição	Manual	✓	✓	✓	✓
	Automática				✓
Modelos Computacionais		✓			
Tipos de componentes	Domínio de aplicação	✓			
	Recursos	✓	✓	✓	
Extensões		✓	✓	✓	✓

Tabela 2.1: Funcionalidades dos SWC quanto à Categoria Projeto.

à Depuração são o Kepler e o Taverna. A funcionalidade de Tolerância a Falha é suportada pelos sistemas Triana e Taverna. Por fim, o sistema Triana é o único a não suportar a funcionalidade Monitoração da Execução.

	Kepler	Triana	Taverna	Vistrails
Depuração	✓		✓	
Tolerância a falha		✓	✓	
Monitoração da execução	✓		✓	✓

Tabela 2.2: Funcionalidades dos SWC quanto à Categoria Execução.

	Kepler*	Triana	Taverna	Vistrails
Modificações no modelo	✓		✓	✓
Resultados	✓		✓	✓
Comparação das mudanças	✓		✓	✓
Linguagem de consulta	✓			✓

Tabela 2.3: Funcionalidades dos SWC quanto à Categoria Proveniência.

A categoria Proveniência é descatada na Tabela 2.3, juntamente com os SWCs que dão suporte as funcionalidades dessa categoria. Neste estudo, considerou-se que

o sistema Kepler tem suporte à categoria Proveniência, apesar dessas funcionalidades estarem em fase de desenvolvimento. Dentre os SWCs estudados, o único que não tem referência alguma à Proveniência é o sistema Triana. As funcionalidades Modificações no modelo, Resultados e Comparação das mudanças são suportadas pelos sistemas Kepler, Taverna e Vistrails. A funcionalidade Linguagem de consulta é suportada pelos sistemas Kepler e Vistrails.

Capítulo 3

Suporte tecnológico

Este capítulo apresenta tecnologias que dão suporte a algumas das funcionalidades dos SWCs, descritas no capítulo anterior.

A primeira tecnologia descrita, na seção 3.1, é o XML, utilizado para transferir dados via rede ou internet. A segunda, na seção 3.2, é a linguagem *script*, que simplifica a adição de novos códigos ao sistema. Na seção 3.3 apresenta o serviço web (*web Service*), que é utilizado para execução remota de tarefas. A linguagem de descrição BPEL, mostrado na seção 3.4, é utilizada para especificar o comportamento de processos comerciais baseados em serviços web. E por fim, na seção 3.5 é apresentada a interface com Grids, que fornece meios para execução de aplicações que necessitam de alto desempenho ou compartilhamento de recursos computacionais.

3.1 *Extensible Markup Language* (XML)

O XML é uma linguagem de marcação simples, de propósito geral para criação de documentos de marcação sob medida [12, 28, 76].

Trata-se de um padrão recomendado pela *World Wide Web Consortium* (W3C), que especifica a sintaxe e os requisitos para análise das palavras. O XML é um padrão aberto e livre para uso.

Seu principal objetivo é facilitar a troca de dados estruturados entre diversos sistemas e plataformas. Muito utilizado por sistemas que trocam informações pela

internet.

Dentre os usos que os SWCs fazem do XML destacam-se o armazenamento dos MWCs em arquivos; o armazenamento dos resultados das execuções dos MWCs; a exportação e importação dos componentes para compartilhamento entre usuários dos SWCs; o armazenamento de configurações particulares dos próprios SWCs; o protocolo de comunicação dos serviços web; a configuração de Tarefas Externas e Internas; e o armazenamento de dados disponíveis nos repositórios para uso nos experimentos científicos.

Entre as linguagem baseadas no XML destacam-se: XHTML (páginas HTML), RDF, SDMX, SMIL, MathML (expressões matemáticas), NCL, XBRL, XSIL, e SVG (gráfico vetorial).

O XML é uma estrutura de árvore para armazenamento de textos ou qualquer outro tipo de dados. Sua sintaxe é bem simples e flexível, e possui como regra de formação a necessidade de existir apenas um elemento raiz na árvore. Elemento é um composto por uma chave (*tag*), valor e/ou propriedades. Uma chave é uma instrução breve marcando o início e o fim do elemento. Propriedade é um composto por par chave e valor com conteúdos simples.

O elemento raiz pode ser precedido, opcionalmente, pela declaração do XML. Essa declaração informa a versão do XML que está em uso, pode conter qual a codificação dos caracteres e outras dependências externas.

A Figura 3.1 demonstra como a sintaxe do XML pode ser utilizada para descrever uma receita de pão. O elemento raiz no exemplo é receita, além dos nós que descrevem o título, ingredientes e instruções da receita. Entre as propriedades, no nó ingrediente por exemplo, tem-se quantidade e unidade.

O XML resolve uma exigência de tecnologia fundamental que aparece em vários lugares. Oferecendo um padrão, flexível e formato de dados extensível por natureza, o XML reduz o custo de organizar as várias tecnologias necessárias para assegurar o sucesso dos serviços web.

Para edição dos XMLs, há disponível no mercado diversas ferramentas que auxiliam na leitura, na gravação e na validação de dados estruturados em XMLs.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<receita nome="pão" tempo_de_preparo="5 minutos" tempo_de_cozimento="1 hora">
  <titulo>Pão simples</titulo>
  <ingredientes>
    <ingrediente quantidade="3" unidade="xícaras">Farinha</ingrediente>
    <ingrediente quantidade="7" unidade="gramas">Fermento</ingrediente>
    <ingrediente quantidade="1.5" unidade="xícaras" estado="morna">Água</ingrediente>
    <ingrediente quantidade="1" unidade="colheres de chá">Sal</ingrediente>
  </ingredientes>
  <instrucoes>
    <passo>Misture todos os ingredientes, e dissolva bem.</passo>
    <passo>Cubra com um pano e deixe por uma hora em um local morno.</passo>
    <passo>Misture novamente, coloque numa bandeja e asse num forno.</passo>
  </instrucoes>
</receita>
```

Figura 3.1: Exemplo de um XML que descreve uma receita de pão. Extraído da referência [76]

3.2 *Scripts*

Os *script* dão suporte aos SWCs na subcategoria Extensões da categoria Projeto, da taxonomia apresentada na Seção 2.2.5. 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.

O uso de *script*s facilita muito a manutenção de códigos, pois não necessitam ser compilados como as linguagens de programação tradicionais, por serem interpretados pelo sistema [28].

Dentre os SWCs estudados foram encontradas as seguintes linguagens de *script*s: Python® (Kepler e Vistrails) e BeanShell®[1] (Taverna).

Python® é uma linguagem de altíssimo nível de abstração, orientada a objetos, com tipagem forte (não há conversões automáticas) e dinâmica (não há declaração de variáveis), multiplataforma, extensível para implementação em conjunto com outras linguagens e interpretada via bytecode.

Beanshell® é um interpretador de comandos escrito em Java®, livre e de código aberto[2], com características de uma linguagem de *script* pequena, simples e

[1]BeanShell®: linguagem de *script* baseada no Java®. `http://www.beanshell.org`

[2]Código aberto: A OSI (*Open Source Initiative*) define o código aberto usando a definição Debian de *software* livre que é apenas um detalhamento das quatro liberdades FSF (*Free Software Foundation*). Dessa forma todo *software* de código aberto é também *software* livre [66].

integrada à plataforma Java®. O interpretador executa comandos e expressões em Java®, além de estender o Java® no domínio de *script* com convensões e sintaxe de linguagem de *script*.

O exemplo de *script* apresentado na Figura 3.2 é um código interpretado pelo Beanshell®, que faz a concatenação das três variáveis `seq1`, `seq2` e `seq3`, que são recebidas como dados de entrada do componente, e disponibiliza o resultado na variável `fasta`, que é enviada como dado de saída do componente.

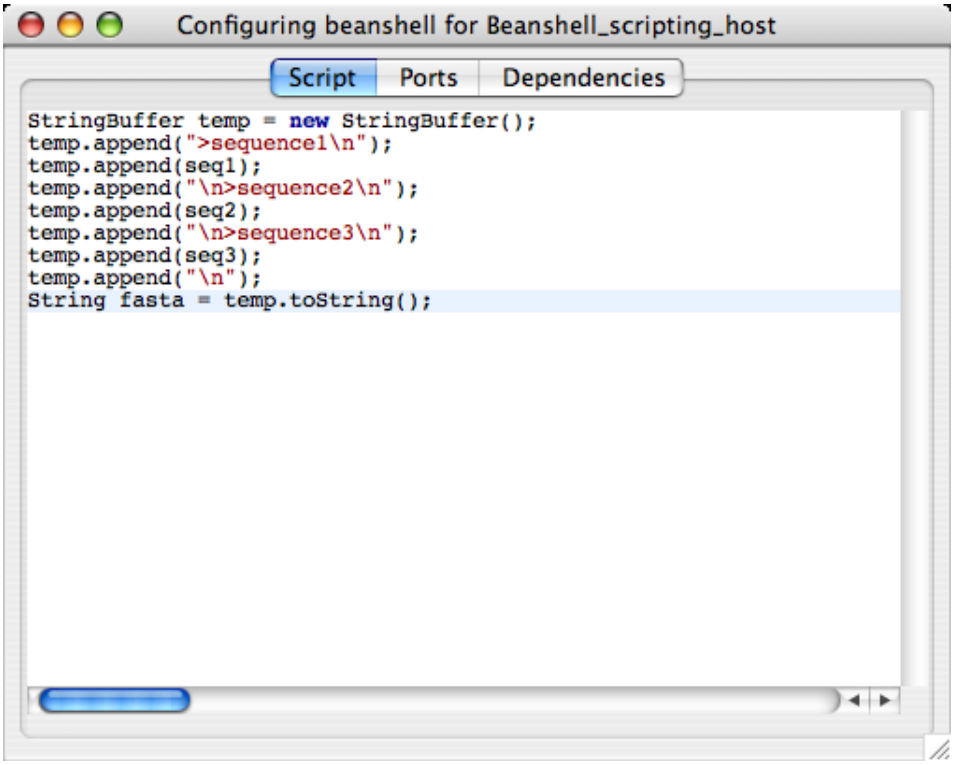

Figura 3.2: Tela de adição de *script* na linguagem BeanShell no sistema Taverna. Extraído da referência [47].

3.3 Serviços web

O serviço web (*web services*) [11] é uma tecnologia que permite a integração entre sistemas e aplicações. Os serviços web são aplicações independentes de plataformas, banco de dados, linguagens de programação e métodos de desenvolvimento. Essas características dão aos desenvolvedores a liberdade para criação de suas aplicações em contraste às limitações previamente encontradas na interface entre aplicativos

[74].

Em relação à taxonomia, os serviços web dão suporte aos SWCs na categoria Extensões do Projeto. Tais sistemas têm componentes que tornam transparente o uso de serviços web, desde que o usuário conheça a localização do serviço, será possível que os componentes reconheçam automaticamente, ainda no momento de sua configuração, os dados de entrada e saída do serviço.

O uso de serviços web para extender as tarefas dos SWCs está presente em todas as ferramentas estudadas. Esta característica é suportada nos sistemas com diversas facilidades para sua inclusão no MWC desenvolvido pelo usuário.

A Figura 3.3 apresenta um exemplo de configuração de um componente *web service* no sistema Kepler, que é uma implementação de um cliente genérico para serviços web. Nos campos de configuração tem-se o endereço do serviço web (*wsdlUrl*), o nome do método a ser executado (*methodName*), o nome do usuário (*userName*) e senha (*password*) caso seja necessário autenticar para ser executado. Os campos *hasTrigger*, *class*, *semanticType00* e *semanticType11* são campos utilizados pelo sistema Kepler.

A W3C (*World Wide Web Consortium*) define o serviço web como um sistema desenvolvido para apoiar a interoperabilidade entre máquinas sobre uma rede de computadores. Ele tem uma interface descrita num formato de máquina processável, especificamente o WSDL. Outros sistemas interagem com o serviço web numa maneira prescrita pela sua descrição usando mensagens SOAP, tipicamente transportadas usando HTTP (*Hypertext Transfer Protocol*) com uma serialização XML em conjunto com outros padrões web relacionados [10].

O WSDL (*Web Services Description Language*) é uma linguagem para descrição de serviços web. Este descreve tais serviços, inicialmente, pelas mensagens que são trocadas entre o agente requisitante e o agente fornecedor. As mensagens por si só são descritas abstratamente para somente depois serem ligadas a um protocolo de rede de computadores concreto e um formato de mensagem.

O SOAP (*Simple Object Access Protocol*) provê um *framework* padrão, extensível, para empacotamento e transporte de mensagens XML. No contexto da arquitetura do serviço web, SOAP também fornece um mecanismo conveniente para provê referências, tipicamente pelo uso de cabeçalhos. Geralmente servidores SOAP ficam

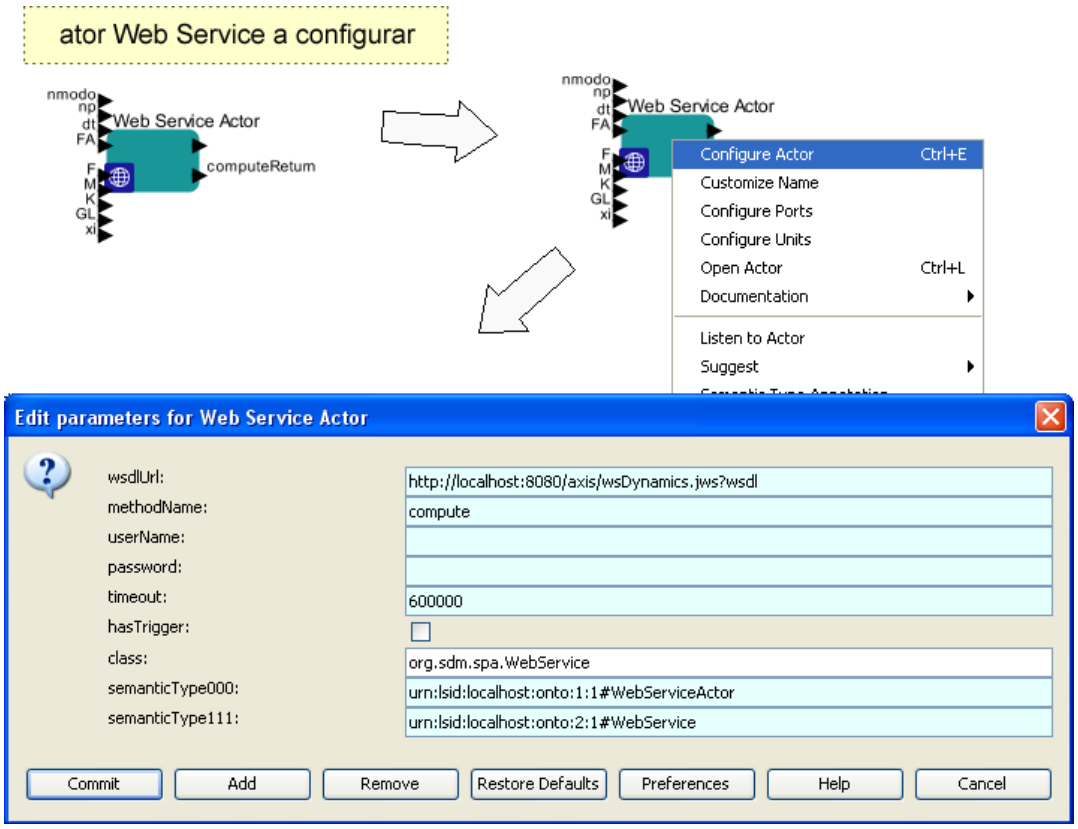

Figura 3.3: Configuração de um componente serviço web no sistema Kepler.

hospedados em servidores HTTP. Este é um protocolo de comunicação utilizado para transferir informações em redes locais ou internet [68].

A W3C ainda define alguns conceitos para melhorar o entendimento da arquitetura dos serviços web. Os conceitos destacados pela entidade são agentes e serviços, requisitantes e provedores, descrição do serviço e semântica. Uma visão geral de como é empregado um serviço web é apresentada no final desta seção e os conceitos são comentados a seguir.

3.3.1 Agentes e serviços

Um serviço web é uma noção abstrata do que deve ser implementado pelo agente concreto. O agente é um pedaço concreto de *software* ou *hardware* que envia e recebe mensagens, enquanto o serviço é o recurso caracterizado pelo conjunto abstrato de funcionalidades que são fornecidas [10].

Para ilustrar essa distinção pode-se citar a criação de um agente num determi-

nado dia, e no dia seguinte é construído outro agente diferente do primeiro com a mesma funcionalidade. Embora o agente tenha mudado, o serviço continua sendo o mesmo.

3.3.2 Requisitantes e provedores

O propósito de um serviço web é, primeiramente, provê alguma funcionalidade em interesse de seu proprietário, que pode ser uma pessoa ou uma organização. A entidade provedora é uma pessoa ou organização que provê um agente apropriado para implementar um serviço particular [10].

A entidade requisitante é uma pessoa ou organização que deseja fazer uso do serviço disponibilizado pela entidade provedora. Ela geralmente usa um agente requisitante para trocar mensagens com o agente provedor da entidade provedora.

Uma pessoa ou organização pode ser considerada o proprietário dos agentes que fornecem ou requisitam serviços web. O interesse desses agentes pode ser comercial ou particular.

3.3.3 Descrição do serviço

A mecânica de troca de mensagens é documentada numa descrição de serviço web chamada WSD (*Web Service Description*). O WSD é uma especificação de máquina processável da interface do serviço web, escrita em WSDL.

Essa especificação define o formato das mensagens, tipo de dados, protocolos de transporte e formatos de serialização utilizados no transporte que são usados entre os agentes requisitantes e fornecedores. Especifica também um ou mais locais de rede de computadores no qual um agente provedor pode ser encontrado, e pode prover informações sobre o modelo de mensagens que são esperadas.

Basicamente, a descrição do serviço representa um acordo do mecanismo de interação com aquele serviço.

3.3.4 Semântica

A semântica refere-se ao estudo do significado, e seu objetivo é totalmente diferente da sintaxe. Enquanto esta lida com a estrutura formal de como algo é expresso, a semântica preocupa-se com o que algo significa. Em termos computacionais, o estudo da semântica é a definição de uma linguagem de representação formal para capturar a semântica de forma processável por máquinas, obtendo uma interpretação consistente [16, 70].

A semântica de um serviço web é a expectativa compartilhada sobre o comportamento de um serviço, em particular na resposta para as mensagens que são enviadas para o mesmo [10].

Em outras palavras, é o acordo entre a entidade requisitante e a entidade provedora respeitando o propósito e as conseqüências da interação. Embora esse contrato represente o entendimento completo entre ditas entidades em como e porque seus respectivos agentes irão interagir, não há necessariamente um acordo escrito ou negociado explicitamente. Esse acordo pode ser explicito ou implícito, oral ou escrito, processado por máquina ou orientado por humanos, pode ser legal ou informal.

Enquanto a descrição do serviço representa um contrato do mecanismo de interação com um serviço particular, a semântica representa um contrato do significado e do propósito da interação.

3.3.5 Visão geral do emprego de um serviço web

Existem diversas maneiras para uma entidade requisitante entrar em contato e fazer uso de um serviço web. Geralmente, os seguintes passos são necessários, como ilustrado na Figura 3.4: as entidades requisitante e provedora tomam conhecimento um do outro, ou pelo menos um toma conhecimento do outro; ambas as entidades, de certo modo, concordam com a descrição do serviço e a semântica que irá direcionar a interação entre os agentes requisitantes e provedores; a descrição do serviço e a semântica são entendidas por ambos agentes; e os agentes requisitante e provedor trocam mensagens, executando assim alguma tarefa em prol das entidades requisitantes e provedoras. Alguns desses passos podem ser automatizados, enquanto outros podem ser executados manualmente.

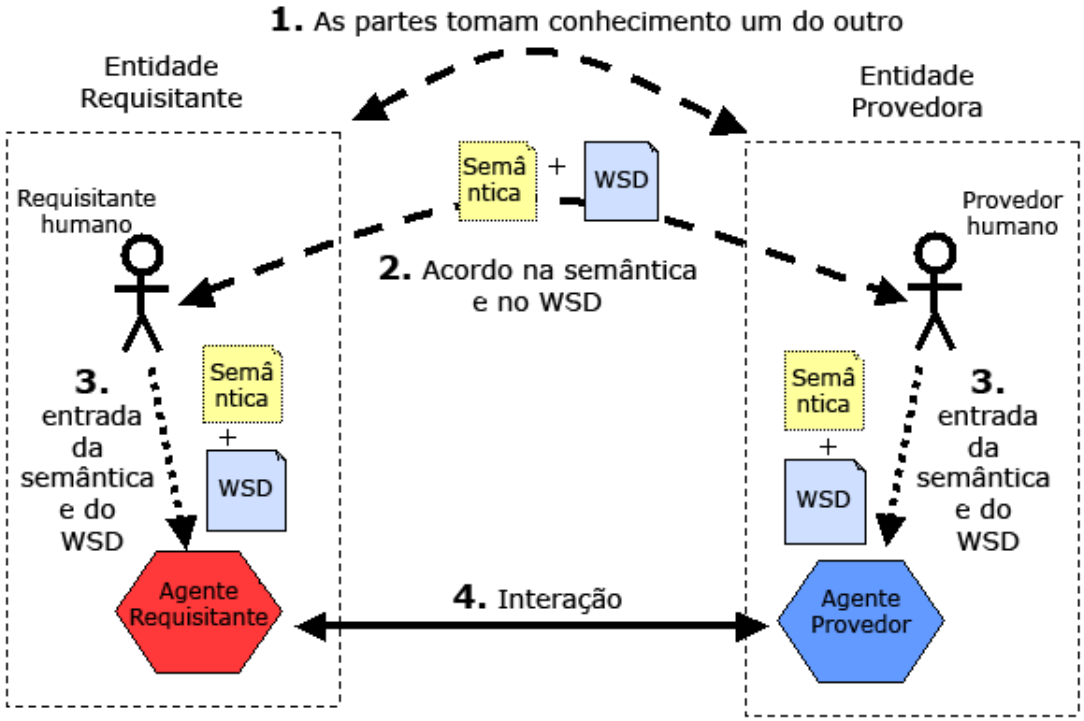

Figura 3.4: Arquitetura de um processo geral do emprego do serviço web. Figura extraída da referência [10]

Esses passos podem ser vistos com mais detalhes na referência [10], mais especificamente na Seção *Web Service Discovery*.

3.4 Linguagem para descrição de *workflows* (WS-BPEL)

O WS-BPEL, em relação à taxonomia, dá suporte aos SWCs na categoria Execução, Seção 2.3. O sistema Triana tem uma extensão para ler e gravar arquivos WS-BPEL [53], que facilita a especificação de processos científicos, pois os usuários podem utilizar o ambiente gráfico de composição de MWC desse sistema para criarem processos WS-BPEL.

O WS-BPEL (*Web Services Business Process Execution Language*), ou simplesmente BPEL, é uma linguagem para especificação do comportamento de processos baseados em serviços web [2]. Processos no WS-BPEL exportam e importam funcionalidades somente pelo uso das interfaces dos serviços web [64].

Originalmente o WS-BPEL foi utilizado para especificar processos comerciais, entretanto suas funcionalidades tem sido aplicadas na descrição de execuções de Workflows Científicos [1]. Diversos serviços web podem ser compostos em um Workflow Científico complexo, onde podem existir vários domínios e organizações envolvidas. Essa situação torna-se mais complexa quando combina-se serviços web espalhados geograficamente em um único serviço composto, na forma de um workflow. Como há um crescente interesse no uso de serviços web para computação científica, paralela e distribuida, o WS-PBEL é uma opção para composição desses Workflows Científicos a partir de serviços web.

WS-BPEL define um modelo e uma gramática para descrição do comportamento de processos baseados na interação entre eles e seus parceiros. A interação com cada parceiro ocorre através de conexões entre serviços web, e a estrutura do relacionamento para o nível de interface é encapsulada no que é nomeado de *partnerLink*. Uma vez que um processo WS-BPEL é uma definição, ela pode ser reutilizada e reaproveitada de diferentes maneiras e em diferentes cenários.

O processo WS-BPEL define como múltiplas interações de serviços com seus parceiros são coordenados para alcançar um objetivo, assim como o estado e a lógica necessária para essa coordenação.

O WS-BPEL tem um mecanismo sistemático capaz de tratar as exceções e falhas que ocorram nos processamentos. Além disso, WS-BPEL propõe um mecanismo para definir como atividades individuais ou compostas, internas a uma unidade de trabalho, serão compensadas em casos em que ocorram exceções ou uma requisição revogada por um parceiro.

Os processos podem ser descritos de duas formas: de modo abstrato ou de modo executável. O WS-BPEL pode modelar o comportamento de ambos as formas de descrição de processos, executável e abstrato. O modo executável modela o comportamento atual do participante na interação entre os processos, ou seja, o que de fato será executado e com as regras bem definidas. O modo abstrato constitui em um processo definido parcialmente, que não tem a intenção de ser executado de fato. Ele pode esconder alguns detalhes operacionais requeridos de um processo concreto.

Entre os conceitos para descrição de processos no WS-BPEL, destacam-se:

– os processos trabalham com ações dependentes dos dados. Em uma cadeia de abastecimento, por exemplo, o processo depende dos seguintes dados: o número de items do pedido; o valor total do pedido; ou a data limite para entrega;

– possibilidade de especificar situações de exceção e suas consequências, além de sequências de reparação. A definição do comportamento para o caso de tudo funcionar corretamente é a funcionalidade menos importante num processamento;

– interações com longo tempo de execução geralmente têm multiplas unidades menores de trabalho, onde cada uma tem seus próprios dados requeridos. Os processos frequentemente necessitam de coordenação entre os parceiros para obterem o status final, de falha ou sucesso, das execuções das unidades de trabalhos nos vários níveis de granularidade.

Para uma análise mais detalhada sobre WS-BPEL, recomenda-se a leitura da referência [2].

3.5 Interfaces com grids

A tecnologia de Grid dá suporte para funcionalidades classificadas na categoria Projeto da taxonomia, dentro da subcategoria Extensões, Tarefas Externas, na Seção 2.2.5. As ferramentas estudadas fazem uso da integração com Grids tanto por meio de protocolo próprio, quanto por meio de serviços web.

Os Grids Computacionais surgiram em meados da década de 90 com a promessa de viabilizar a computação de aplicações paralelas em recursos geograficamente espalhados e pertencentes a diversas organizações. Essa proposta tinha duas principais inovações: a primeira tinha a intenção de fornecer uma plataforma muito mais barata para execução de aplicações distribuídas, comparadas aos supercomputadores paralelos da época; a segunda era a possibilidade de executar aplicações paralelas em uma escala muito maior que um supercomputador seria capaz de fazê-la, isso através da aglomeração de recursos espalhados [60].

Depois de um tempo, com a evolução da tecnologia de Grids, percebeu-se que a composição automática de um conjunto de recursos, para servir uma aplicação, criava a oportunidade de oferecer serviços sob demanda. Assim, surgiu o conceito de um Grid no qual é possível prover sob demanda qualquer serviço computacional, não apenas serviços para computação de alto desempenho.

Como conseqüência, as tecnologias de Grids Computacionais se fundiram com os serviços web e se tornaram uma tecnologia fundamental para computação no século XXI [60].

De fato, as tecnologias de Grid sempre receberam atenção pela sua capacidade de prover ampla colaboração interativa entre indivíduos e instituições, serviços de gerenciamento global e compartilhamento de recurcos computacionais.

Entre os grupos de trabalho em ambiente Grid e ferramentas de tecnologias de Grid estão o Globus [3], o myGrid, GridLab e EcoGrid [4].

Para uma leitura mais profunda sobre a tecnologia Grid e suas principais características, recomenda-se a leitura das referências [32] e [31]. Para informações a respeito de Grid no contexto de *workflow* científico há o fórum na referência [26].

A arquitetura de um ambiente Grid, de acordo com a referência [25], é frequentemente descrita em termos de camadas, onde cada camada tem uma função específica. As camadas mais altas são geralmente destinadas para usuários, enquanto as camadas mais baixas são focadas no *hardware* (computadores e redes).

A arquitetura é dividida em 4 partes, ilustradas na Figura 3.5, comentadas a seguir, da mais baixa para a mais alta:

camada de rede essa é responsável por conectar os recursos do Grid;

camada de recursos é nessa camada que reside os recursos do Grid, tais como computadores, sistemas de armazenamento, catálogo eletrônico de dados, sensores, entre outros;

[3]Globus: ambiente Grid para compartilhamento de recursos computacionais. `http://www.globus.org`

[4]EcoGrid: ambiente Grid com recursos na área da ecologia. `http://seek.ecoinformatics.org/Wiki.jsp?page=EcoGrid`

camada de mediação essa camada provê ferramentas para permitir a participação de vários elementos (servidores, armazenamento, redes, etc) no Grid. A camada de mediação, às vezes, é a inteligência atrás do Grid computacional;

camada de aplicação e serviços essa camada inclue aplicações na ciência, na engenharia, nos negócios, nas finanças, bem como portais na internet e ferramentas para desenvolvimento, todas para darem suporte às aplicações. É por meio dessa camada que o usuário interage com o Grid. Além de incluir serviços que executam funções de gerenciamento tais como rastreamento de quem está provendo os recursos e quem os está usando.

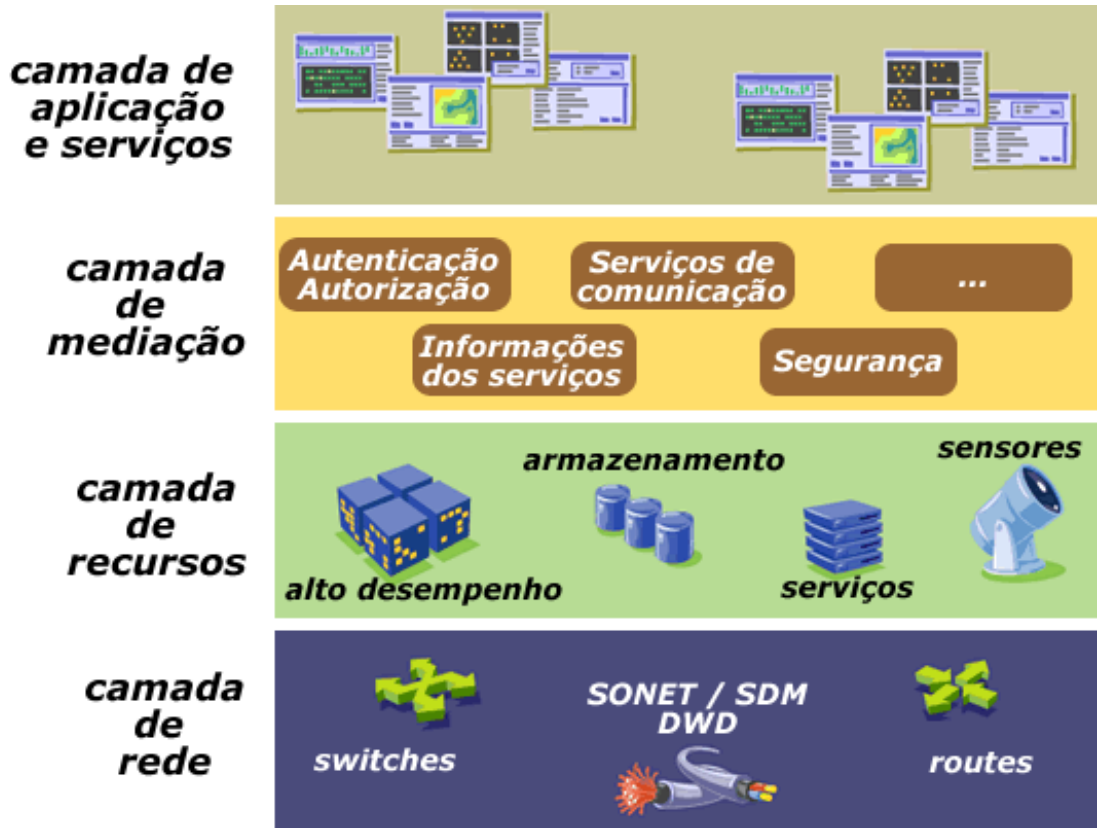

Figura 3.5: Ilustração de arquitetura grid dividida em camadas. Extraído da referência [25].

Embora este trabalho tenha como um dos objetivos a avaliação do desempenho da execução dos MWCs, esta avaliação se limita ao desempenho com relação ao custo adicional de processamento ou armazenamento que, como conseqüência, piora o desempenho dos SWCs em relação aos sistemas legados. O estudo mais detalhado sobre técnicas para computação de alto desempenho está fora do escopo deste

trabalho.

Capítulo 4

O Sistema de Workflow Científico Kepler

Dentre os SWCs apresentados na Introdução, foi escolhido para desenvolvimento do estudo de caso mostrado no Capítulo 6, o sistema Kepler.

O sistema Vistrails, com a implementação da proveniência, tem destaque nas pesquisas exploratórias, onde as mudanças no MWC e nos resultados obtidos são requisitos importantes. Apesar disso, seus componentes, em sua maioria, são para executar tarefas de processamentos de imagens.

Os principais componentes do SWC Triana são destinados para processamento de sinais, de texto e de imagens. Além de ter um procedimento de instalação complicado, onde é preciso copiar arquivos, configurar variáveis de ambiente e limitações quanto a espaços nos nomes de diretórios da aplicação e do JRE (*Java Runtime Environment*) [1]. Outro ponto negativo desse sistema é a falta de atividade do projeto, que lançou sua última versão em julho de 2006.

As funcionalidades disponíveis no SWC Taverna são focadas apenas na área de biologia e bioinformática. Suas funcionalidades de extensão das tarefas do SWC são limitadas ao uso de uma linguagem script BeanShell® e o *software* R®.

Assim, a escolha do SWC Kepler para realização do estudo de caso deve-se às características destacadas a seguir:

[1] JRE: conjunto de bibliotecas necessárias para executar códigos Java.

- possui uma interface gráfica amigável, com um conceito de componentes organizados em blocos de construção, onde o usuário move os blocos, ou componentes, para a área de composição de MWCs;
- possui um conceito de modelos computacionais, que descrevem como será o comportamento da execução de um MWC;
- possui uma documentação de fácil entendimento com exemplos práticos de vários tipos de MWCs;
- possui um procedimento de instalação prático;
- possui um código aberto escrito na linguagem Java®;
- possui uma documentação instrutiva para criação de novos componentes;
- possui um objetivo de ser funcional para diversas áreas de conhecimento;
- possui uma constante atividade do grupo de desenvolvimento; e
- possui um suporte à maioria das funcionalidades apresentadas na taxonomia, no Capítulo 2.

Esse capítulo apresenta uma introdução às suas principais funcionalidades, em detalhes suficientes para preparar o leitor para a leitura do Capítulo 6, que discute o estudo de caso. Para um entendimento mais aprofundado de como utilizar esse *software*, recomenda-se a leitura das referências [45] e [46].

A Seção 4.1 apresenta uma visão geral do sistema Kepler. Na Seção 4.2 é abordada a criação de *workflows* científicos no Kepler. Na Seção 4.3, são apresentados a interface gráfica e os componentes básicos do Kepler. Por fim, a Seção 4.4 relaciona as categorias da taxonomia de *workflow* científico que são suportadas pelo sistema Kepler.

4.1 Introdução ao Kepler

O principal objetivo da comunidade desenvolvedora do sistema analisado é criar um SWC de código aberto, que facilite a criação de MWCs, além de executá-los de

forma eficiente, tanto localmente como pela computação distribuída por meio da tecnologia emergente baseada em Grid. O Kepler foi construído a partir do *framework* Ptolemy II, que é uma ferramenta para estudos de modelos computacionais.

Trata-se de uma aplicação criada para análise e modelagem de dados científicos. Ao usar sua interface gráfica e seus componentes, cientistas com pouco conhecimento em ciência da computação são capazes de criar MWCs executáveis. O SWC Kepler fornece ferramentas para acessar dados científicos (imagens médicas e de satélite, dados observados, processamento de sinais, etc) e para executar análises complexas nos dados recuperados.

A Figura 4.1 ilustra a interface gráfica do Kepler através de um MWC, na área de ecologia, que processa dados de ocorrências de espécies animais, para criar um modelo de nicho ecológico, e o resultado é apresentado numa imagem com cores para destacar as regiões com maior ocorrência de cada espécie.

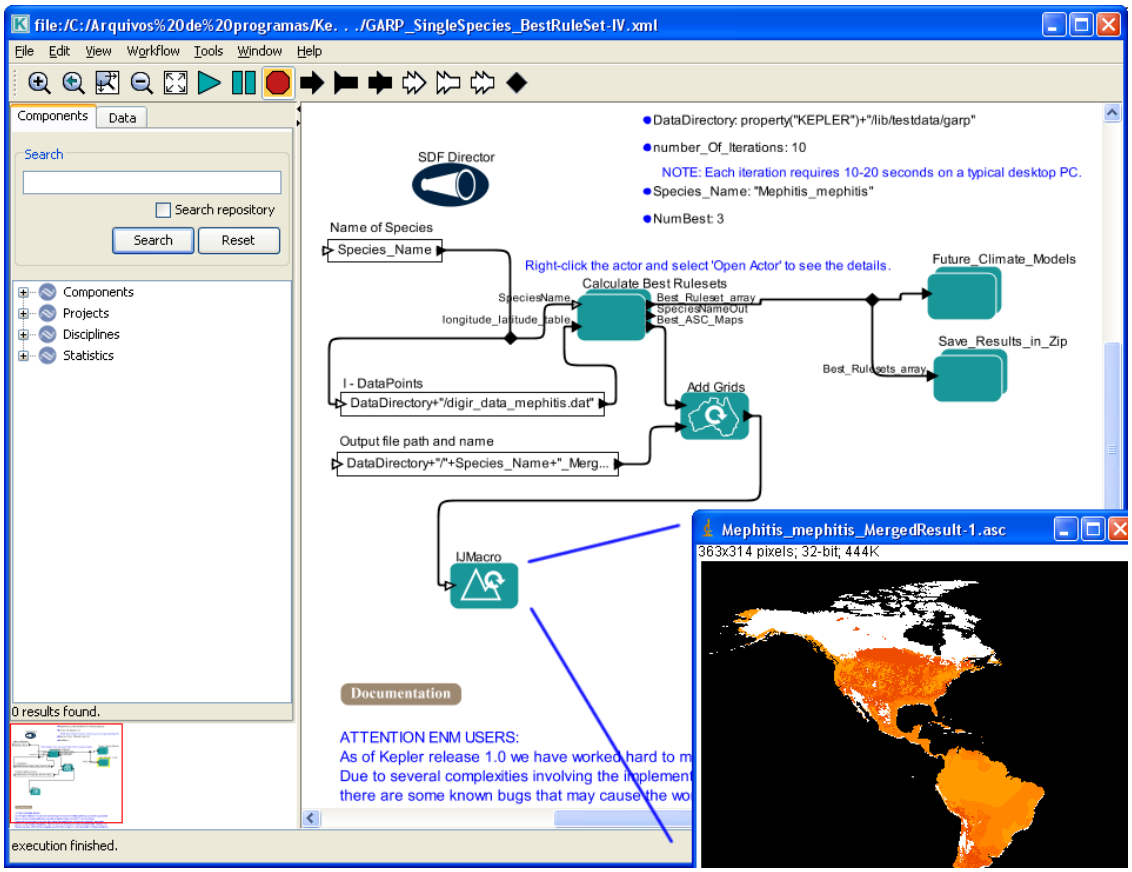

Figura 4.1: Exemplo de *workflow* científico no sistema Kepler. Extraído da referência [46].

O SWC Kepler armazena os MWCs num formato que pode ser facilmente trocado entre usuários da ferramenta, arquivado, gerenciado com controle de versão e executado. Esse formato é denominado MoML (*Modeling Markup Language*) [7], descrito em XML, cujas marcações foram definidas pelo sistema Ptolemy.

Essa ferramenta faz uso da programação visual para composição dos MWCs, além de utilizar uma modelagem orientada a atores. Este nome é devido ao fato de que os componentes de modelagem nessa ferramenta são denominados com tal. A composição de MWCs é feita através dos atores organizados em blocos de montagens, adicionando-se componentes na área de composição e ligando-os através dos canais de comunicação. Essa organização facilita a criação de MWCs e a reutilização de componentes.

Além dessa abordagem da programação visual, um grande número de componentes são disponibilizados pelo Kepler. Esses componentes visam resolver problemas de aplicações científicas, tais como acesso de metadados e dados remotos, transformações de dados, análise de dados, interface com aplicações legadas, serviços web, proveniência e vários outros. Esse número de componentes cresce constantemente devido à contribuição de projetos em colaboração com o Kepler. Os principais domínios de aplicação alvo do sistema Kepler são bioinformática, química computacional, eco-informática e geo-informática.

O SWC Kepler oferece variados graus de granularidade de componentes para composição de MWCs, permitindo eficiência, através do uso de componentes com granularidade grande, e reuso, através do uso de componentes com granularidade fina.

O sistema Kepler provê um modelo para anotação semântica de seus componentes, usando terminologias da ontologia. Essas anotações suportam várias características avançadas, incluindo recursos de buscas, validação automática e edição aperfeiçoada do MWC [6]. As classificações e buscas semânticas implementadas no sistema Kepler estão fora do objetivo desse trabalho. Para um melhor entendimento da semântica utilizada no sistema Kepler, recomenda-se a leitura das referências [6] e [18].

4.2 *Workflows* científicos no Kepler

O sistema Kepler simplifica o esforço necessário para modelar e analisar dados científicos, ao usar uma representação gráfica dos seus processos. Essa representação, do MWC, descreve o fluxo de dados entre os componentes da modelagem.

Destaca-se, no sistema estudado, o mecanismo de arrastar e soltar componentes dentro da área de composição do MWC, que auxilia bastante os usuários a criarem seus próprios modelos executáveis. Além disso, os componentes são ligados, para construir um fluxo de dados específico, através dos canais de comunicação. Como a representação de todo o *workflow* é visual, torna-se fácil o entendimento de como o fluxo de dados ocorre de um componente para outro.

O Kepler possibilita que atores sejam criados e enviados para um repositório na internet, onde serão compartilhados com outros usuários desse sistema. Esses novos atores podem ser baixados e adicionados ao Kepler e utilizados normalmente nos *workflows*.

O Kepler permite, ainda, que um MWC seja embutido num único componente chamado Ator Composto (*Composite Actor*), cuja definição é elaborada em outro diagrama de MWCs. Atores compostos são muito úteis para criar componentes específicos que façam operações complexas e que sejam reaproveitados em outros MWCs. Esse componente do Kepler está classificado na taxonomia como Recurso Encapsulamento, em Tipo de Componente, na Seção 2.2.4.

Usuários do Kepler com pouco conhecimento de computação científica conseguem criar MWCs com os componentes padrões do sistema, ou ainda modificar os modelos existentes para adaptar às suas necessidades. Análises quantitativas ou estatísticas podem ser feitas usando os mecanismos de extensão para criar rotinas na linguagem R®.

Outro exemplo de Tarefas externas são os componentes do Kepler que permitem o acesso às tecnologias de computação distribuída, que são usadas pelos usuários para compartilhar seus dados e MWCs com outros usuários por meio de repositórios, além da computação de alto desempenho. A utilização desses componentes diminuem a complexidade do uso dessas tecnologias, automatizando a comunicação com os servidores Grid.

A interface de Grid fornecida pelo Kepler é para o sistema Globus, que é uma comunidade que desenvolve tecnologias para Grid e permite que pessoas compartilhem computação de alto desempenho, banco de dados e outras ferramentas corporativas.

Os MWCs no sistema Kepler podem ser executados a qualquer momento durante sua composição sem a necessidade de compilação. O Kepler também permite que seus usuários examinem e visualizem os dados durante a execução do MWC. Essa funcionalidade é particularmente útil, pois auxilia na modelagem e junto com a outra característica do Kepler, permite que se mude facilmente os parâmetros do modelo, quando necessário, produzindo uma maior variedade de resultados experimentais.

4.3 Interface e componentes básicos

A interface gráfica de edição de *workflows* do sistema Kepler é compreendida pelas seguintes seções:

Barra de menu disponibiliza acesso a todas as funcionalidades do Kepler;

Barra de ferramentas disponibiliza acesso rápido às funcionalidades mais comuns;

Área de componentes e dados viabiliza a visualização de abas para realizar pesquisas de componentes e de dados do Kepler, além de apresentar uma "árvore" de componentes agrupados de acordo com suas características;

Área do *workflow* área disponível para o usuário incluir os componentes, conexões, modelo computacional e comentários;

Área de navegação a área de navegação apresenta todo o *workflow* reduzido para que o usuário possa clicar numa região e o Kepler posicionar os componentes na tela. Esse recurso é muito usado quando o *workflow* não cabe na área do *workflow*.

A interface gráfica do Kepler pode ser vista na Figura 4.2, que destaca todas as partes do sistema.

Um MWC no Kepler é composto de vários elementos: o Diretor (*directors*), que define o modelo computacional; os Atores (*actors*), que representam os componentes

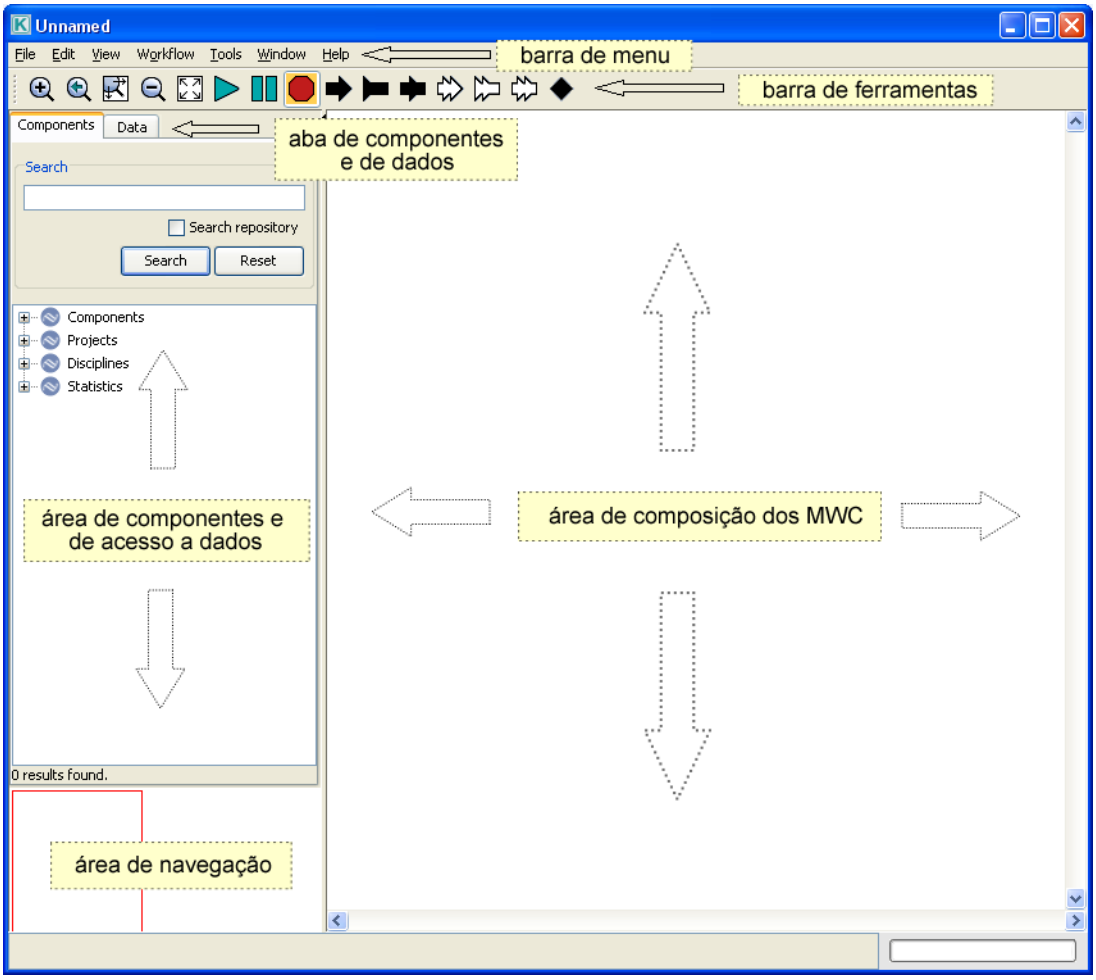

Figura 4.2: Interface gráfica do sistema Kepler com suas principais partes destacadas. Extraído da referência [46].

do *workflow*; os Parâmetros e as Portas (*parameters*; *ports*), que representam os dados de entrada e de saída; além de conexões que possibilitam a comunicação entre os componentes. A Figura 4.3 apresenta um MWC no Kepler com os principais elementos identificados.

O MWC mostrado na Figura 4.3 é um modelo de Lotka Volterra, também denominado "*Presa versus predador*", usado para modelar a relação entre duas populações, a de predadores e suas presas, no tempo. A figura mostra também o resultado da execução do modelo que são gráficos produzidos por componentes para esse fim (*TimedPlotter* e *XYPlotter*).

Os elementos Atores são a base da construção de MWC no Kepler, que traz um conjunto de atores prontos para serem usados. Cada ator é desenvolvido para

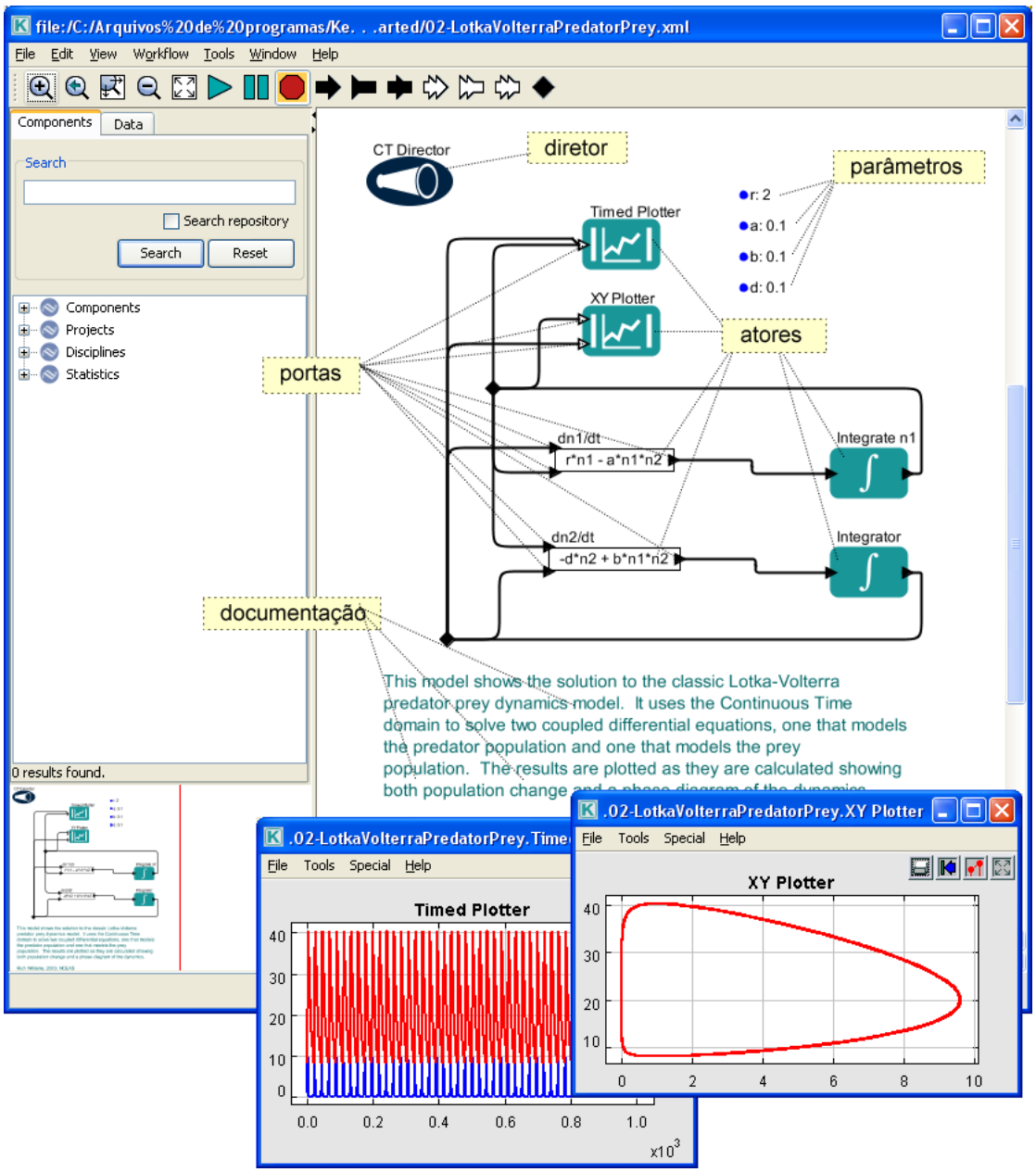

Figura 4.3: Principais elementos de um *workflow* científico no sistema Kepler. Extraído da referência [46].

executar uma tarefa específica. Por exemplo, uma seqüência bem característica de atores na composição de um *workflow* é como se segue: um ator é usado para ler ou importar dados que serão usados no *workflow*, outro ator faz a transformação desses dados para outro formato para ser analisado, um terceiro faz a análise dos dados ou esboça um gráfico com os dados, e por fim, outro ator envia os dados resultantes para um arquivo no disco ou no monitor. Esses dados passam de um ator para outro através de conexões, representadas no Kepler por linhas na área de edição do

workflow.

Todos os atores no Kepler contêm uma ou mais Portas, usadas para receber ou enviar os dados de entrada e saída do componente. Os atores são conectados através dessas portas, que indicam o fluxo de execução do *workflow*. Essas conexões entre portas são chamadas de canal (*channel*). O sistema Kepler tem um tipo especial de porta para ser adicionada nos *workflows* embutidos, utilizada para trocas de dados entre o componente composto e o *workflow*.

Outra funcionalidade disponível no Kepler é o uso de Parâmetros (*parameter*) do *workflow*, que são variáveis que recebem valores iniciais para execução do *workflow*. Esses dados podem ser utilizados no controle da execução do *workflow* e/ou nas computações dos componentes.

O elemento Ligação (*relation*) é utilizado no sistema Kepler para criar ramificações de um canal de comunicação. Ou seja, os dados de saída de um componente podem ser enviados para portas de entrada de vários componentes.

Outra funcionalidade do Kepler é a janela de execução que facilita o acompanhamento da execução do *workflow*, além de permitir pausas e mudanças nos parâmetros e variáveis do MWC. Um exemplo da janela de execução é apresentada na Figura 4.4 com a execução do *workflow* de Lotka Volterra, da presa versus predador.

4.4 Categorias da taxonomia suportadas pelo Kepler

O sistema Kepler suporta alguns elementos da taxonomia nas categorias projeto e execução, que serão comentadas com mais detalhes a seguir. Entretanto, o Kepler não tem suporte para a proveniência, que está em fase de discussão pelos membros do projeto. Há registros de sugestões de como implementar a proveniência no sistema Kepler e disponibilizá-la para seus usuários. Para conhecer detalhes sobre essas sugestões, recomenda-se a leitura das referências [30], [20] e [19]. A Tabela 4.1 apresenta resumidamente quais categorias o sistema Kepler suporta.

Como apresentado na Tabela 4.2, o Kepler suporta, em relação a categoria Pro-

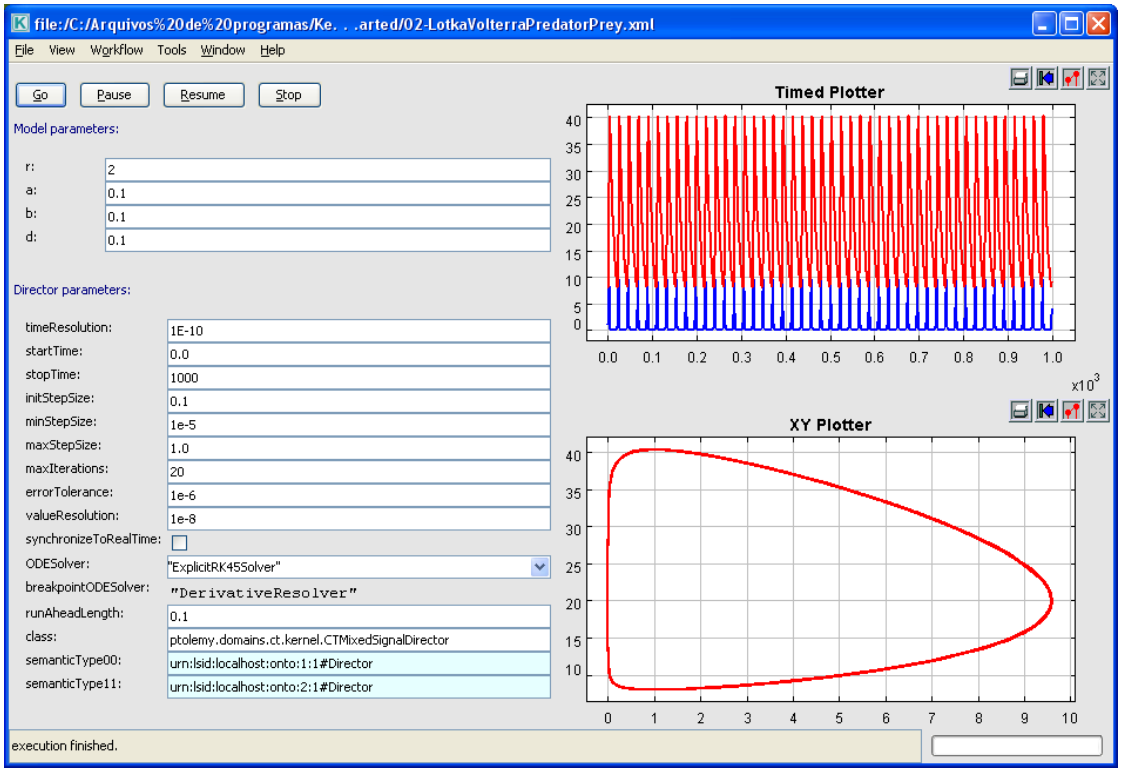

Figura 4.4: Janela de acompanhamento da execução do *workflow* no sistema Kepler. Extraído da referência [46].

Sistema *workflow*	
Categoria	**Kepler suporta**
Projeto	✓
Execução	✓
Proveniência	em desenvolvimento

Tabela 4.1: Principais categorias da taxonomia suportadas pelo SWC Kepler.

jeto, todas as funcionalidades das subcategorias Estrutura, Modelos Computacionais, Tipos de Componentes e Extensões. Todavia, a subcategoria Composição é suportada apenas parcialmente, como justificado abaixo.

A Estrutura no Kepler pode ser DAG ou não-DAG. Os Modelos Computacionais suportados são: tempo contínuo, fluxo de dados dinâmico, evento discreto e fluxo de dados síncronos. Os Tipos de Componentes por Domínio de Aplicação e Recursos estão presentes no sistema e também as Extensões por meio de tarefas externas e tarefas internas.

Projeto	
Subcategoria	**Kepler suporta**
Estrutura	✓
Composição	em parte
Modelos computacionais	✓
Tipos de componentes	✓
Extensões	✓

Tabela 4.2: Subcategorias da categoria projeto da taxonomia que o sistema Kepler suporta.

Para a subcategoria Tipos de Componentes, Domínios de Aplicação, existem componentes para as áreas da computação, estatística, ecologia, biologia, oceanografia e várias outras. Quanto à subcategoria Recursos, existem componentes para Controle de Execução, Encapsulamento, Documentação do *workflow*, Tratamento de exceções, Execução de comandos do sistema operacional, entre outros.

Na subcategoria Extensões, o Kepler fornece suporte às tarefas externas com *script*s Python®, linguagens R® , Matlab® e serviços web, além de tarefas internas por meio de criação de componentes.

Na subcategoria Composição é suportado apenas o tipo de composição direcionado pelo usuário. O Kepler não tem suporte para composição automática de *workflows*. Nesse sentido existe apenas um auxílio à busca de componentes baseada em palavras chave e anotações semânticas.

Na categoria Execução, como resumido na Tabela 4.3, o Kepler suporta as subcategorias Depuração e Monitoração da execução. Apesar de fazer tratamento de exceções com alertas e mensagens detalhadas do erro ocorrido, não há componentes para redirecionar a execução do *workflow* por outro caminho.

Em relação à subcategoria Depuração, o Kepler provê funcionalidades que facilitam a busca por problemas e conferência dos resultados obtidos em cada ponto na execução de um modelo *workflow*. Dentre as funcionalidades de depuração podem-se citar: a execução animada, que destaca de vermelho o componente que está computando no momento; a mensagem de erro detalhada, que informa em qual linha do

Execução	
Subcategoria	**Kepler suporta**
Depuração	✓
Tolerância a falha	
Monitoração da execução	✓

Tabela 4.3: Subcategorias da categoria execução da taxonomia que o sistema Kepler suporta.

código o erro ocorreu e a mensagem correspondente; a verificação das configurações do sistema operacional e do Kepler; e o acompanhamento da execução do componente Diretor, que apresenta todas as atividades do modelo computacional durante a execução do *workflow*.

As funcionalidades disponíveis para depuração são: a pausa, o retomar e a parada, que interagem diretamente com o fluxo de execução do *workflow*, e auxiliam no acompanhamento dos valores que as variáveis recebem durante a computação.

O Kepler disponibiliza componentes para a subcategoria Monitoração da Execução que auxiliam o usuário a acompanhar as tarefas do *workflow*. Esses componentes, discutidos na Seção 2.3.3, são capazes de informar quantas vezes um componente é usado, a progressão da execução e a situação de um componente. Outra funcionalidade é a janela para acompanhamento da execução, apresentada na Seção 4.3, que auxilia no acompanhamento geral da execução e permite mudar de maneira fácil os parâmetros do *workflow*.

Capítulo 5

Modelagem computacional de sistemas dinâmicos deformáveis

Esse capítulo tem por objetivo dar uma visão geral sobre a modelagem numérica computacional de um sistema dinâmico deformável.

Inicia-se com a análise do problema clássico de 1 grau de liberdade (GL) modelado por um sistema massa-mola-amortecedor. A seguir, esta análise é generalizada para sistemas com vários GL visando modelar estruturas com maior nível de complexidade. Em seguida apresenta-se o método de superposição modal e a integração numérica das equações diferenciais de equilíbrio dinâmico através de uma discretização temporal via diferenças finitas.

Este capítulo resume os conceitos apresentados em alguns textos clássicos da literatura relativos à dinâmica de sistemas deformáveis. Para uma análise mais aprofundada recomenda-se a leitura das referências [51], [4] e [13].

5.1 Sistemas com um grau de liberdade

5.1.1 Componentes de um sistema dinâmico com um grau de liberdade

De uma forma geral, as propriedades físicas essenciais de um sistema mecânico sujeito a uma fonte externa de excitação ou carregamento dinâmico são: massa, propriedades elásticas e amortecimento ou perda de energia mecânica. No modelo mais simples de um sistema com 1 GL, assume-se que cada uma destas propriedades esteja concentrada em um único elemento físico. As Figuras 5.1 (a) e 5.1 (b) mostram um esquema deste sistema e um diagrama de corpo livre para esse modelo, respectivamente.

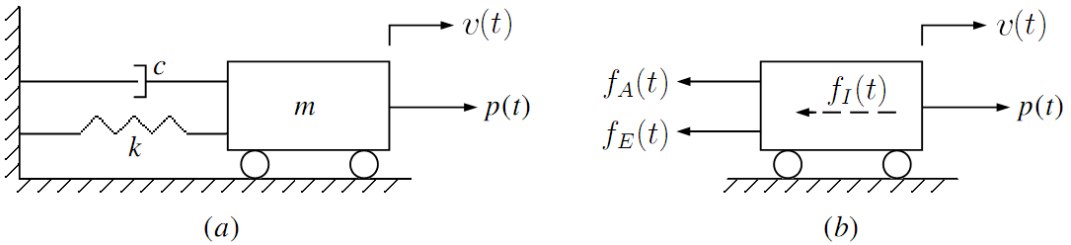

Figura 5.1: Sistema 1 GL - a) componentes básicos. b) diagrama de corpo livre.

Considera-se que a massa m do sistema esteja contida em um bloco rígido que pode se mover somente em translação simples. Assim, a coordenada de deslocamento $v(t)$ define completamente sua posição. A resistência elástica ao deslocamento é devida a uma mola de rigidez k e peso desprezível, enquanto o mecanismo de perda de energia (ou amortecimento) é representado pelo amortecedor c. O carregamento dinâmico externo, variante no tempo, é representado por $p(t)$.

A equação de movimento de um sistema como o mostrado na Figura 5.1 (a) é obtida diretamente a partir da expressão de equilíbrio de todas as forças atuantes no bloco mostradas no diagrama de corpo rígido da Figura 5.1 (b) e pode ser expressa como:

$$f_I(t) + f_A(t) + f_E(t) = p(t), \tag{5.1}$$

onde $f_I(t)$ representa a força inercial, $f_A(t)$ é a força dissipativa causada pelo amor-

tecimento, $f_E(t)$ é a força elástica e $p(t)$, o carregamento.

De acordo com o princípio de d'Alembert, a força inercial é obtida pelo produto da massa pela aceleração:

$$f_I(t) = m\ddot{v}(t), \tag{5.2}$$

onde $\ddot{v}(t)$ representa a derivada segunda temporal de $v(t)$.

Assumindo um mecanismo de amortecimento viscoso, a força de amortecimento pode ser representada como o produto da constante de amortecimento c pela velocidade:

$$f_A(t) = c\dot{v}(t), \tag{5.3}$$

onde $\dot{v}(t)$ representa a derivada primeira temporal de $v(t)$.

E, finalmente, considerando a força elástica como o produto da rigidez da mola pelo deslocamento, vem:

$$f_E(t) = kv(t). \tag{5.4}$$

Substituindo as Equações (5.2) e (5.4) na Equação (5.1), obtém-se a equação de movimento para um sistema de 1 GL representada pela equação diferencial ordinária de segunda ordem com coeficientes constantes:

$$m\ddot{v}(t) + c\dot{v}(t) + kv(t) = p(t). \tag{5.5}$$

5.1.2 Análise de vibrações livres

Um sistema dinâmico se encontra em vibrações livres quando o carregamento aplicado $p(t)$ é nulo. Assim, tem-se da Equação (5.5):

$$m\ddot{v}(t) + c\dot{v}(t) + kv(t) = 0. \tag{5.6}$$

Para sistemas com amortecimento subcrítico a solução da Equação (5.6) pode ser expressa como:

$$v(t) = \hat{v}\,\mathrm{sin}(\omega_D t + \theta)e^{-\xi\omega t}, \tag{5.7}$$

onde a amplitude $\hat{v}$ é dada por:

$$\hat{v} = \sqrt{[v(0)]^2 + \left[\frac{\dot{v}(0)}{\omega}\right]^2}; \tag{5.8}$$

sendo $v(0)$ e $\dot{v}(0)$ o deslocamento e a velocidade inicial, respectivamente; θ o ângulo de fase escrito como:

$$\theta = \tan^{-1}\left[\frac{-\dot{v}(0)}{\omega v(0)}\right]; \tag{5.9}$$

ω a freqüência angular definida como:

$$\omega = \sqrt{\frac{k}{m}}; \tag{5.10}$$

ω_D a freqüência de vibração livre amortecida dada por:

$$\omega_D = \omega\sqrt{1-\xi^2}; \tag{5.11}$$

e ξ taxa de amortecimento definida por:

$$\xi = \frac{c}{2m\omega}. \tag{5.12}$$

A Figura 5.2 mostra um esboço da resposta temporal de um sistema com 1 GL em vibrações livres com amortecimento subcrítico.

5.1.3 Análise de vibrações forçadas

A solução da equação diferencial (5.5) para o caso de vibrações forçadas depende obviamente da função que define a excitação $p(t)$. Em muitos casos essa solução só

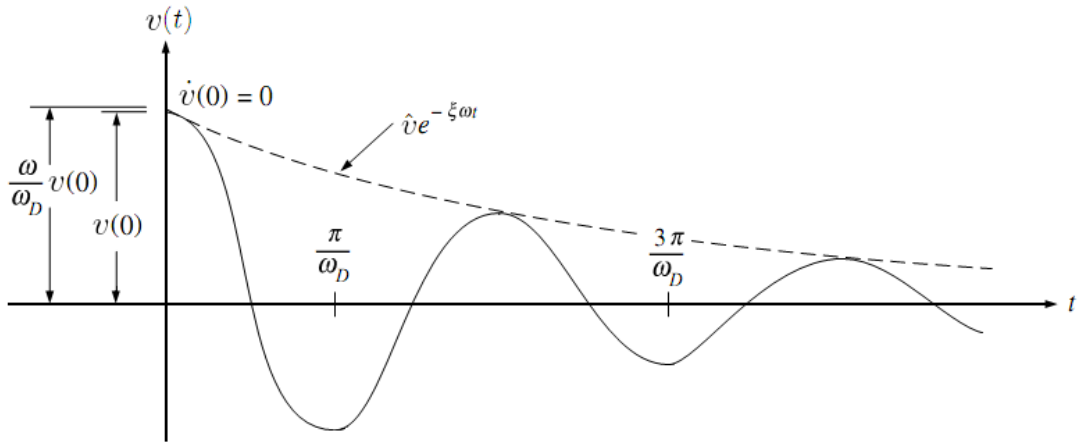

Figura 5.2: Resposta temporal em vibrações livres com amortecimento subcrítico para um sistema de 1 GL. Extraído da referência [51].

é possível de se obter numericamente. Apresenta-se na equação que segue, a título de ilustração, a resposta para uma excitação harmônica do tipo $p(t) = p_0 \sin(\overline{\omega}t)$:

$$\begin{aligned} v(t) &= \left[v(0)\cos(\omega_D t) + \dot{v}(0)\sin(\omega_D t)\right] e^{-\xi\omega t} \\ &\quad + \frac{p_0}{k}\left[\frac{1}{\left(1-\beta^2\right)^2 + \left(2\xi\beta\right)^2}\right]\left[\left(1-\beta^2\right)\sin(\overline{\omega}t) - 2\xi\beta\cos(\overline{\omega}t)\right], \end{aligned} \quad (5.13)$$

onde $\overline{\omega}$ é a frequência de excitação e β é dado por:

$$\beta = \frac{\overline{\omega}}{\omega}. \quad (5.14)$$

5.2 Sistemas com múltiplos graus de liberdade

As equações descritas na Seção 5.1.1 podem ser generalizadas para sistemas dinâmicos com vários GL. Seja a estrutura representada pela Figura 5.3, modelada por N GL.

As equações de movimento de sistemas com vários GL podem ser escritas expressando-se, de maneira similar aos sistemas de 1 GL, o equilíbrio de forças associadas a cada um dos GL. De forma geral, quatro tipos de forças atuarão em 1 GL: o carregamento externo aplicado $p_i(t)$ e as forças de inércia f_{Ii}, de amortecimento f_{Ai} e elástica f_{Ei}. Assim, para cada GL, as equações de equilíbrio se escrevem como:

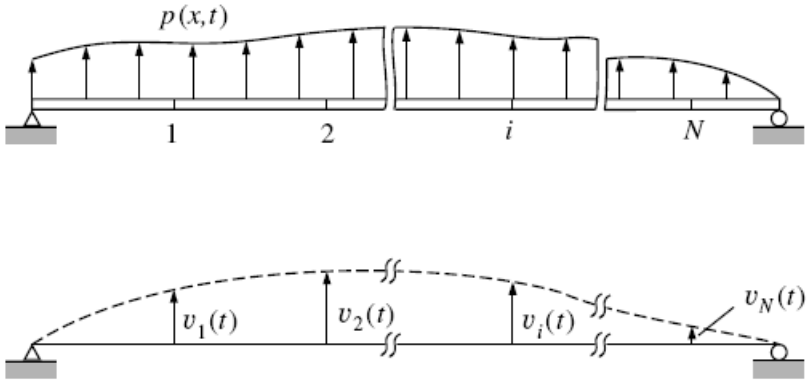

Figura 5.3: Discretização de uma estrutura de viga arbitrária. Extraído da referência [51].

$$\begin{aligned} f_{I1} + f_{A1} + f_{E1} &= p_1(t), \\ f_{I2} + f_{A2} + f_{E2} &= p_2(t), \\ &\vdots \\ f_{IN} + f_{AN} + f_{EN} &= p_N(t). \end{aligned} \tag{5.15}$$

Escrevendo as Equações (5.15) em notação vetorial,

$$\mathbf{f}_I(t) + \mathbf{f}_A(t) + \mathbf{f}_E(t) = \mathbf{p}(t), \tag{5.16}$$

tem-se a representação equivalente à Equação (5.1) para sistemas com vários GL.

Cada força interna pode ser escrita de maneira mais conveniente utilizando-se um conjunto de coeficientes de influência. Considerando, por exemplo, a componente da força elástica atuante no GL 1, pode-se mostrar que ela depende das componentes dos deslocamentos dos outros pontos da estrutura. Assim,

$$f_{E1} = k_{11}v_1 + k_{12}v_2 + \ldots + k_{1N}v_N. \tag{5.17}$$

Generalizando,

$$f_{Ei} = k_{i1}v_1 + k_{i2}v_2 + \ldots + k_{iN}v_N. \tag{5.18}$$

Nestas expressões assume-se que o sistema possui comportamento linear e que, portanto, o princípio da superposição de efeitos se aplica. Os coeficientes k_{ij} são denominados coeficientes de rigidez e são definidos como:

$$k_{ij} = \text{força correspondente ao GL } i \text{ devido a um deslocamento unitário no GL.}$$

Escrevendo estas expressões em forma matricial, vem:

$$\begin{Bmatrix} f_{E1} \\ f_{E2} \\ \vdots \\ f_{Ei} \\ \vdots \\ f_{EN} \end{Bmatrix} = \begin{bmatrix} k_{11} & k_{12} & \dots & k_{1i} & \dots & k_{1N} \\ k_{21} & k_{22} & \dots & k_{2i} & \dots & k_{2N} \\ \vdots & \vdots & \ddots & \vdots & \dots & \vdots \\ k_{i1} & k_{i2} & \dots & k_{ii} & \dots & k_{iN} \\ \vdots & \vdots & \dots & \vdots & \ddots & \vdots \\ k_{N1} & k_{N2} & \dots & k_{Ni} & \dots & k_{NN} \end{bmatrix} \begin{Bmatrix} v_1 \\ v_2 \\ \vdots \\ v_i \\ \vdots \\ v_N \end{Bmatrix} \tag{5.19}$$

ou simplesmente,

$$\mathbf{f}_E = \mathbf{K}\mathbf{v}, \tag{5.20}$$

onde a matriz de coeficientes de rigidez $\mathbf{K}$ é denominada matriz de rigidez do sistema; $\mathbf{v}$ representa o vetor de deslocamentos e $\mathbf{f}_E$ é o vetor de forças elásticas.

Desenvolvendo de maneira similar para a força de amortecimento ($\mathbf{f}_A$) e a força inercial ($\mathbf{f}_I$), tem-se:

$$\mathbf{f}_A = \mathbf{C}\dot{\mathbf{v}}, \tag{5.21}$$

onde a matriz de coeficientes de amortecimento $\mathbf{C}$ é denominada matriz de amortecimento do sistema, e

$$\mathbf{f}_I = \mathbf{M}\ddot{\mathbf{v}}, \tag{5.22}$$

onde a matriz de coeficientes de massa $\mathbf{M}$ é denominada matriz de massa do sistema.

Substituindo as Equações (5.20), (5.21) e (5.22) na Equação (5.16), obtém-se a equação matricial de equilíbrio dinâmico, considerando todos os GL do sistema:

$$\mathbf{M}\ddot{\mathbf{v}} + \mathbf{C}\dot{\mathbf{v}} + \mathbf{K}\mathbf{v} = \mathbf{p}(t). \tag{5.23}$$

Assim, a Equação (5.23) expressa as N equações diferenciais de movimento que definem a resposta de um sistema com vários GL e é a generalização da Equação (5.5).

A solução do sistema acoplado de N equações diferenciais de 2ª ordem definido pela Equação (5.23) pode ser feito aplicando-se o método de superposição modal. Neste caso o conjunto das N equações diferenciais é resolvido de forma desacoplada, o que reduz consideravelmente o esforço computacional a ser dispendido.

5.3 Método da superposição modal

O problema de autovalor descrito por:

$$|\mathbf{K} - \omega_j^2\mathbf{M}|\boldsymbol{\phi}_j = \mathbf{0}, \tag{5.24}$$

fornece N modos de vibrações $\boldsymbol{\phi}_j$ e frequências naturais dentre ω_j. Utilizando-se o Método da Superposição Modal [51], admite-se então que a resposta da estrutura é dada por:

$$\boldsymbol{v}(x,t) = \sum_{j=1}^{N} \boldsymbol{\phi}_j(x) y_j(t) \tag{5.25}$$

onde,

- $\boldsymbol{\phi}_j(x)$ são os modos naturais de vibração;
- $y_j(t)$ são coordenadas generalizadas.

Dependendo do caso tratado, as respostas dinâmicas do sistema podem ser obtidas com relativa precisão se tomarmos J ($J \leq N$) modos de vibração superpostos.

$$\boldsymbol{v}(x,t) \approx \sum_{j=1}^{J} \phi_j(x) y_j(t) \tag{5.26}$$

Necessita-se, então, determinar as coordenadas generalizadas ($y(t)$) para se obter a resposta dinâmica de um sistema estrutural utilizando-se o Método da Superposição Modal.

Substituindo-se a Equação (5.26) e suas derivadas na equação diferencial do movimento para sistemas discretos (5.23), utilizando-se as propriedades de ortogonalidade entre os modos de vibração, e considerando-se um amortecimento proporcional à massa e/ou à rigidez, chega-se ao sistema de equações diferenciais desacopladas:

$$\overline{\boldsymbol{M}}\ddot{\boldsymbol{y}} + \overline{\boldsymbol{C}}\dot{\boldsymbol{y}} + \overline{\boldsymbol{K}}\boldsymbol{y} = \overline{\boldsymbol{F}} \tag{5.27}$$

onde:

- $\overline{\boldsymbol{M}} = \boldsymbol{\Phi}^{\boldsymbol{T}}\boldsymbol{M}\boldsymbol{\Phi}$ é a matriz de massa generalizada (diagonal);
- $\overline{\boldsymbol{C}} = \boldsymbol{\Phi}^{\boldsymbol{T}}\boldsymbol{C}\boldsymbol{\Phi}$ é a matriz de amortecimento generalizada (diagonal);
- $\overline{\boldsymbol{K}} = \boldsymbol{\Phi}^{\boldsymbol{T}}\boldsymbol{K}\boldsymbol{\Phi}$ é a matriz de rigidez generalizada (diagonal);
- $\overline{\boldsymbol{F}} = \boldsymbol{\Phi}^{\boldsymbol{T}}\boldsymbol{F}$ é o vetor de forças generalizada.

Assim sendo, para cada j-ésimo modo de vibração tem-se a equação modal:

$$m_j\ddot{y}_j + c_j\dot{y}_j + k_j y_j = f_j \tag{5.28}$$

onde,

- m_j, c_j *e* k_j são respectivamente a massa, o amortecimento e a rigidez modal;
- f_j é a força de excitação modal.

Resolvidas as Equações (5.28), e voltando a Equação (5.26), tem-se a resposta do sistema pelo método da superposição modal.

Pode-se dizer então que a solução de um problema de dinâmica de corpos formáveis resolvido com a aplicação do método da superposição modal recai na solução de N equações diferenciais de 2ª ordem idênticas à Equação (5.5). De uma forma geral, a solução destas equações requer um método numérico de integração. Nesse texto apresenta-se resumidamente a solução da equação diferencial de equilíbrio dinâmico através de uma discretização temporal feita via diferença finitas.

5.4 Integração numérica das equações diferenciais de equilíbrio dinâmico

5.4.1 Método das diferenças finitas

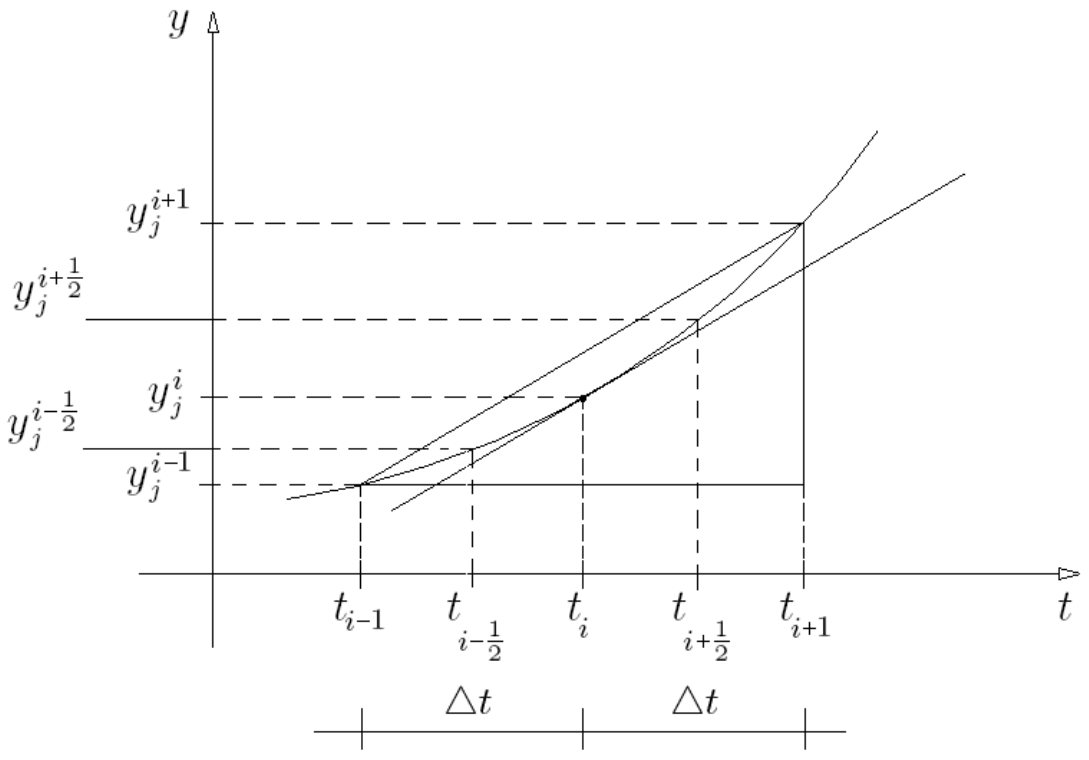

Figura 5.4: Exemplo de gráfico de deslocamento.

A partir da Figura 5.4, considerem-se os deslocamentos y_j^{i-1}, y_j^i e y_j^{i+1} referentes, respectivamente, aos tempos t_{i-1}, t_i e t_{i+1} do j-ésimo deslocamento modal separados entre si por um intervalo de tempo $\triangle t$. Para o tempo intermediário entre t_{i-1} e t_i

avaliação linear da velocidade pode ser estabelecida por:

$$\dot{y}_j^{i-\frac{1}{2}} = \frac{y_j^i - y_j^{i-1}}{\triangle t}, \tag{5.29}$$

enquanto que para o tempo intermediário entre t_i e t_{i+1} , a avaliação linear velocidade será dada por:

$$\dot{y}_j^{i+\frac{1}{2}} = \frac{y_j^{i+1} - y_j^i}{\triangle t}. \tag{5.30}$$

Os vetores velocidade obtidos pelas expressões (5.29) e (5.30) permitem que se obtenha uma avaliação linear do vetor aceleração $\ddot{y}_j^i$ que será dada por:

$$\ddot{y}_j^i = \frac{\dot{y}_j^{i+\frac{1}{2}} - \dot{y}_j^{i-\frac{1}{2}}}{\triangle t}. \tag{5.31}$$

Substituindo a Equação (5.29) e a Equação (5.30) na Equação (5.31), obtém-se uma aproximação do valor da aceleração no tempo t_i:

$$\ddot{y}_j^i = \frac{1}{\triangle t^2}\left(y_j^{i+1} - 2y_j^i + y_j^{i-1}\right). \tag{5.32}$$

Ao fazer a média da Equação 5.29 e Equação 5.30, obtém-se uma avaliação da velocidade no tempo t_i:

$$\dot{y}_j^i = \frac{\dot{y}_j^{i-\frac{1}{2}} + \dot{y}_j^{i+\frac{1}{2}}}{2} = \frac{y_j^{i+1} - y_j^{i-1}}{2\triangle t}. \tag{5.33}$$

Tem-se da Equação (5.28) que:

$$m\ddot{y}_j^i + c\dot{y}_j^i + ky_j^i = f^i, \tag{5.34}$$

para a qual se pode substituir $\ddot{y}_j^i$ e $\dot{y}_j^i$ e pelas suas expressões (5.32) e a (5.33), obtendo-se:

$$m\left[\frac{1}{\triangle t^2}(y_j^{i+1} - 2y_j^i + y_j^{i-1})\right] + c\left(\frac{y_j^{i+1} - y_j^{i-1}}{2\triangle t}\right) + ky_j^i = f^i. \tag{5.35}$$

Isolando-se, na expressão (5.35), os termos em y_j^{i+1}, obtém-se:

$$\left[\frac{1}{\triangle t^2}m + \frac{1}{2\triangle t}c\right] y_j^{i+1} = f^i - \left[k - \frac{2}{\triangle t^2}m\right] y_j^i - \left[\frac{1}{\triangle t^2}m - \frac{1}{2\triangle t}c\right] y_j^{i-1}. \quad (5.36)$$

Uma vez conhecido os valores do deslocamento para y_j^i e y_j^{i-1} (condições iniciais), fica possível se determinar o valor de y_j^{i+1}. Assim, num procedimento incremental, determina-se o histórico temporal de $y_j(t)$. O método de integração aqui apresentado insere-se na família de métodos definidos por explícitos, uma vez que a determinação de y_j^i depente explicitamente de avaliação de y_j em tempo anterior (y_j^{i-1} e y_j^{i-2}).

Desenvolvendo a integração numérica para cada j-ésimo modo de vibração e de posse do autovetores $\boldsymbol{\phi}_j$ (modos naturais de vibração obtidos pela solução do problema de autovalor descrito pela equação 5.24), pode-se voltar à equação 5.26 para o cálculo dos deslocamentos modais $v(x,t)$.

Capítulo 6

Estudo de caso

O estudo de caso proposto consiste na criação de MWCs no domínio de Dinâmica de Corpos Deformáveis.

Analisa-se nesta seção o comportamento dinâmico de um modelo de viga bi-apoiada, onde deslocamentos verticais, formas modais e freqüências naturais de vibração serão analisados.

Trata-se de um exemplo simples, cujo comportamento dinâmico é bastante conhecido pela comunidade científica. Entretanto, o objetivo desta análise é o de verificar as facilidades e dificuldades de se fazer avaliações de modelos dinâmicos com suporte de *workflows* científicos. Obviamente, estas avaliações poderão ser extendidas em trabalhos futuros a problemas dinâmicos mais complexos.

A Seção 6.1 descreve o problema que será estudado. A Seção 6.2 apresenta os MWCs propostos para resolver o problema. Os resultados obtidos a partir da execução dos MWCs são mostrados na Seção 6.3. Na Seção 6.4 é analisado o desempenho dos MWCs sugeridos. E, por último, a Seção 6.5 apresenta comentários sobre as facilidades e dificuldades de extensão do sistema Kepler.

6.1 Descrição do problema

Seja o modelo de uma viga bi-apoiada discretizada em elementos de pórtico plano descrito na Figura 6.1, sendo que a seção transversal da viga é mostrada na

Figura 6.2.

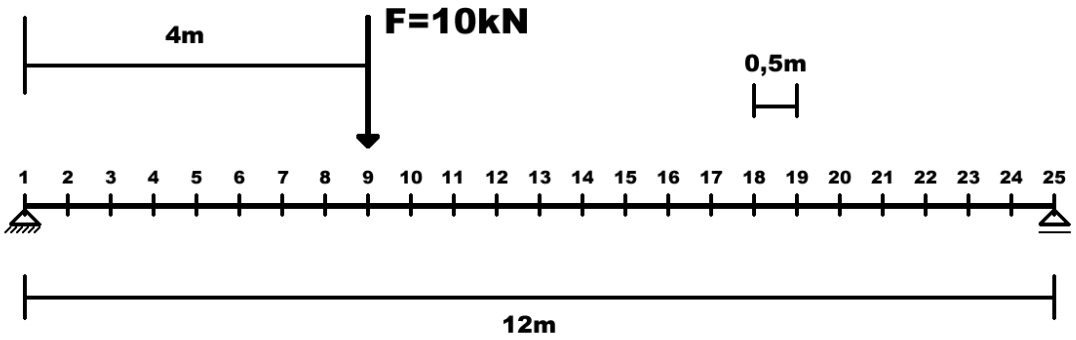

Figura 6.1: Modelo de uma viga bi-apoiada.

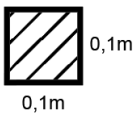

Figura 6.2: Seção transversal da viga bi-apoiada.

Os dados desse modelo são definidos como:

$$\begin{aligned} &E = 210G\text{Pa}, \\ &I = 8,333e^{-6}\text{m}^4, \\ &\rho = 7850\text{kg/m}^3, \\ &L = 12\text{m e} \\ &A = 0,01\text{m}^2, \end{aligned} \tag{6.1}$$

onde,

- E é o módulo de elasticidade,
- I é o momento de inércia,
- ρ é a massa específica,
- L é a largura da viga bi-apoiada e
- A é a área da viga bi-apoiada.

Para esse modelo busca-se determinar:

– os deslocamentos vertificais nos pontos 4, 6, 13, 16 e 22;

– as formas modais de vibração[1] e

– as freqüências naturais de vibração.

As respostas dinâmicas do sistema foram modeladas com 3 GL modais sendo as taxas de amortecimento para cada modo iguais a 5%. A força aplicada no ponto 9 é de 10kN, constante no tempo. A discretização do tempo é dividida em 5000 passos de $0,0014$s com tempo total de 7 segundos.

Os resultados dinâmicos obtidos, como já destacados anteriormente, são mostrados na Seção 6.3.

6.2 Modelos de *workflows* científicos propostos

Foram propostos três MWCs para resolver o problema apresentado na Seção 6.1, que visam avaliar as dificuldades e facilidades do sistema Kepler.

O primeiro MWC foi feito com o componente `MatlabExpression`. O segundo foi feito para ilustrar o suporte à execução de serviços web. Por fim, o terceiro MWC composto ilustra a construção de um novo componente que, neste caso, faz a integração numérica diretamente no sistema Kepler.

Em todos os MWCs implementados, foi assumido que as matrizes de massa, rigidez, e amortecimento, bem como o vetor de forças do sistema são conhecidos. Desta forma, a inserção desses MWCs em problemas mais complexos fica simplificada, bastanto, por exemplo, o acoplamento de componentes que gerem essas matrizes (um componente de elemento finitos, por exemplo) ou simplesmente a passagem desses dados advindos de outras plataformas de desenvolvimento ou *softwares*.

Assim, as tarefas necessárias para resolução desse experimento, e que devem ser implementadas no MWC, são:

– ler as matrizes de massa, de rigidez e de amortecimento do modelo;

[1] Utiliza-se nesse trabalho o algoritmo de Jacob[44] para determinação de auto-valores (freqüências naturais) e auto-vetores (formas modais de vibração).

– ler o vetor de forças aplicadas;

– calcular as formas modais de vibração, as freqüências naturais de vibração e os deslocamentos verticais;

– apresentar os resultados: deslocamentos verticais, por meio de gráficos; formas modais de vibração e freqüências naturais de vibração.

Entre as tarefas citadas, pode ser necessário adicionar novas tarefas para conversão do formato dos dados.

6.2.1 Modelo de *workflow* científico com componente MatlabExpression

Esse MWC foi construído de forma que a integração numérica das equações modais diferenciais de equilíbrio dinâmico tenha sido feita utilizando um código do sistema Matlab®. O Matlab® é um *software* interativo de alto desempenho voltado para o cálculo numérico e é executado pelo componente `MatlabExpression` do sistema Kepler, que é um exemplo de mecanismo de extensão dos SWCs.

O componente `MatlabExpression` original tinha algumas limitações nos dados de entrada e saída, pois não suportava matrizes, reconhecendo apenas vetores e tipos de dados básicos, tais como texto, valores lógicos e números. Assim, foi necessário alterar o código fonte do componente para que trabalhasse com as matrizes de massa, de rigidez e de força.

A Figura 6.3 mostra o MWC composto para execução desse experimento, que segue todos os passos descritos anteriormente de tarefas necessárias para resolução do mesmo.

O modelo computacional usado nesse MWC é o fluxo de dados síncronos, discutido na Seção 2.2.3, pois trata-se de um modelo sequencial simples, onde é preciso assegurar a execução correta das tarefas. Apesar da computação do deslocamento vertical depender de um parâmetro que varie no tempo, essa necessidade está embutida no componente responsável pela computação da integração numérica.

Os componentes criados ou adaptados para composição desse MWC, ilustrados

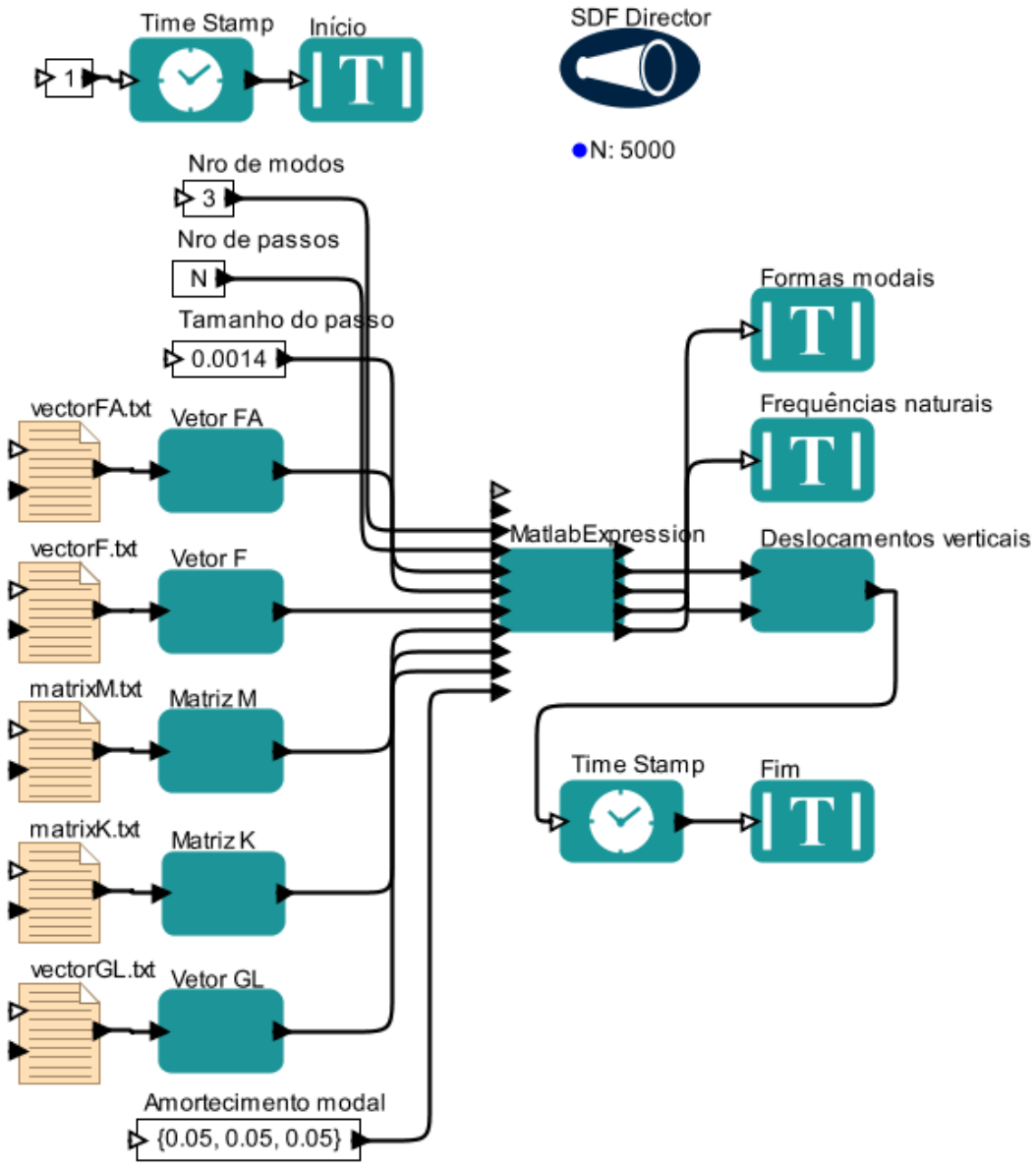

Figura 6.3: Modelo de *workflow* que utiliza componente MatLabExpression com *script* feito em Matlab® para fazer a integração numérica.

na Figura 6.3, são: o `Vetor`, a `Matriz`, o `MatLabExpression` e o `Deslocamentos verticais`.

Os componentes que tem um nome de arquivo como descrição são responsáveis pela leitura dos dados de massa (`matrixM.txt`), de rigidez (`matrixK.txt`), de intensidade da força (`vectorF.txt`), de força aplicada (`vectorFA.txt`) e de GL (`vectorGL.txt`) a partir de arquivos armazenados no disco.

Esses dados tem um formato de armazenamento simples, que visa tratar o pro-

blema de matrizes esparsas, onde cada linha tem a informação da posição do elemento e o seu respectivo valor, como no exemplo a seguir:

```
(7,3)  9.3643200e+008
(1,4)  1.5607200e+008
(4,4)  6.2428800e+008
...
```

Os componentes `Matrizes` e `Vetores`, criados pelo autor, são responsáveis por converter os dados lidos dos arquivos e passá-los para um formato conhecido pelo sistema Kepler. O código fonte do primeiro componente é apresentado no Apêndice A.2.

Os parâmetros numéricos de amortecimento modal são inseridos no MWC utilizando-se os componentes *Expression* ou *Constant*, que são componentes que computam expressões aritméticas ou que convertem texto para números, vetores ou matrizes, respectivamente. O amortecimento foi tomado proporcional à massa. Desta forma, a matriz de amortecimento fica definida com 3 constantes de proporcionalidade.

O componente `MatlabExpression` apresentado no MWC foi adaptado pelo autor para reconhecer as matrizes e vetores do experimento e tornar possível a execução do mesmo. Esse componente é reponsável por fazer a integração com o *software* Matlab®. Outra melhoria foi a inclusão de textos de depuração para serem visualizados na interface gráfica de acompanhamento da computação de um componente, que normalmente registram o que este está sendo executado.

As alterações feitas no código fonte desse componente, ilustrada no Apêndice A.1, são comentadas a seguir:

- a função fire (), linha 1, foi modificada para aprimorar a execução e a depuração;
- a principal modificação na função buildScript (), linha 134, foi na gravação de um arquivo com o algoritmo do Matlab® ao invés de passá-lo direto como parâmetro;

– a função parserOutput(), linha 174, teve que ser reescrita para reconhecer os resultados do tipo matriz, além de continuar reconhecendo os outros tipos de dados;

– a função getTokenValue(), linha 324, foi alterada para reconhecer matriz; e

– a função setOutputToken(String portName, String[][] value), linha 356, foi criada para armazenar uma matriz num dado de saída.

A Seção 6.5.1 ilustra a criação de um ambiente para edição desses componentes.

Os dados do componente `MatlabExpression` para esse experimento, apresentados na Figura 6.4, são compostos pelas seguintes propriedades:

expression contém o código do Mablab® a ser executado. Esse código é ilustrado no Algoritmo 6.1;

Matlab Executable contém o caminho e o nome do arquivo executável do *software*;

class contém o nome interno que identifica a tarefa no sistema Kepler e

semanticType contém uma descrição semântica utilizada para classificação do componente e apresentação do mesmo na paleta do sistema.

```
[xx Ms] = size(M);
[xx FAs] = size(FA);
if FAs < Ms;
  FA = [ FA zeros(1, Ms - FAs) ];
end;
FA = FA';
xi = xi';
[phi, w2] = eigs(K,M,nmodo,'sm');
[w2,aux] = sort(diag(w2));
for i=1:nmodo
  phiaux(:,i)=phi(:,aux(i),:);
end
phi=phiaux;
w=sqrt(w2);
Fm = phi'*FA;
```

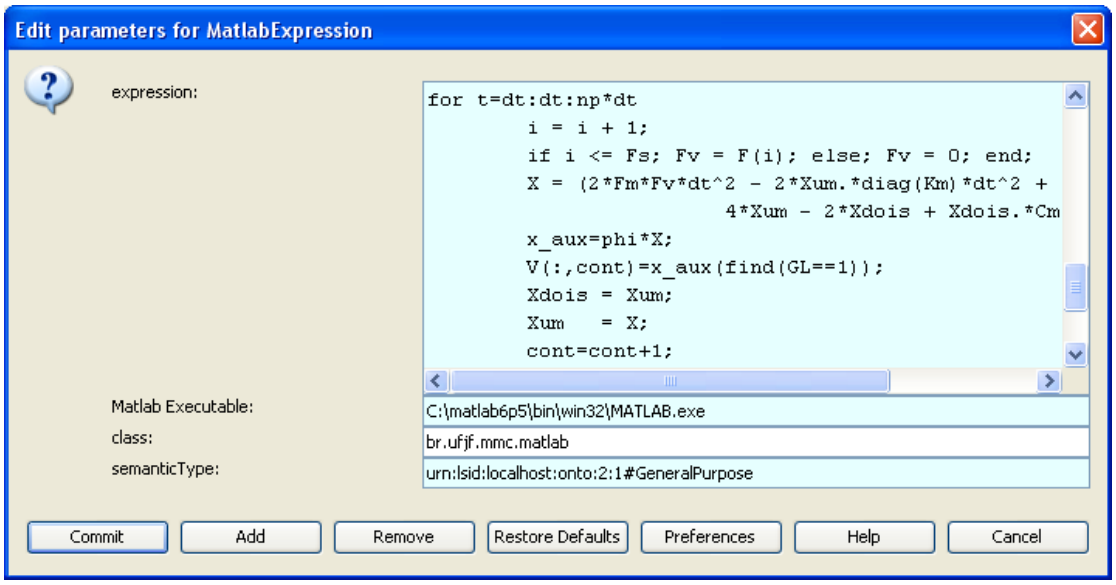

Figura 6.4: Parâmetros passados para o componente MatLabExpression para fazer a integração numérica.

```
Mm = phi'*M*phi;
Km = phi'*K*phi;
Cm = w*2.*xi(1:nmodo,:);
[xx Fs] = size(F);
Xum(1:nmodo,1)  = 0;
Xdois(1:nmodo,1)= 0;
cont=1;
for t=dt:dt:np*dt
  if cont <= Fs; Fv = F(cont); else; Fv = 0; end;
  X = (2*Fm*Fv*dt^2 - 2*Xum.*diag(Km)*dt^2 + ...
       4*Xum - 2*Xdois + Xdois.*Cm*dt) ./(2+Cm*dt);
  x_aux=phi*X;
  V(:,cont)=x_aux(find(GL==1));
  Xdois = Xum;
  Xum   = X;
  cont=cont+1;
end
X=dt:dt:np*dt;
X = X';
V = V';
```

Algoritmo 6.1: Código fonte Matlab® utilizado no componente *MatlabExpression*.

O código fonte Matlab® utilizado nesse experimento foi adaptado do algoritmo desenvolvido no curso de Dinâmica dos Corpos Deformáveis, cursado pelo autor,

oferecido pelo professor Flávio Barbosa do Mestrado em modelagem computacional da UFJF.

A Figura 6.5 ilustra a configuração dos dados de entrada e saída de um componente no sistema Kepler. No exemplo, os dados *expresssion*, *output* e *triggerSwitch* são dados de entrada e saída padrão do componente; e os dados `nmodo`, `np`, `dt`, `FA`, `F`, `M`, `K`, `GL`, `xi`, `X`, `V`, `phi` e `w` são dados de entrada e saída, adicionados pelo autor, para serem utilizados na interação com o *software* externo Matlab®. Esses dados podem ser adicionados ou removidos de acordo com a necessidade do componente, não havendo limite estipulado pelo sistema Kepler. Além de adicionar os dados de entrada e saída é preciso definir as seguintes propriedades:

Input indica que é dado de entrada;

Output indica que é dado de saída;

Multiport indica que o dado a receber ou a enviar pode ser passado em várias partes;

Type indica a tipagem do dado;

Direction informa qual é a direção do desenho do triângulo que representa o dado no componente (norte, sul, leste e oeste);

Show Name informa que é para mostrar o nome do dado no triângulo que o representa no componente;

Hide informa que é para esconder o triângulo que representa o dado no componente;

Units indica qual é a unidade de medida do dado (ampere, centímetro, grama, etc).

Esses dados de entrada são adicionados no algoritmo como variáveis e estarão disponíveis para serem utilizados. Por outro lado, espera-se que existam variáveis com os mesmos nomes dos dados de saída.

Os componentes `Deslocamentos verticais`, `Formas modais` e `Freqüências naturais` são responsáveis pelo pós-processamento, ou seja, pela apresentação dos resultados obtidos da computação dos dados do experimento. O componente `Deslo-`

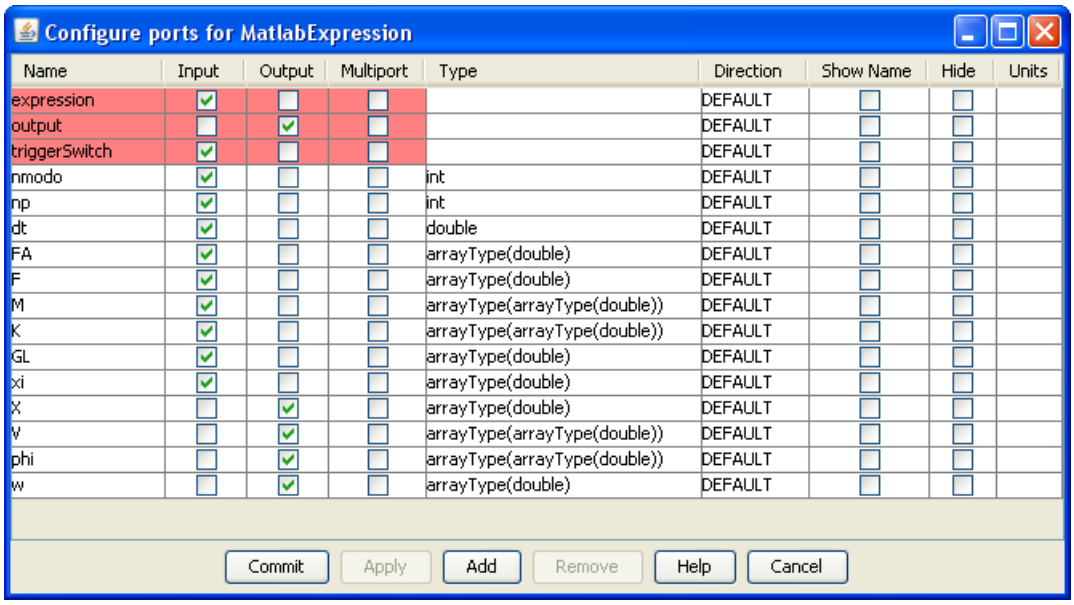

Figura 6.5: Configuração de dados de entrada e saída do componente MatlabExpression.

`camentos verticais` foi adaptado para desenhar os gráficos a partir de vetores e matrizes com o objetivo de otimizar a apresentação do resultado, pois os componentes padrões do sistema Kepler trabalham apenas com valores numéricos simples. Os componentes `Formas modais` e `Freqüências naturais` apresentam valores numéricos dos modos e freqüências de vibração, respectivamente.

As seguintes mudanças, ilustradas no Apêndice A.3, foram realizadas no componente `Deslocamentos verticais`:

- a função postfire () foi removida;
- a função fire (), foi adicionada para desenhar o gráfico de acordo com as matrizes passadas como argumentos.

O tempo de início e de fim da execução do MWC são recuperados a partir dos componentes `Time Stamp`, `Início` e `Fim`. Esses dados são utilizados na Seção 6.4, que trata da análise do desempenho dos MWCs.

O Matlab® é um *software* comercial e não faz parte do pacote de distribuição do projeto Kepler. Assim, para se fazer uso do componente `MatlabExpression` é preciso instalar esse *software* na máquina.

6.2.2 Modelo de *workflow* científico com componente *Web Service*

Esse MWC utiliza um serviço web para fazer a computação da integração numérica dos deslocamentos verticais, além do cálculo das formas modais e das freqüências naturais de vibração. Esse serviço é acionado pelo Kepler por meio de um cliente genérico de serviços web chamado `Web Service Actor`.

O serviço web foi escrito em Java® [27, 28] e adicionado ao *framework* Axis® [2], que é uma implementação do SOAP. O Axis® automatiza e facilita todo o processo de criação de um serviço web. Suas funcionalidades são disponibilizadas para uso através do sistema Tomcat®[3], que é um servidor de aplicações Java®.

Um exemplo de um serviço web disponibilizado pelo Axis® é ilustrado na Figura 6.6. Ao usar esse *software*, o usuário precisa criar uma classe Java® para que todo o processo para descrever e publicar um serviço web seja feito automaticamente. Os passos necessários para criar um novo serviço no Axis® são listados a seguir:

- criar uma classe Java® e nomeá-la como `Calcular`, como ilustrado no Algoritmo 6.2;
- copiar o arquivo criado para o diretório de serviços do Axis® (`<Tomcat>\webapps\axis`) e as bibliotecas para o subdiretório `classes` (`<Axis>\WEB-INF\classes`);
- renomear a extensão do arquivo de `java` para `jws` (`Calcular.jws`).

O *software* Axis® é responsável por compilar e criar a descrição WSDL do serviço disponibilizado. Para visualizar a descrição basta acessar o endereço do serviço com o parâmetro `wsdl`, como por exemplo `http://www.meudominio.com.br/axis/Calcular.jws?wsdl`. O serviço web criado no exemplo fornece os serviços de soma e de subtração de dois números inteiros. Para mais detalhes sobre a criação de serviços web utilizando Axis®, recomenda-se a leitura da referência [48].

[2] Axis®: uma implementação do SOAP. `http://ws.apache.org/axis`

[3] Tomcat®: servidor de aplicações Java®. `http://tomcat.apache.org`

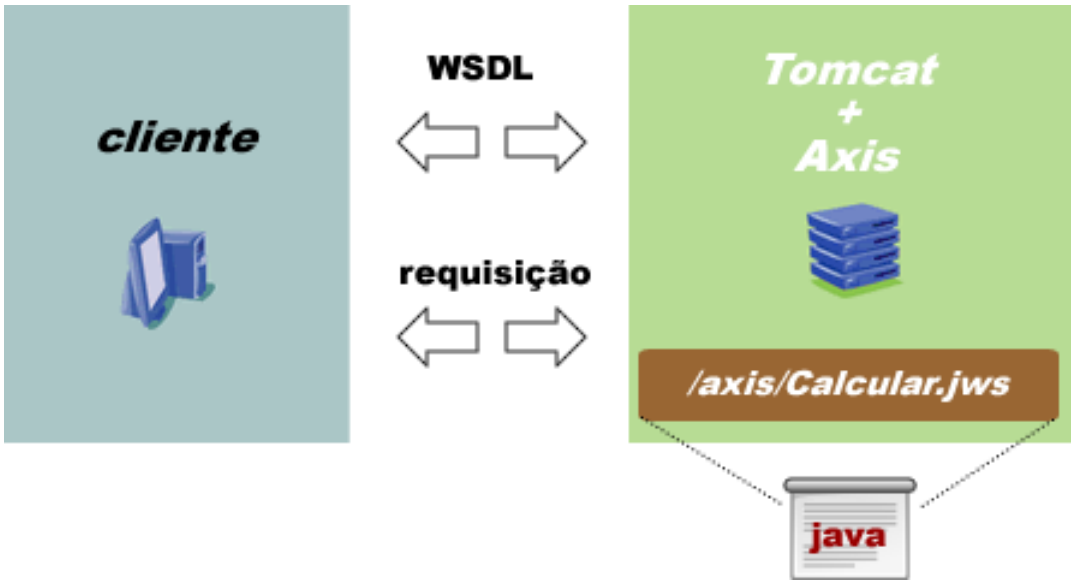

Figura 6.6: Arquitetura de um serviço web disponibilizado pelo Axis®.

```
public class Calcular {
  public int adicionar(int n1, int n2) {
    return n1 + n2;
  }
  public int subtrair(int n1, int n2) {
    return n1 - n2;
  }
}
```

Algoritmo 6.2: Exemplo de uma classe básica para criação de um serviço web no Axis®. Extraído da referência [48]

Para a implementação do algoritmo do serviço web do experimento, foi utilizada a biblioteca gratuita JAMA (*Java Matrix Package*)[4], que é desenvolvida pelo *National Institute of Standards and Technology* e pela Universidade of Maryland. Esta biblioteca é utilizada nos algoritmos de integração para manipulação de matrizes (soma, multiplicação, transposição, etc) e para a resolução do problema de autovalor (cálculo de autovalores e autovetores). Atualmente (Julho de 2008), a versão disponível desta biblioteca é a 1.0.2, possui código aberto e está em constante atualização.

[4]JAMA: biblioteca Java®para manipulação de matrizes e resolução do problea de autovalor. `http://math.nist.gov/javanumerics/jama`

O MWC composto no sistema Kepler utilizando o componente `Web Service Actor` é mostrado na Figura 6.7. Esse MWC segue todos os passos para a execução do experimento, como apresentados anteriormente na descrição do problema.

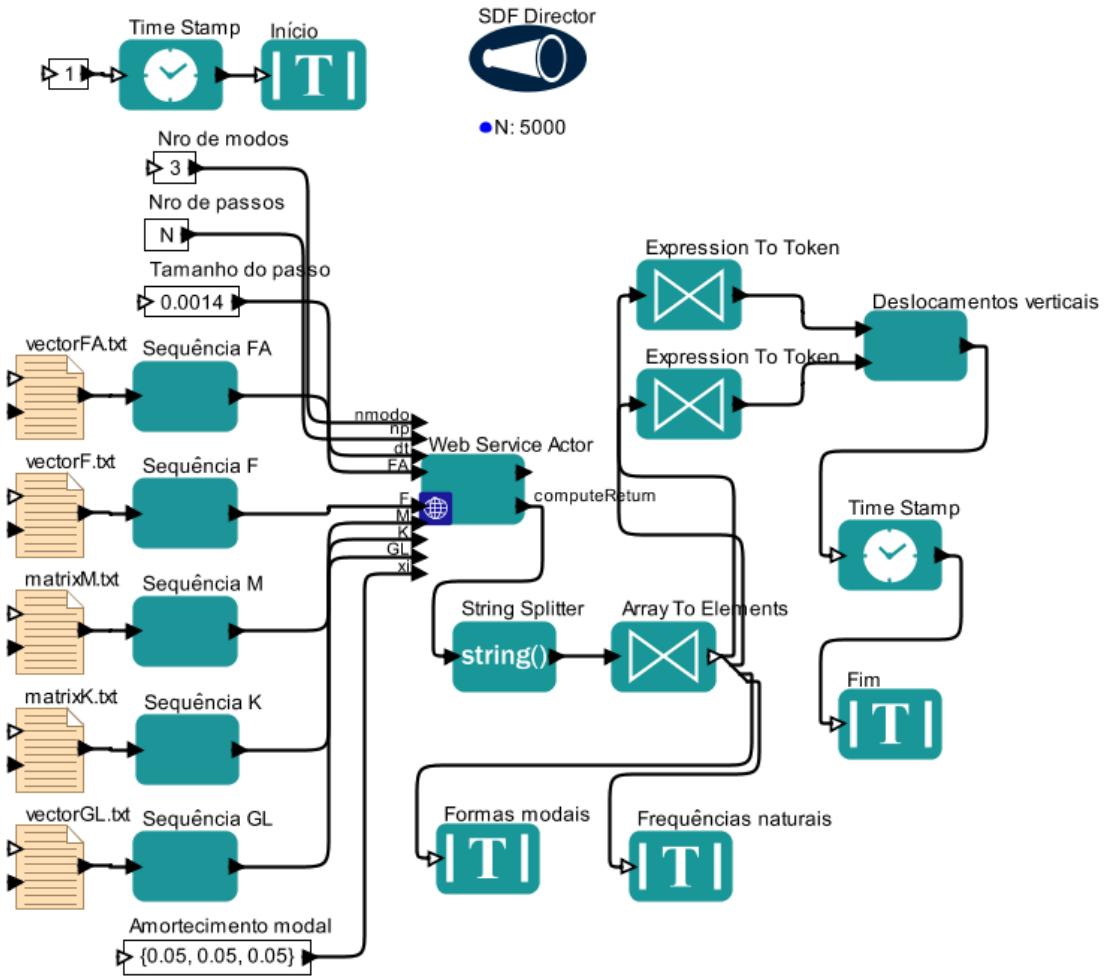

Figura 6.7: Modelo de *workflow* que utiliza componente *Web Service* para requisitar a integração numérica a partir de um serviço web.

Os componentes criados ou adaptados para composição desse MWC, ilustrados na Figura 6.7, são: a `Sequência` e o `Deslocamentos verticais`.

A leitura dos dados de massa, de rigidez, de intensidade da força, de força aplicada e de GL é feita a partir de arquivos, os mesmos apresentados na seção anterior, pelos componentes que tem o nome de arquivo como descrição.

Os componentes `Sequências`, criados pelo autor, armazenam nos seus dados de saída o texto em um dos formatos reconhecidos pelo componente `Web Service Actor`. O código fonte do componente `Sequência` é mostrado no Apêndice B.2.

Os componentes *Expression* e *Constant* são usados para armazenar no MWC os parâmetros numéricos (`Nro de modos`, `Nro de passos` e `Tamanho do passo`) e o vetor de amortecimento (`Amortecimento modal`).

O componente `Web Service Actor` é responsável por receber os dados de entrada, de fazer a requisição ao serviço web e de disponibilizar o resultado nos dados de saída. O código fonte do serviço web utilizado nesse MWC é apresentado no Apêndice B.1.

Como ilustrado na Figura 6.8, os dados de entrada e saída são bem parecidos com o MWC que utiliza o componente `MatlabExpression`. A diferença é que o serviço web retorna um único valor texto com todos os dados de saída e o componente `MatlabExpression` retorna seus dados separadamente.

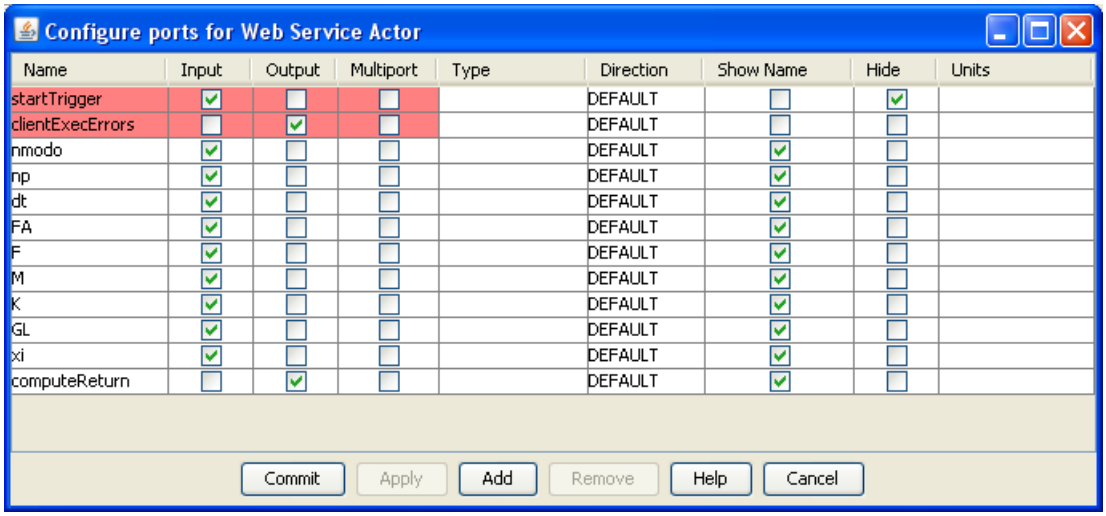

Figura 6.8: Configuração de dados de entrada e saída do componente *Web Service Actor*.

Pelo fato dos dados retornados estarem em um único valor texto, é necessário realizar a conversão dos dados de saída do formato texto para o formato vetores e matrizes numéricas. Os componentes responsáveis por essa conversão são o `String Splitter`, o `Array to Elements` e `Expression to Token` (Figura 6.7). O primeiro componente quebra o texto recebido em várias partes de acordo com um separador pré-determinado. O segundo recebe o vetor de textos e, por meio de um dado de saída do tipo *Multiport*, envia os textos em partes. Assim, as duas primeiras partes são enviadas para os componentes `Expression to Token` que convertem o texto em um vetor e uma matriz para os dados de entrada `X` e `Y`, respectivamente, do

componente `Deslocamentos verticais`. Desta forma, esse componente é capaz de apresentar os gráficos de acordo com os dados recebidos. A terceira parte do texto, por sua vez, é enviada para o componente `Formas modais`, que apresenta valores numéricos das formas modais. A quarta, e última parte, é enviada para o componente `Freqüências naturais`, que apresenta os valores numéricos das freqüências de vibração.

Os componentes `Time Stamp`, `Início` e `Fim` são utilizados para recuperar o tempo de início de fim da execução do MWC. Esses tempos são utilizados na Seção 6.4, que trata da análise do desempenho dos MWCs.

6.2.3 Modelo de *workflow* científico com novo componente de integração numérica

O último MWC composto para realizar a computação da integração numérica dos deslocamentos verticais, além do cálculo das formas modais e das freqüências naturais de vibração, incorporou um novo componente no sistema Kepler.

O MWC criado é apresentado na Figura 6.9, e a principal diferença, em relação ao primeiro MWC composto, está no componente que realiza a computação dos dados.

O novo componente, desenvolvido pelo autor, está identificado nesse exemplo como `Dynamics`. Além deste, os componentes `Vetor`, `Matriz` e `Deslocamentos verticais` foram também criados ou adaptados pelo autor deste trabalho.

A inclusão de novos componentes permite o aproveitamento de todos os recursos do sistema Kepler, além de evitar a execução desnecessária de componentes que necessitam interagir com aplicações externas, como é o caso dos componentes utilizados nos MWC anteriores. Na maioria das vezes, a execução ou o consumo de recursos computacionais será menor, tornando o MWC mais eficiente.

Como dito na Seção 6.2.1, os componentes que tem um nome de arquivo como descrição são responsáveis pela leitura dos dados de massa, de rigidez, de intensidade da força, de força aplicada e de GL a partir de arquivos. Os componentes `Vetores` e `Matrizes` convertem os dados lidos para um formato conhecido pelo

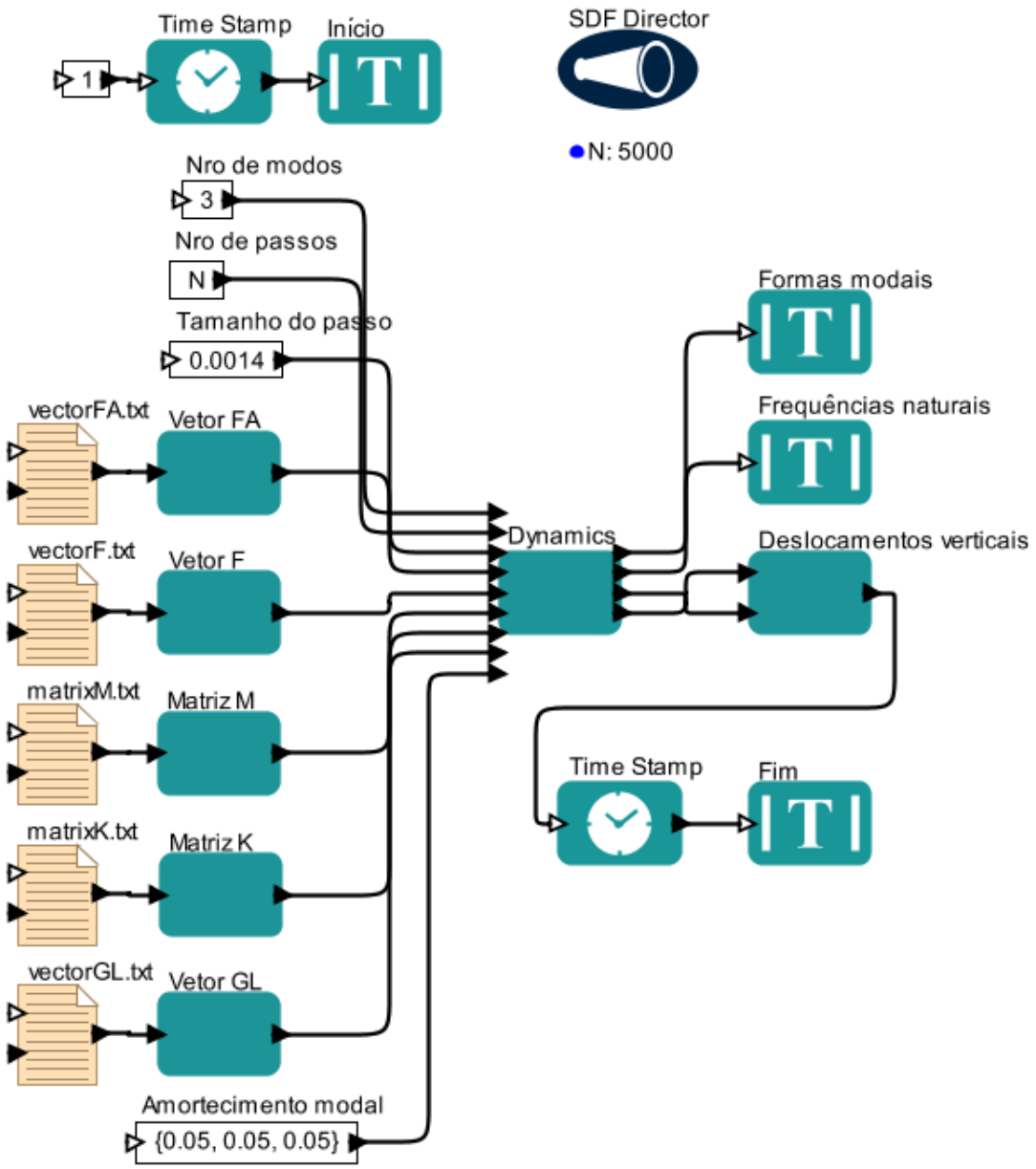

Figura 6.9: Modelo de *workflow* que utiliza novo componente que computa a integração numérica.

sistema Kepler. Os parâmetros numéricos e o vetor de amortecimento modal são inseridos no MWC utilizando-se os componentes *Expression* ou *Constant*. Os componentes `Deslocamentos verticais`, `Formas modais` e `Freqüências naturais` são responsáveis pela apresentação dos resultados obtidos da computação dos dados do experimento.

O tempo de início e de fim da execução do MWC são recuperados a partir dos componentes `Time Stamp`, `Início` e `Fim`. Esses dados são utilizados na análise do

desempenho dos MWCs, discutido na Seção 6.4.

A Figura 6.10 ilustra os dados de entrada e saída, que são iguais ao primeiro MWC que utiliza o componente `MatlabExpression`. A principal diferença dos dados de entrada e saída com relação aos primeiros MWCs apresentados está na flexibilidade das configurações de novos dados. Nesse componente os dados são fixos e obrigatórios, permitindo apenas a configurações dos mesmos, além dos dados obrigatórios do componente `MatlabExpression` não estarem presentes.

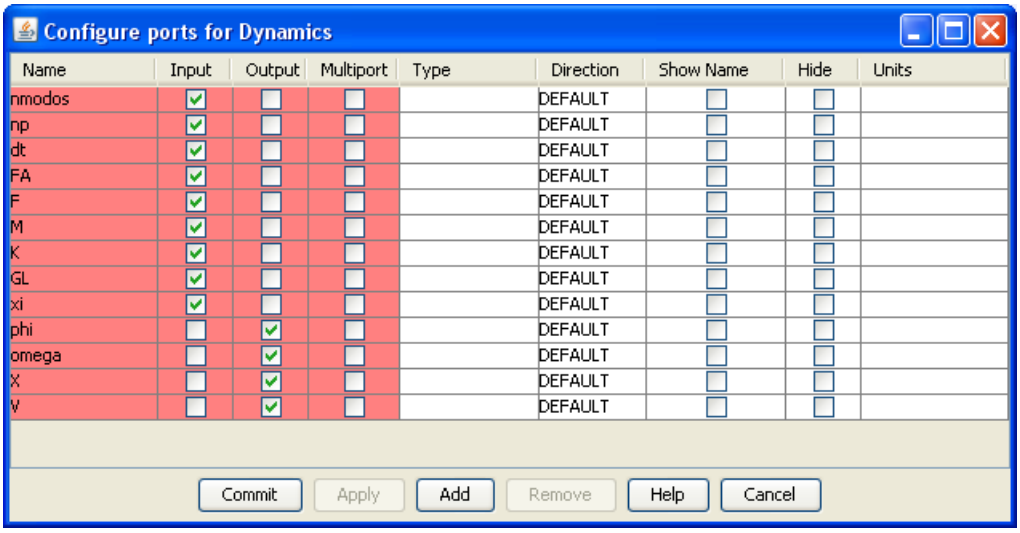

Figura 6.10: Configuração de dados de entrada e saída do componente *Dynamics*.

A dificuldade de um usuário avançado em criar um novo componente no sistema Kepler dependerá diretamente do nível de conhecimento da especialidade de sistemas computacionais, já que na referência [46] há documentação disponível com exemplo passo a passo que explica como criar um novo componente. Essa documentação facilita a primeira experiência desse desenvolvimento de um novo componente para o Kepler, pois fornece um esqueleto onde o usuário deve somente preencher os pontos que realizam a tarefa de fato. Além deste documento, também são encontrados outros, nas referências [18, 46], para a produção de componentes mais complexos, que faça uma interação maior com o sistema em questão.

O componente foi feito usando a linguagem Java®. Basicamente o código que faz a computação dos dados é o mesmo utilizado no serviço web, comentado na seção anterior (6.2.2). O código fonte do novo componente criado para esse experimento pode ser visualizado no Apêndice C.1.

A criação de um novo componente é feita a partir da especialização de uma classe disponível pelo sistema Kepler, com uma interface Java definida por esse sistema. Essa criação pode ser visualizada na Seção 6.5.1, que descreve também a preparação do ambiente de desenvolvimento para criação desse novo componente.

6.3 Resultados obtidos da análise dinâmica

A Figura 6.11 apresenta o gráfico com os resultados obtidos para os deslocamentos verticais dos pontos 4, 6, 13, 16 e 22. Os resultados de todos os MWCs apresentados neste trabalho são praticamente idênticos e em conformidade com os resultados teóricos esperados. Por esse motivo, apenas uma figura com os gráficos de deslocamentos é apresentada.

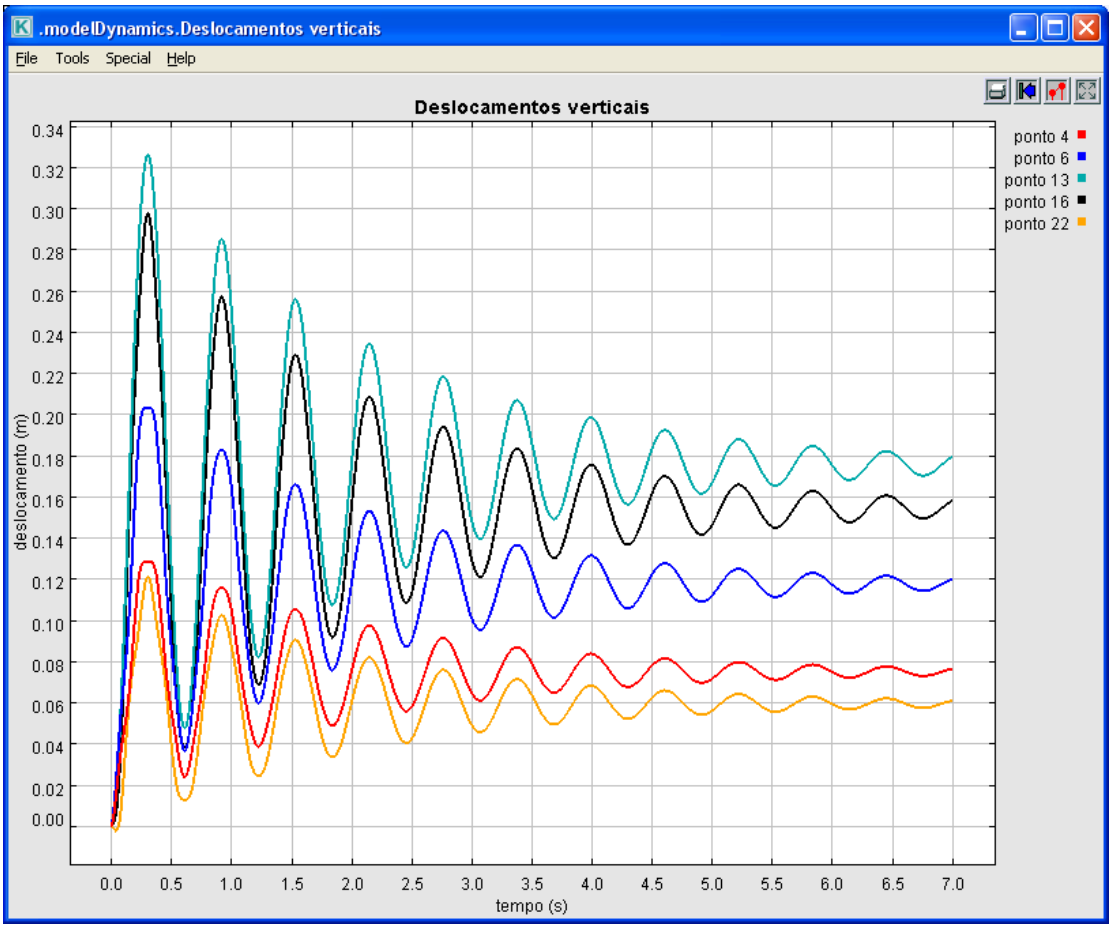

Figura 6.11: Resultado de deslocamento vertical esperado. Deslocamento em metros na vertical e tempo em segundos na horizontal (m/s).

A ferramenta de plotagem usada para gerar o gráfico apresentado na Figura 6.11 é muito útil para os usuários do sistema Kepler, pois além de plotar a figura ela ainda fornece funcionalidades para aumentar e diminuir o tamanho da imagem ou partes dela; criar legendas e títulos; exportar no formato eps; imprimir a imagem; e salvar e carregar a imagem plotada. Os componentes que utilizam essa ferramenta são todos aqueles que trabalham com plotagem, tais como: `Array Plotter`, `Sequence Plotter`, `Timed Plotter`, `XY Plotter`, entre outros.

A Figura 6.12 mostra os gráficos com as formas modais de vibração do deslocamento vertical apresentado na Figura 6.11. Da mesma forma que o deslocamento vertical, as formas modais também são similares em todos os MWCs apresentados e são idênticos aos resultados teóricos esperados.

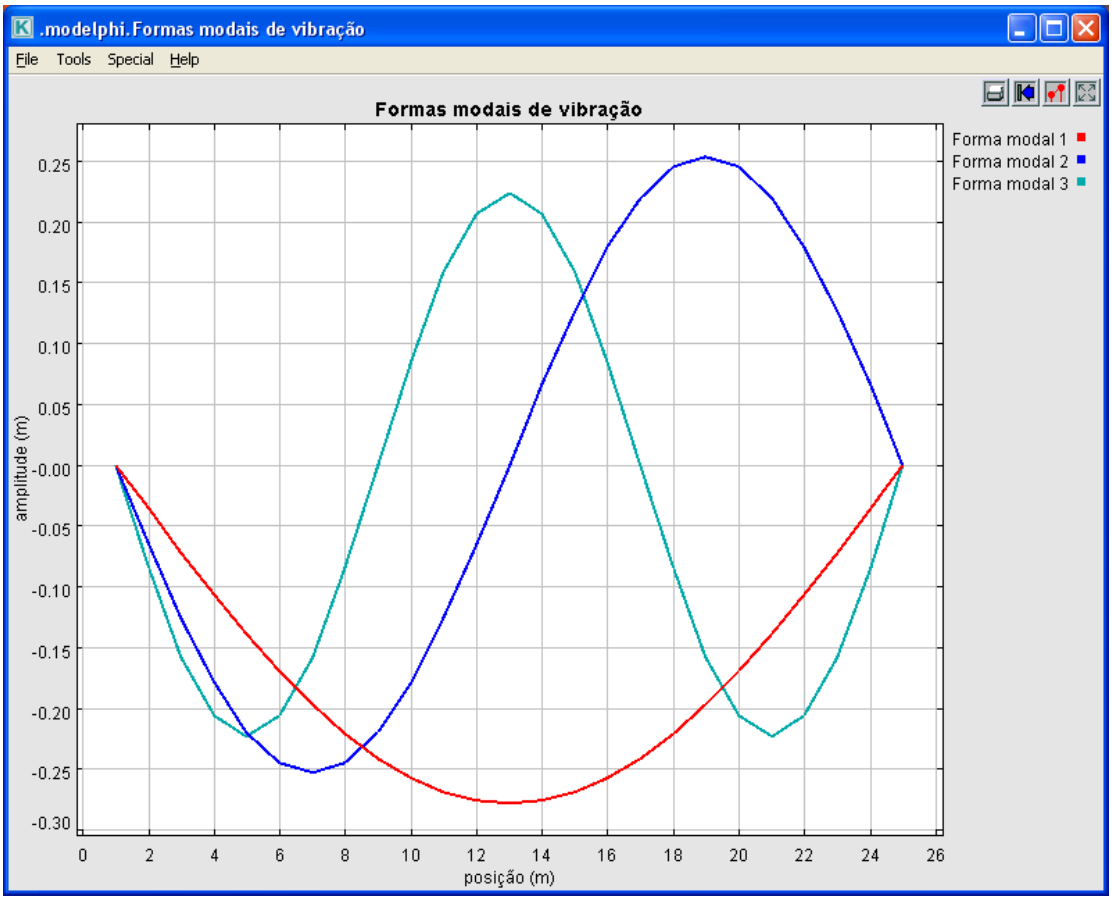

Figura 6.12: Resultado das formas modais de vibração esperadas.

As freqüências naturais de vibração obtidas para todos os MWC são apresentadas na Tabela 6.1 e comparadas com valores teóricos apontando uma excelente concordância.

Freqüência natural (rad/s)	**Freqüência natural teórica** (rad/s) [51]
10,233	10,233
40,934	40,934
92,102	92,103

Tabela 6.1: Freqüências naturais de vibração obtidas e teóricas.

6.4 Análise do desempenho

Esta seção apresenta o desempenho de cada MWC descrito na Seção 6.2. O desempenho foi medido cinco vezes, para somente depois utilizar a média como tempo resultante do MWC. Os tempos podem ser vistos nas Tabelas 6.2, 6.3 e 6.4.

Os tempos foram obtidos numa máquina com as seguintes configurações: processador AMD Athlon 64 X2 5200+, placa-mãe M2N-MX, memória RAM de 2GB DRR2-666. Para o experimento que faz uso do serviço web, o servidor foi a máquina citada e o MWC foi executado na máquina com as seguintes configurações: Notebook HP Pavilion dv6230BR, processador AMD Turion MK-36, memória RAM de 2GB DDR2.

Medição	**Tempo (s)**
1	31,453
2	30,985
3	30,906
4	31,391
5	30,424
Média	31,031

Tabela 6.2: Desempenho do modelo de *workflow* que utilizou o componente MatLabExpression

A Tabela 6.5 apresenta os dados de desempenho do código Matlab®, utilizado no MWC com o componente `MatlabExpression`, executado no sistema Matlab® para se utilizar como comparação.

Na Tabela 6.6 são apresentadas as médias dos desempenhos dos MWCs. Além

Medição	Tempo (s)
1	107,765
2	98,272
3	96,171
4	96,043
5	95,149
Média	98,680

Tabela 6.3: Desempenho do modelo de *workflow* que utilizou o componente *Web Service*

Medição	Tempo (s)
1	0,969
2	1,160
3	1,078
4	1,000
5	1,063
Média	1,054

Tabela 6.4: Desempenho do modelo de *workflow* que utilizou o novo componente

Medição	Tempo (s)
1	0,687
2	0,750
3	0,734
4	0,656
5	0,734
Média	0,712

Tabela 6.5: Desempenho do código no sistema MatLab®

das etapas necessárias para a execução dos MWCs ou do código Matlab®.

Dentre os MWCs do estudo de caso, o modelo com o novo componente é o mais

Tempo	31,031	98,680	1,054	0,712
Modelo	`MatlabExpression`	`Web Service Actor`	Novo Componente	MatLab®
Etapas	administração do MWC	administração do MWC	administração do MWC	
	conversão dos dados	conversão dos dados		
		tráfego		
	execução do Matlab®	execução do serviço Web (Java®)	computação dos dados (Java®)	execução do algoritmo
		tráfego		
	conversão dos dados	conversão dos dados		
	apresentação do resultado	apresentação do resultado	apresentação do resultado	apresentação do resultado

Tabela 6.6: Desempenho médio de todos os modelos e composição dos tempos dos experimentos.

rápido, pois a computação da tarefa é feita no próprio componente, diferentemente dos outros modelos, que necessitam de uma Tarefa Externa para computação dos dados.

O componente `MatLabExpression` aciona o sistema MatLab® para realizar a integração numérica dos deslocamentos verticais e calcular as formas modais e freqüências naturais de vibração. Para tal, os dados de entrada são convertidos do formato interno do sistema Kepler para a linguagem do sistema Matlab®, e os dados de saída são convertidos do texto para o formato interno do sistema Kepler. Comparando os tempos de $1,054$ do novo componente e de $0,712$ do MatLab®, pode-se supor que

grande parte do tempo de 31,031 segundos gastos para execução do MWC foram gastos nas conversões dos dados.

Assim, o uso do componente `MatlabExpression` em MWC no Kepler só é justificável quando a redução do tempo de desenvolvimento comparado a criação de código fonte na linguagem Java®, devido as facilidades de realizar cálculos numéricos do sistema MabLab®, com todas as suas "*toolboxes*" disponíveis, compensar a perda de desempenho na execução.

O pior desempenho, nesse estudo de caso, foi o MWC que utiliza o componente `Web Service Actor`. Esse desempenho se deve ao fato da necessidade de conversão dos dados; da transmissão via internet até o serviço web; e o caminho inverso. Como o código utilizado no serviço web é basicamente o mesmo utilizado no novo componente criado, pode-se concluir que o tempo de conversão e transmissão dos dados entre o Kepler e o serviço web demorou aproximadamente 97 segundos.

O uso do componente `Web Service Actor` é recomendado para tarefas prontas e disponíveis como serviços web, que aumentariam a eficiência na composição de MWCs. Ou ainda, em tarefas que necessitam de muitos recursos computacionais, tais como processamento e armazenamento, que possam ser executados em computadores de alto desempenho ou clusters.

Nesse experimento em particular, a execução do algoritmo do Matlab® é o mais rápido entre eles. Isso poderia suscitar questionamentos sobre a utilidade dos SWCs. Entretanto, é preciso considerar que o experimento apresentado não possui um número significativo de etapas, caso em que os benefícios do gerenciamento dessas etapas ficaria mais evidente.

6.5 Avaliação da facilidade de extensão do sistema de *workflow*

Nesta seção será descrito o esforço necessário para criação de cada MWC composto nesse estudo de caso. O objetivo é apontar as dificuldades encontradas e comentar algumas das facilidades disponíveis para os usuários de SWCs.

Um dos objetivos desse trabalho é avaliar a aplicabilidade de SWCs em um grupo de pesquisa onde um dos requisitos de interesse é a possibilidade de estender o ambiente de experimentação com novas funcionalidades desenvolvidas pelo próprio grupo. A extensão deve ser feita de tal forma que as novas funcionalidades adicionadas fiquem disponíveis para o reuso por outros pesquisadores. Isso aumentaria a eficiência do desenvolvimento de MWCs.

Assim, algumas das extensões desejáveis, implementadas nos MWCs apresentados, foram:

- a introdução de uma nova funcionalidade não suportada pelo SWC, como por exemplo, um novo método de integração numérica;
- a integração ao MWC de aplicações disponíveis como serviços web;
- a integração ao MWC de tarefas automatizadas para conversão de formato de dados;
- a integração de tarefas que necessitam de alto desempenho voltado para o cálculo numérico.

Outras extensões desejáveis interessantes não vistas neste trabalho são:

- a integração ao MWC de aplicações implementadas em Fortran [5] e C++ [6];
- a introdução de novos componentes para manipulação de matrizes e operações da álgebra linear;
- a construção de um visualizador da deformada das estruturas em tempo real;
- a introdução ao MWCs de tarefas que apliquem os mecanismos de processamentos distribuídos em ambientes Grid.

O primeiro MWC criado foi o que utiliza o componente `MatlabExpression`, e a principal dificuldade encontrada na construção desse modelo foi a criação de códigos que fossem aplicáveis a programas executados pela linha de comando, além

[5] Fortran: linguagem de programação que permite a criação de programas que prioriza a velocidade de execução.

[6] C++: linguagem de programação orientada a objetos, de alto nível, com facilidades para o uso em baixo nível.

da correção de seus erros. Entretanto, pode-se considerar que é de pouca complexidade criar um arquivo externo ao sistema Kepler, fazer vários testes e aplicar a técnica de *debug* antes de incluir o código fonte desenvolvido dentro do componente `MatlabExpression`. Outro problema encontrado foi a limitação de não trabalhar com matrizes, mas foi solucionado com a modificação do código fonte do componente original.

A grande vantagem de se usar o componente `MatlabExpression` está no aproveitamento de códigos criados no programa Matlab®. Caso o desempenho seja um requisito importante para a execução do MWC, esse componente poderia ser útil na criação de protótipos para avaliação das possibilidades de computação, e somente depois de concluir qual a melhor opção, criaria-se um componente em definitivo escrito em Java®.

Para a montagem do segundo MWC foi utilizado o componente `Web Service Actor`. Neste modelo foi utilizado o componente padrão do sistema Kepler, *Web Service Actor*, pois não houve necessidades além das funcionalidades disponíveis nesse componente. Entretanto, para a execução desse modelo, foi necessário transformar as matrizes e os vetores com valores numéricos *double* para o formato texto. Formato este utilizado para transmissão dos dados através do protocolo de comunicação do serviço web.

O uso de serviços web possibilita aproveitar todas as funcionalidades que essa tecnologia fornece, comentadas na Seção 3.3. Adicionalmente, é importante destacar também as inúmeras tarefas já disponíveis para uso na internet, como o reaproveitamento de código criado por outros pesquisadores ou instituições e a utilização de máquinas remotas com recursos computacionais melhores que a utilizada para execução do MWC. Esses benefícios compensam o esforço de implementação necessário para contornar as dificuldades como, por exemplo, a criação do componente para converter as matrizes e os vetores no MWC com o componente `Web Service Actor`.

O uso do *framework* Axis® foi importante pois automatizou o processo de criação do serviço web. O Axis®, a partir de uma classe na linguagem Java®, gerou a descrição do serviço e gerenciou todo o protocolo de comunicação entre o cliente `Web Service Actor` do Kepler e a tarefa criada pelo autor, tornando a publicação de um serviço web em um processo de pouca complexidade.

O último MWC criado recebeu um novo componente responsável pela computação da integração numérica. Esse componente foi criado com base nas explicações encontradas na referência [46].

O ambiente de desenvolvimento Eclipse® [7], utilizado no desenvolvimento do código fonte do novo componente, foi de grande ajuda no estudo das classes do sistema Kepler. Esta ferramenta fornece mecanismos para ligação do projeto às classes Java do Kepler e recupera a partir do código fonte suas classes, métodos e documentação, tornando o aprendizado rápido e fácil.

Apesar dessas facilidades, os usuários que não tem conhecimento da especialidade de sistemas computacionais podem ter dificuldades para conseguir produzir um novo componente. O exemplo, apresentado em formato de guia na Seção 6.5.1, contribui para minimizar essa dificuldade.

Com relação à facilidade de extensão das tarefas do sistema, dentre os MWCs criados nesse estudo de caso, é de maior complexidade o modelo que cria um novo componente, pois requer conhecimentos da especialidade de sistemas computacionais. O MWC que interage com um serviço web é de menor complexidade, pois o componente `Web Service Actor` do Kepler facilita a utilização do serviço ao adicionar os dados de entrada e saída dinamicamente no componente, após configuração da descrição do serviço (WSDL). Por fim, o MWC que utiliza o *software* Matlab® é de pouca complexidade, pois o pesquisador precisa só conhecer como implementar na linguagem desse sistema.

6.5.1 Preparativos para criação de novo componente

O procedimento necessário para preparação do ambiente utilizado por este trabalho para edição dos novos componentes e dos componentes existentes é apresentado nesta seção, bem como um exemplo simples que ilustra a criação de um novo componente no sistema Kepler.

Os passos necessários para preparação deste ambiente são listados a seguir:

[7] Eclipse®: ambiente de desenvolvimento integrado para Java®/C/C++. `http://www.eclipse.org`

- instalar o JDK (*Java Development Kit*);
- instalar o Eclipse®;
- criar um novo projeto Java® no Eclipse®;
- nas propriedades do projeto, como ilustrado na Figura 6.13, adicionar os arquivos das bibliotecas necessárias para compilação do componente. Para adicionar os arquivos, siga os passos:
  - escolher a opção *Properties* no menu *Project*;
  - escolher o ítem *Java Build Path*;
  - clicar na guia *Libraries*;
  - clicar no botão *Add External JARs...*;
  - selecionar o arquivo da biblioteca necessária;
  - repetir os dois passos anteriores até adicionar todos os arquivos requeridos; e
  - clicar no botão *OK*.

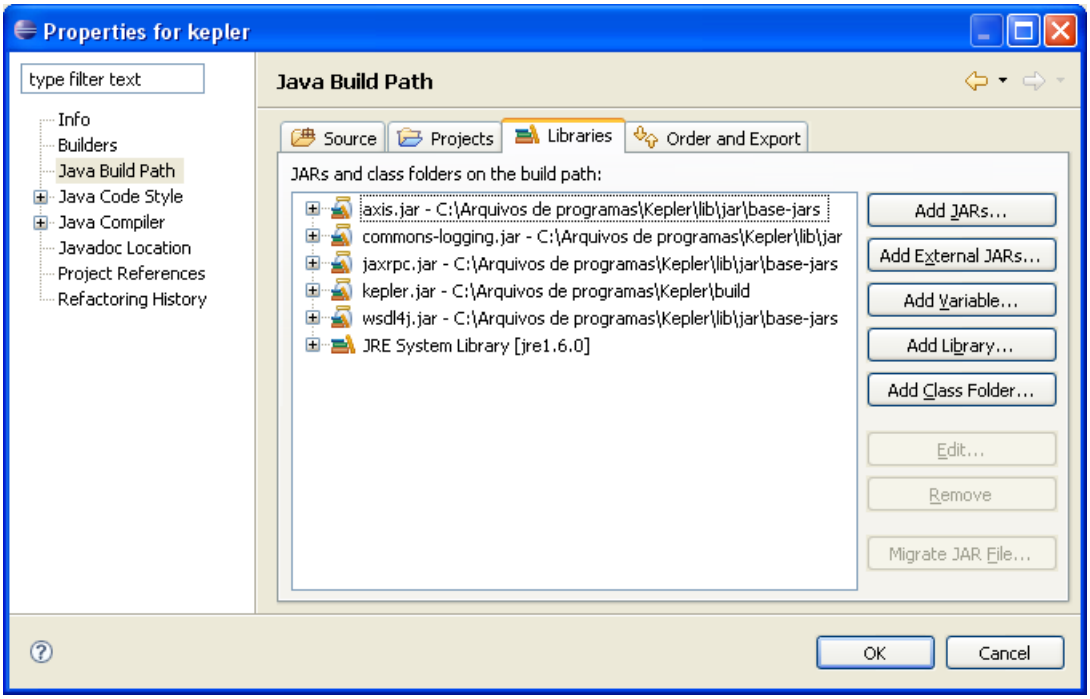

Figura 6.13: Bibliotecas adicionadas ao projeto na tela de propriedades no sistema Eclipse.

Apesar da referência [46] citar apenas um arquivo de bilioteca (`kepler.jar`), o ambiente utilizado por este trabalho fez uso de outros arquivos, tais como `axis.jar`, `commons-logging.jar`, `jaxrpc.ar` e `wsdl4j.jar`, todos ilustrados na figura. Essas bibliotecas foram adicionadas porque o Eclipse® disponibiliza uma funcionalidade onde o usuário pode navegar no código das diversas classes do projeto. Como o Kepler possui código aberto, é possível visualizar todas as biliotecas utilizadas pelos componentes.

O próximo passo é criar um novo componente, para tal é preciso criar uma especialização de uma classe disponível pelo sistema Kepler. O Algorítimo 6.3 apresenta um código exemplo de uma classe que pode ser especilizada.

```
/* HelloWorld actor for getting used to PtolemyII concepts.
Copyright (c) 1998-2003 The Regents of the University of California.
All rights reserved.
*/
// Exercise#1: ADD THE PACKAGE DESCRIPTION HERE
// Exercise#1: ADD THE import STATEMENTS HERE
import ptolemy.actor.TypedAtomicActor;
import ptolemy.kernel.CompositeEntity;
import ptolemy.kernel.util.*;

//////////////////////////////////////////////////////////////////////////
//// HellowWorld
/**
* Exercise#1: Add the class description here
  @author yourName
*/
public class HelloWorld extends TypedAtomicActor {

  // Exercise#1: ADD THE CLASS CONSTRUCTOR JAVADOC DESCRIPTION HERE.
  public HelloWorld(CompositeEntity container, String name)
      throws NameDuplicationException, IllegalActionException  {
    super(container, name);
    // Exercise#1: CREATE THE PARAMETER HERE!
    // Exercise#1: CREATE THE OUTPUT HERE...AND SET THE TYPE
    _attachText("_iconDescription", "<svg>\n" +
      "<rect x=\"0\" y=\"0\" "
      + "width=\"60\" height=\"20\" "
      + "style=\"fill:white\"/>\n" +
```

```
        "</svg>\n");
    }

    ////////////////////////////////////////////////////////////////
    ////                       ports and parameters                 ////
    // Exercise#1: ADD THE OUTPUT PORT DEFINITION HERE.
    // Exercise#1: ADD THE PARAMETER DEFINITION HERE.
    ////////////////////////////////////////////////////////////////
    ////                         public methods                     ////
    // Exercise#1: ADD THE JAVADOC DESCRIPTION FOR THE FIRE METHOD HERE.
    public void fire() throws IllegalActionException {
        super.fire();
        // Exercise#1: ADD THE LINE TO SEND THE greeting message FROM THE
            OUTPUT PORT.
    }
}
```

Algoritmo 6.3: Exemplo de uma classe básica para criação de um novo componente no sistema Kepler. Extraído da referência [46]

Seguindo os comentários do Algoritmo 6.3 (`Exercise#1`) e os passos descritos no exercício 1 da referência [46], o código fonte de um novo componente pronto para ser adicionado ao sistema Kepler fica como o apresentado no Algoritmo 6.4.

```
/* HelloWorld actor for getting used to PtolemyII concepts.
 Copyright (c) 1998-2003 The Regents of the University of California.
 All rights reserved.
*/
package edu.tutorial.sample;

import ptolemy.actor.TypedAtomicActor;
import ptolemy.kernel.CompositeEntity;
import ptolemy.kernel.util.*;
import ptolemy.actor.IOPort;
import ptolemy.actor.TypedIOPort;
import ptolemy.data.Token;
import ptolemy.data.StringToken;
import ptolemy.data.type.BaseType;
import ptolemy.data.type.Type;
import ptolemy.data.expr.Parameter;

////////////////////////////////////////////////////////////////
```

```
//// HellowWorld
/**
 * This is the implementation of a HelloWorld actor using Kepler.
 * This actor outputs "Hello World!" as string from its output port.
 @author Aldemon
*/
public class HelloWorld extends TypedAtomicActor {

  /** Construct a HelloWorld source with the given container and name.
   *  @param name The name of this actor.
   *  @exception IllegalActionException If the entity cannot be
   *   contained by the proposed container.
   *  @exception NameDuplicationException If the container already has
   *   an actor with this name.
   */
  public HelloWorld(CompositeEntity container, String name)
      throws NameDuplicationException, IllegalActionException  {
    super(container, name);
    userName = new Parameter(this, "userName", new StringToken(""));
    output = new TypedIOPort(this, "output", false, true);
    // Set the type constraint.
    output.setTypeEquals(BaseType.STRING);
    _attachText("_iconDescription", "<svg>\n" +
      "<rect x=\"0\" y=\"0\" "
      + "width=\"60\" height=\"20\" "
      + "style=\"fill:white\"/>\n" +
      "</svg>\n");
  }

  ////////////////////////////////////////////////////////////////
  ////                     ports and parameters                 ////
  /** The output port.  The type of this port will be set to String. */

  public TypedIOPort output = null;
  /** The name of the user. */
  public Parameter userName;

  ////////////////////////////////////////////////////////////////
  ////                         public methods                   ////
  /** Send the token in the value  parameter to the output.
   *  @exception IllegalActionException If it is thrown by the
   *   send() method sending out the token.
   */
  public void fire() throws IllegalActionException {
```

```
    super.fire();
    String userNameStr =  ((StringToken)userName.getToken()).
        stringValue();
    output.send(0, new StringToken("Hello " + userNameStr + "!"));
  }
}
```

Algoritmo 6.4: Exemplo de um novo componente pronto para ser adicionado ao sistema Kepler.

O componente criado no Algoritmo 6.4 é um exemplo simples, onde o componente recebe um nome como dado de entrada e armazena uma saudação (`"Hello "` + userNameStr + `"!"`) com o nome recebido no dado de saída. A seguir é descrito o que cada trecho de código adicionado faz em relação ao componente criado:

linha 5 indica o nome do pacote, ou seja, onde está armazenado a classe Java®;

linhas 10 à 16 são importadas as bibliotecas do Kepler para uso do componente;

linhas 37 à 40 são criados os objetos dos dados de entrada (userName) e dos dados de saída (output), e é difinada a tipagem (STRING) do dado de saída;

linhas 50 à 53 são definidos os objetos que representam os dados de entrada (userName) e os dados de saída (output);

linhas 63 à 64 o dado de entrada (userName) é utilizado para compor o texto que será armazenado no dado de saída (output).

Como apresentado nesse exemplo de componente, o número de dados de entrada e saída de um componente pode variar de acordo com a sua necessidade, inclusive, esses dados podem ser criados dinamicamente, como no componente `Web Service Actor`.

A interação do componente e do sistema Kepler é feito por meio de métodos que representam eventos durante a execução do MWC. Por exemplo, no componente apresentado, o método fire () é acionado quando o Kepler reconhece o evento de execução desse componente. Dentre os métodos que representam eventos são: preinitialize (), initialize (), prefire (), fire (), postfire (), entre outros.

Após criar o componente é preciso copiar os arquivos Java®, fonte e compilado, para o diretório de bibliotecas do Kepler (`<diretório do kepler>\lib`) e adicionar

o componente pela opção *Instantiate Componente* do menu *Tools*. Não há necessidade de recompilação do sistema Kepler, apenas do código do componente. Esse processo é ilustrado na Figura 6.14.

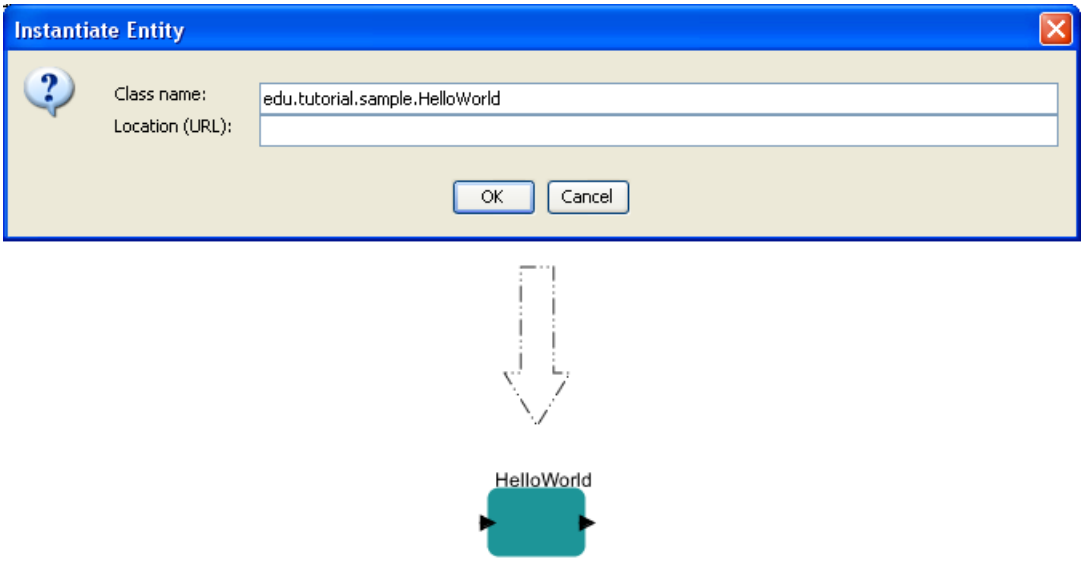

Figura 6.14: Exemplo de adição no MWC de um novo componente desenvolvido pelo usuário.

Para facilitar a reutilização do componente criado, recomenda-se adicioná-lo na paleta de componentes do sistema Kepler. Para isso, deve-se: clicar com o botão direito no componente, escolher a opção *Save in Library*, informar o nome do componente, escolher suas categorias e clicar no botão OK. Esses passos e o componente na paleta é ilustrado na Figura 6.15.

Outra opção de adição de componentes na paleta é por meio do encapsulamento de componentes existentes do Kepler. Uma fórmula no componente *Expression* também é outro exemplo de componente possível de ser adicionado na paleta do sistema, como ilustrado na referência [46].

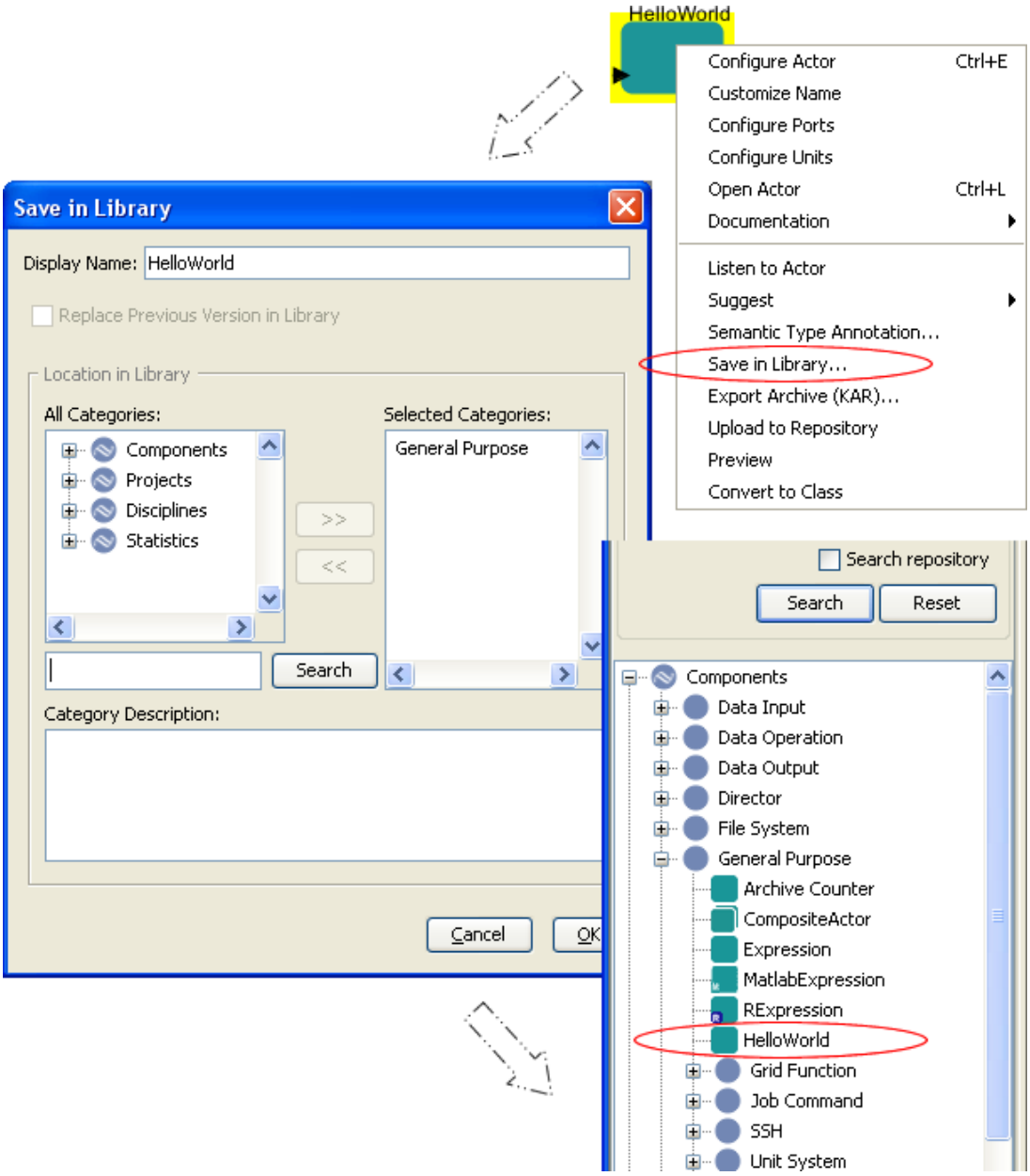

Figura 6.15: Exemplo de adição de um novo componente no MWC desenvolvido pelo usuário. Extraído do sistema Kepler.

Capítulo 7

Conclusões

Os SWCs fornecem meios para facilitar a construção e execução de MWCs. Baseado no estudo de quatro SWCs, foi produzida uma taxonomia, apresentada na Seção 2.1, que sintetiza e classifica o conjunto de conceitos e funcionalidades nesse contexto. Assim, o resultado prático da taxonomia é o de facilitar a compreensão dos conceitos envolvidos num SWC.

Constatou-se que um não especialista em sistemas computacionais é capaz de desenvolver novos MWCs, reutilizar componentes disponíveis nos SWCs, ou partes de outros MWCs desenvolvidos por outros pesquisadores. No caso do SWC Kepler, isso é facilitado por uma interface gráfica amigável e ajuda *online*. Nesse sentido, foram apresentados na Seção 2.2.4 vários tipos de componentes de diversas áreas do conhecimento; adicionalmente, relativo ao processo de execução de MWCs, foram apresentadas na Seção 2.3.1 opções de auxílio para encontrar erros nos MWCs; na Seção 2.3.2 maneiras de tolerância a falhas; e na Seção 2.3.3 foram discutidas possíveis formas para monitorar a execução. Este trabalho também traz contribuições com alguns guias de desenvolvimento.

Quando, por questões de eficiência na criação dos MWCs, for de interesse o uso de ferramentas externas ao SWC, existem mecanismos de extensão que facilitam essa integração. Como apresentado na Seção 2.2.5 e nos Capítulos 3 e 6, os SWCs, principalmente a ferramenta Kepler utilizada nesse trabalho, permitem integrar programas *ad hoc* e *software*s de diferentes tecnologias, tais como *web services*, ambiente Grid, Matlab®e R®.

Caso não seja possível compor a funcionalidade desejada para um MWC, usando os componentes disponíveis em um SWC, existe a possibilidade de se criar novos componentes. Essa alternativa demanda algum conhecimento de sistemas computacionais. Todo o processo é exemplificado na Seção 6.2.3. Esse exemplo foi apresentado no formato de um guia que serve para o usuário interessado em executar tais extensões.

A escolha do sistema Kepler, como justificado no Capítulo 4, foi baseada na facilidade de uso e de extensão de suas tarefas. Além de seus modelos computacionais, sua boa documentação e um processo de instalação prático e fácil.

Com relação às deficiências comentadas na Introdução, quanto ao processo de experimentação *in silico*, o uso de SWCs aprensenta as seguintes contribuições:

- a baixa reutilização de programas e sistemas é abordada utilizando-se as funcionalidades de exportação e importações de componentes no Kepler, em conjunto com os componentes que encapsulam um MWC, facilitando sua reutilização e sua distribuição;
- a integração dos sistemas computacionais usados nas diversas etapas do experimento *in silico* é facilitada ao agrupar esses sistemas em um MWC, o qual faz a automação e a gerência da passagem dos dados entre esses sistemas;
- o gerenciamento dos vários processos envolvidos nas etapas de construção e execução de um projeto que envolva diversos sistemas computacionais são facilitados com o uso de um SWC, que é capaz de salvar sua estrutura e carregá-la em outros momentos para execução. Em alguns casos, os SWCs também armazenam os resultados obtidos nas diversas etapas de construção de um MWC para posteriores comparações.

Outra contribuição deste trabalho foram os exemplos práticos do uso do sistema Kepler no domínio da área de sistemas Dinâmicos dos Corpos Deformáveis, que foram apresentados no Capítulo 6.

Concluiu-se que um MWC típico da área de sistemas Dinâmicos dos Corpos Deformáveis é suportado adequadamente pelo sistema Kepler, com as ressalvas comentadas a seguir.

Quanto a limitações encontradas para o desenvolvimento de um MWC na área de sistemas Dinâmicos dos Corpos Deformáveis pode-se citar a escassez de componentes que dêem suporte a manipulação de matrizes de forma eficiente e que contemplem as mais diversas operações da álgebra linear.

Outros benefícios que merecem destaque são: facilidade de conversão de formatos de dados; flexibilidade de integração com o Matlab® que contribuiu para a eficiência da criação de MWCs; facilidade de apresentação de resultados, principalmente gráficos e imagens; auxílio na depuração da execução; flexibilidade de integração com o R®; e código aberto escrito na linguagem Java®.

Existe também a necessidade do domínio das técnicas de programação e das tecnologias de suporte ao desenvolvimento de aplicações Java® no sistema Kepler para inclusão de novos componentes, como ilustrado na Seção 6.2.3. Os guias disponibilizados nesse trabalho facilitam grandemente essa tarefa.

Foram também estudados os custos adicionais de processamento ou armazenamento gerados pelos SWCs em relação à execução isolada das tarefas que compõem o MWC. A Seção 6.4 dá a dimensão desses custos através de uma medição dos tempos gastos em diferentes configurações de MWC. Quanto ao desempenho de execução dos experimentos, as funcionalidades dos SWCs devem ser escolhidas conforme as características do MWC em questão. Funções que visam a eficiência da modelagem, através do reuso de soluções prévias, acrescentam um tempo na execução do experimento que pode ou não ser aceitável. Considerações nesse particular foram feitas na seção 6.4.

MWCs onde o tempo de execução seja crítico, podem se beneficiar de funcionalidades no sentido da computação de alto desempenho, como discutido na Seção 3.5. Entretanto, em tais MWCs deve haver parcimônia quanto a aplicação de funções que visam a eficáfia do processo de desenvolvimento de MWCs.

Conclui-se portanto, que o uso do sistema Kepler contribui satisfatoriamente para solução dos problemas identificados na Introdução e é recomendado para o trabalho em grupos de pesquisa da área de modelagem computacional, particularmente em sistemas Dinâmicos dos Corpos Deformáveis.

Dentre as sugestões de trabalhos futuros, destacam-se:

- uma pesquisa mais detalhada da aplicação de mecanismos de processamento distribuído em ambientes Grid;
- criação de MWCs mais complexos na área de sistemas Dinâmicos dos Corpos Deformáveis que permitam uma melhor avaliação dos ganhos com gerenciamento do sistema de workflow;
- criação de um visualizador em tempo real da deformação das estruturas;
- criação de componentes para manipulação eficiente de matrizes e operações da álgebra linear;
- criação de um componente que aproveite as aplicações legadas feitas em fortran e C/C++;
- otimização da conversão dos dados para o componente `MatlabExpression`;
- estudo das classificações e buscas semânticas no sistema Kepler.

Referências Bibliográficas

[1] Akram A., Meredith D., and Allan R. Evaluation of BPEL to scientific workflows. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, pages 269–274, Washington, DC, USA, 2006. IEEE Computer Society.

[2] Alves A., Arkin A., Askary S., Bloch B., Curbera F., Goland Y., Kartha N., Sterling, König D., Mehta V., Thatte S., van der Rijn D., Yendluri P., and Yiu A. Web services business process execution language version 2.0. OASIS Committee Draft, Maio 2006.

[3] Barker A. and Hemert J.v. Scientific workflow: A survey and research directions. In *PPAM*, pages 746–753, 2007.

[4] Groehs A.G. *Mecânica Vibratória*. Editora Unisinos, São Leopoldo, Brasil, 2001.

[5] Ludäscher B., Altintas I., Berkley C., Higgins D., Jaeger E., Jones M., Lee E.A., Tao J., and Zhao Y. Scientific workflow management and the kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006.

[6] Berkley C., Bowers S., Jones M., Ludäscher B., Schildhauer M., and Tao J. Incorporating semantics in scientific workflow authoring. In *SSDBM'2005: Proceedings of the 17th international conference on Scientific and statistical database management*, pages 75–78, Berkeley, CA, US, 2005. Lawrence Berkeley Laboratory.

[7] Brooks C., Lee E.A., Liu X., Neuendorffer S., Zhao Y., and Zheng H. Heterogeneous concurrent modeling and design in java (volume 1: Introduction to

Ptolemy II). Technical Report UCB/EECS-2007-7, EECS Department, University of California, Berkeley, Janeiro 2007.

[8] Brooks C., Lee E.A., Liu X., Neuendorffer S., Zhao Y., and Zheng H. Heterogeneous concurrent modeling and design in java (volume 2: Ptolemy II software architecture). Technical Report UCB/EECS-2007-8, EECS Department, University of California, Berkeley, Janeiro 2007.

[9] Brooks C., Lee E.A., Liu X., Neuendorffer S., Zhao Y., and Zheng H. Heterogeneous concurrent modeling and design in java (volume 3: Ptolemy ii domains). Technical Report UCB/EECS-2007-9, EECS Department, University of California, Berkeley, Janeiro 2007.

[10] World Wide Web Consortium. Web services architecture. `http://www.w3.org/TR/ws-arch`, 2004.

[11] World Wide Web Consortium. Web services activity. `http://www.w3.org/2002/ws`, 2005.

[12] World Wide Web Consortium. Extensible markup language (XML). `http://www.w3.org/XML`, 2008.

[13] Ewins D. *Modal Testing, Theory, Practice, and Application*. Research Studies Press, Forest Grove, 2000.

[14] UC Berkeley Department of EECS. Ptolemy II. `http://ptolemy.eecs.berkeley.edu/ptolemyII`, Setembro 2007.

[15] Lee E.A. and Parks T.M. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801, 1995.

[16] Matos E.E.S. Celows : um framework baseado em ontologias com serviços web para modelagem conceitual em biologia sistêmica. Master's thesis, Universidade Federal de Juiz de Fora, Juiz de Fora, Minas Gerais, Brasil, Abril 2008.

[17] Kepler National Science Foundation. Kepler: Kepler execution monitoring. `http://www.kepler-project.org/Wiki.jsp?page=KeplerExecutionMonitoring`, Dezembro 2007.

[18] Kepler National Science Foundation. Kepler: Kepler project. `http://kepler-project.org`, Setembro 2007.

[19] Kepler National Science Foundation. Kepler: Kepler provenance use cases. `http://www.kepler-project.org/Wiki.jsp?page=KeplerProvenanceUseCases`, Dezembro 2007.

[20] Kepler National Science Foundation. Kepler: Kepler provenance framework. `http://www.kepler-project.org/Wiki.jsp?page=KeplerProvenanceFramework`, Fevereiro 2008.

[21] Vistrail National Science Foundation. Vistrails wiki. `http://www.vistrails.org`, Setembro 2007.

[22] Kahn G. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Information Processing '74: Proceedings of the IFIP Congress*, pages 471–475. North-Holland, New York, NY, 1974.

[23] Zhou G. Dynamic data flow modeling in Ptolemy II. Technical Report UCB/ERL M05/7, EECS Department, University of California, Berkeley, Jan 2005.

[24] Fox G.C. and Gannon D. Special issue: Workflow in grid systems. *Concurrency and Computation: Practice and Experience*, 18(10):1009–1019, 2006.

[25] GridTalk. How does grid computing work? `http://gridcafe.web.cern.ch/gridcafe/gridatwork/architecture.html`, Agosto 2008.

[26] gridworkflow.org. Grid workflow forum. `http://www.gridworkflow.org`, 2008.

[27] Deitel H.M. and Deitel P.J. *Java: Como Programar*. Prentice-Hall, 2005.

[28] Deitel H.M., Deitel P.J., and Nieto T.R. *Internet e World Wide Web: Como Programar*. Bookman, 2003.

[29] Altintas I., Berkley C., Jaeger E., Jones M., Ludascher B., and Mock S. Kepler: An extensible system for design and execution of scientific workflows. *ssdbm*, 00:423, 2004.

[30] Altintas I., Barney O., and Jaeger-Frank E. Provenance collection support in the kepler scientific workflow system. In *IPAW*, pages 118–132, 2006.

[31] Foster I. What is the grid? - a three point checklist. *GRIDtoday*, 1(6), July 2002.

[32] Foster I., Kesselman C., and Tuecke S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputing Applications*, 15(3), 2002.

[33] Taylor I.J., Deelman E., Gannon D.B., and Shields M. *Workflows for e-Science: Scientific Workflows for Grids*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[34] Cowan J. and Tobin R. Xml information set (second edition). World Wide Web Consortium, Recommendation REC-xml-infoset-20040204, Fevereiro 2004.

[35] Freire J., Koop D., Santos E., and Silva C.T. Provenance for computational tasks: A survey. *Computing in Science and Engineering*, 10(3):11–21, 2008.

[36] Yu J. and Buyya R. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Rec.*, 34(3):44–49, 2005.

[37] Bathe K. and Wilson E.L. *Numerical Methods in Finite Element Analysis*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1976.

[38] Plankensteiner K., Prodan R., Fahringer T., Kertesz A., and Kacsuk P. Fault-tolerant behavior in state-of-the-art Grid Workflow Management systems. In *Proc. of the CoreGRID Integration Workshop 2008*, Hersonisson, Crete, April 2008.

[39] Lins L., Koop D., Anderson E.W., Callahan S.P., Santos E., Scheidegger C.E., Freire J., and Silva C.T. Examining statistics of workflow evolution provenance: A first study. In *SSDBM*, pages 573–579, 2008.

[40] Muliadi L. Discrete event modeling in Ptolemy II. Technical Report UCB/ERL M99/29, EECS Department, University of California, Berkeley, 1999.

[41] Seffino L.A., Medeiros C.B., Rocha J.V., and Yi B. Woodss — a spatial decision support system based on workflows. *Decision Support Systems*, 27(1-2):105–123, 1999.

[42] J. Liu. Continuous time and mixed-signal simulation in Ptolemy II. Technical Report UCB/ERL M98/74, EECS Department, University of California, Berkeley, 1998.

[43] Goel M. Process networks in Ptolemy II. Technical Report UCB/ERL M98/69, EECS Department, University of California, Berkeley, 1998.

[44] National Institute of Standards and Technology. Jama: Java matrix package. `http://math.nist.gov/javanumerics/jama`, Setembro 2007.

[45] Kepler Project. *Getting Started with Kepler*, Maio 2008. versão 1.0.0.

[46] Kepler Project. *Kepler User Manual*, Maio 2008. versão 1.0.0.

[47] Taverna Project. *Taverna 1.7 Manual*, Dezembro 2007.

[48] The Apache XML Project. Axis user's guide. `http://ws.apache.org/axis`, 2005.

[49] Vistrails Project. *Vistrails User's Guide*, Setembro 2007. versão 1.0.

[50] Brown R. Calendar queues: a fast priority queue implementation for the simulation event set problem. *Communications of the ACM*, 31(10):1220–1227, 1988.

[51] Clough R. *Dynamics of Structures*. McGraw-Hill, New York, 1975.

[52] Hoare C. A. R. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, Agosto 1978.

[53] Majithia S., Shields M., Taylor I., and Wang I. Triana: a graphical web service composition and execution toolkit. In *Triana: a graphical Web service composition and execution toolkit*, pages 514–521, 2004.

[54] Callahan S.P., Freire J., Santos E., Scheidegger C.E., Silva C.T., and Vo H.T. Vistrails: visualization meets data management. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 745–747, New York, NY, USA, 2006. ACM.

[55] Ho T. and Abramson D. Active data: Supporting the grid data life cycle. In *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on*

Cluster Computing and the Grid, pages 39–48, Washington, DC, USA, 2007. IEEE Computer Society.

[56] Oinn T., Addis M., Ferris J., Marvin D., Senger M., Greenwood M., Carver T., Glover K., Pocock M.R., Wipat A., and Li P. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.

[57] Taverna. Taverna project website. `http://taverna.sourceforge.net`, Setembro 2007.

[58] The Triana Team. *Triana User Guide*, Março 2005.

[59] Cardiff University. The triana project. `http://www.trianacode.org`, Setembro 2007.

[60] Cirne W. and Neto E.S. Grids computacionais: da computação de alto desempenho a serviços sob demanda. 23rd Brazilian Symposium on Computer Networks, Maio 2005.

[61] Wikipédia. Grafos acíclicos dirigidos. `http://pt.wikipedia.org/wiki/Grafos_acíclicos_dirigidos`, Novembro 2007.

[62] Wikipédia. Taxonomia. `http://pt.wikipedia.org/wiki/Taxonomia`, Novembro 2007.

[63] Wikipédia. Bug. `http://pt.wikipedia.org/wiki/Bug`, Fevereiro 2008.

[64] Wikipédia. Business process execution language. `http://en.wikipedia.org/wiki/BPEL`, Fevereiro 2008.

[65] Wikipédia. Cache. `http://pt.wikipedia.org/wiki/Cache`, Fevereiro 2008.

[66] Wikipédia. Código aberto. `http://pt.wikipedia.org/wiki/Código_aberto`, Fevereiro 2008.

[67] Wikipédia. Depuração. `http://pt.wikipedia.org/wiki/Depura%C3%A7%C3%A3o`, Fevereiro 2008.

[68] Wikipédia. Hypertext transfer protocol. http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol, Fevereiro 2008.

[69] Wikipédia. Loose coupling. http://en.wikipedia.org/wiki/Loose_coupling, Fevereiro 2008.

[70] Wikipédia. Semântica. http://pt.wikipedia.org/wiki/Semântica, Agosto 2008.

[71] Wikipédia. Sistema de controle de versão. http://pt.wikipedia.org/wiki/Sistema_de_controle_de_vers%C3%A3o, Fevereiro 2008.

[72] Wikipédia. Soap. http://pt.wikipedia.org/wiki/SOAP, Fevereiro 2008.

[73] Wikipédia. Tabela hash. http://pt.wikipedia.org/wiki/Tabela_hash, Fevereiro 2008.

[74] Wikipédia. Web service. http://pt.wikipedia.org/wiki/Web_service, Fevereiro 2008.

[75] Wikipédia. Web services description language. http://en.wikipedia.org/wiki/Web_Services_Description_Language, Fevereiro 2008.

[76] Wikipédia. Xml. http://pt.wikipedia.org/wiki/XML, Fevereiro 2008.

[77] Simmhan Y.L., Plale B., and Gannon D. A survey of data provenance in e-science. *SIGMOD Rec.*, 34(3):31–36, 2005.

Apêndice A

Códigos utilizados no modelo de *workflow* científico com MatlabExpression

A.1 Partes do código do componente *MatlabExpression* adaptado

O código fonte do componente apresentado abaixo foi adaptado do código disponibilizado no componente original no sistema Kepler, disponível na referência [18].

```
public synchronized void fire() throws IllegalActionException {
  super.fire();
  boolean fireSwitch=true;
  if (triggerSwitch.getWidth() > 0) {
    Object inputSwitch = triggerSwitch.get(0);
    if(inputSwitch instanceof IntToken) {
      if(((IntToken)inputSwitch).intValue() >= 1 ) {
        fireSwitch = true;
      }
      else
        fireSwitch = false;
    }
```

```
    else if (inputSwitch instanceof BooleanToken) {
      if(((BooleanToken)inputSwitch).booleanValue()) {
        fireSwitch = true;
      }
      else
        fireSwitch = false;
    }

  }
  if (fireSwitch) {
    _debug("building script");
    String script = buildScript();
    String outputString = "";
    int randInt = Math.abs((new Random()).nextInt());
    String randomFilename = tempFilename + "." + randInt;
    boolean isWindows = (System.getProperty("os.name").indexOf("Windows
        ") > -1);
    _debug("Matlab file = " + tempFilenameM + ".m");
    _debug("Temp file = " + randomFilename);
    _debug("write script file");
    try {
      FileWriter fileWriter = new FileWriter(tempFilenameM+".m");
      BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);
      bufferedWriter.write(script);
      bufferedWriter.close();
    } catch (IOException e) {
      _debug("IOException: " + e.getMessage());
    }
    String[] argList;
    if (isWindows) {
      argList = new String[1];
      argList[0] = mlCmd.getExpression()
      + " -r " + tempFilenameM
      + " -nodesktop"
      + " -nosplash"
      + " -logfile " + randomFilename;
      String logstr = "";
      for (int i = 0; i < argList.length; i++) {
        logstr += argList[i] + " ";
      }
      _debug("cmd = " + logstr);
    } else {
      argList = new String[4];

```

```
      argList[0] = mlCmd.getExpression();
      argList[1] = "-nodesktop";
      argList[2] = "-nosplash" + " ";
      argList[3] = "-r" + " \"" + tempFilenameM + "\"";
    }
    try {
      _debug("running...");
      if (isWindows) {
        _debug("execute +");
        _p = Runtime.getRuntime().exec(argList[0]);
        _debug("done +");
      } else {
        _p = Runtime.getRuntime().exec(argList);
      }
      int result = -1;
      if (isWindows) {
        _debug("waitFor");
        result = _p.waitFor();
        _debug("result = " + result);
        if (result <= 1) {
          _debug("getting result");
          BufferedReader in = new BufferedReader(new FileReader(
              randomFilename));
          int count = 0;
          while (in.ready()) {
            outputString += in.readLine() + "\n";
            if ((++count % 1024) == 0)
              _debug("outputString size = " + outputString.length());
          }
          in.close();
        }
      } else {
        _debug("getting result");
        BufferedInputStream inputstreambuffer = new BufferedInputStream
            (_p.getInputStream());
        BufferedInputStream bufferedInputStream = new
            BufferedInputStream(inputstreambuffer);
        BufferedReader bufferedReader = new BufferedReader(new
            InputStreamReader(bufferedInputStream));
        int count = 0;
        while ( (outputString += bufferedReader.readLine()) != null ) {
          if (++count % 100 == 0)
            _debug("outputString size = " + outputString.length());
        }
```

```
        bufferedReader.close();
      }
      if (outputString == "") {
        _debug("inputstream is null");
      }
      if (isWindows) {
        _debug("delete files");
        (new File(randomFilename)).delete();
        (new File(tempFilenameM+".m")).delete();
      } else {
        result = _p.waitFor();
      }
      switch (result) {
      case 0:
        break;
      case 1:
        break;
      case -113:
        output.send(0, new StringToken("Matlab process was forcfully
            killed"));
        return;
      case 129:
        output.send(0, new StringToken("Matlab process was forcfully
            killed"));
        return;
      default:
        output.send(0, new StringToken("Matlab process returned \"" +
            result + "\".\nSomething must have gone wrong"));
      return;
      }
    } catch (IOException e) {
      _debug("IOException: " + e.getMessage());
    } catch (InterruptedException e) {
      _debug("interupted!");
    }
    _debug("parsing output");
    parseOutput(outputString);
  }
  _debug("send output");
}

private String buildScript() throws IllegalActionException {
  List ipList = inputPortList();
  Iterator ipListIt = ipList.iterator();
```

```
    String inputs = "";
    while (ipListIt.hasNext()) {
      TypedIOPort tiop = (TypedIOPort)ipListIt.next();
      if(!(tiop.getName().equals("expression")) && !(tiop.getName().
          equals("triggerSwitch")) ) {
        Token[] token = new Token[1];
        token[0] = tiop.get(0);
        String variable = tiop.getName() + " = ";
        variable += "[" + getTokenValue(token) + "];";
        _debug(variable);
        inputs += variable + "\n";
      }
    }
    List opList = outputPortList();
    Iterator opListIt = opList.iterator();
    String outputs = "";
    while (opListIt.hasNext()) {
      TypedIOPort tiop = (TypedIOPort)opListIt.next();
      if (!tiop.equals(output)) {
        outputs += "if exist('" + tiop.getName() + "');\n" + tiop.getName
            () + "\nelse;\n" + tiop.getName() + "=0\nend;\n";
      }
    }
    expression.update();
    String script =
      inputs +
      "sprintf('----');\n" +
      ((StringToken)expression.getToken()).stringValue() + "\n" +
      "sprintf('----');\n" +
      outputs +
      "quit\n";
    _debug("------------------------------------------------");
    _debug("  SCRIPT");
    _debug("------------------------------------------------");
    _debug(script);
    _debug("------------------------------------------------");
    return script;
  }

  private void parseOutput(String outputString) throws
      IllegalActionException {
    String results = outputString.replaceAll("\r", "");
    results = results.replaceAll("\n\n\n", "*");
    results = results.replaceAll("\n\n", "\n");
```

```
    StringTokenizer st = new StringTokenizer(results, "*");
    boolean outputSent = false;
180 while (st.hasMoreTokens()) {
      String ssst = st.nextToken();
      if (outputSent) {
        StringTokenizer rst = new StringTokenizer(ssst, "\n");
        if (rst.countTokens() > 2) {
185       Token tokenValue = null;
          String portName = "";
          TypedIOPort tiop = null;
          while (rst.hasMoreTokens()) {
            String sst = rst.nextToken();
190         StringTokenizer ist = new StringTokenizer(sst);
            if (portName == "") {
              portName = ist.nextToken();
              String fss = ist.nextToken();
              if (fss.equals("=")) {
195             if (portName.equals(output.getName())) {
                  throw new IllegalActionException("sending a custom
                      token out of port " + output.getName() + " is bad!"
                      );
                }
                Iterator opListIt = outputPortList().iterator();
                while (opListIt.hasNext()) {
200               tiop = (TypedIOPort)opListIt.next();
                  if (tiop.getName().equals(portName)) {
                    break;
                  }
                }
205           }
              else {
                portName = "";
              }
            }
210         else {
              if (ist.countTokens() == 1) {
                if (rst.countTokens() > 0) {
                  int count = 0;
                  Token[] list = new Token[rst.countTokens()+1];
215               int ttype = 0; // 1 - Double; 2 - Integer; 3 - String;
                  String value = ist.nextToken();
                  if (tiop.getType().equals(new ArrayType(BaseType.DOUBLE
                      ))) {
                    list[count++] = new DoubleToken( value );
```

```
                ttype = 1;
            }
            else if (tiop.getType().equals(new ArrayType(BaseType.
                INT))) {
              list[count++] = new IntToken( value );
              ttype = 2;
            }
            else {
              list[count++] = new StringToken( value );
              ttype = 3;
            }
            while (rst.hasMoreTokens()) {
              ist = new StringTokenizer(rst.nextToken());
              value = ist.nextToken();
              if (ttype == 1) {
                list[count++] = new DoubleToken( value );
              }
              else if (ttype == 2) {
                list[count++] = new IntToken( value );
              }
              else { // ttype == 3
                list[count++] = new StringToken( value );
              }
            }
            tokenValue = new ArrayToken(list);
          }
          else {
            String value = ist.nextToken();
            if (tiop.getType().equals(BaseType.DOUBLE)) {
              tokenValue = new DoubleToken( value );
            }
            else if (tiop.getType().equals(BaseType.INT)) {
              tokenValue = new IntToken( value );
            }
            else {
              tokenValue = new StringToken( value );
            }
          }
        }
        else {
          int count = 0, column = 0;
          Token[] list = new Token[rst.countTokens()+1];
          int ttype = 0; // 0 - String; 1 - Double; 2 - Integer
          int rowSize = ist.countTokens();
```

```
                Token[] row = new Token[rowSize];
                while (ist.hasMoreTokens()) {
                  String value = ist.nextToken();
265               if (ttype == 0) {
                    if (tiop.getType().equals(new ArrayType(new ArrayType
                        (BaseType.DOUBLE)))) {
                      row[column++] = new DoubleToken( value );
                      ttype = 1;
                    }
270                 else if (tiop.getType().equals(new ArrayType(new
                        ArrayType(BaseType.INT)))) {
                      row[column++] = new IntToken( value );
                      ttype = 2;
                    }
                    else {
275                   row[column++] = new StringToken( value );
                      ttype = 3;
                    }
                  }
                  else if (ttype == 1) {
280                 row[column++] = new DoubleToken( value );
                  }
                  else if (ttype == 2) {
                    row[column++] = new IntToken( value );
                  }
285               else if (ttype == 3) {
                    row[column++] = new StringToken( value );
                  }
                }
                list[count++] = new ArrayToken(row);
290             while (rst.hasMoreTokens()) {
                  column = 0;
                  row = new Token[rowSize];
                  ist = new StringTokenizer(rst.nextToken());
                  while (ist.hasMoreTokens()) {
295                 if (ttype == 1) {
                      row[column++] = new DoubleToken( ist.nextToken() );
                    }
                    else if (ttype == 2) {
                      row[column++] = new IntToken( ist.nextToken() );
300                 }
                    else if (ttype == 3) {
                      row[column++] = new StringToken( ist.nextToken() );
                    }
```

```
                }
                list[count++] = new ArrayToken(row);
              }
              tokenValue = new ArrayToken(list);
            }
          }
        }
        if (tiop != null) {
          tiop.send(0, tokenValue);
        }
      }
      System.gc();
    }
    else {
      outputSent = true;
      output.send(0, new StringToken(ssst));
    }
  }
}

private String getTokenValue(Token[] token) throws
    IllegalActionException {
  String returnval = "";
  for (int i = 0 ; i < token.length ; i++) {
    if (token[i].getType().isCompatible(new ArrayType(BaseType.UNKNOWN)
        )) {
      returnval += " [ " + getTokenValue(((ArrayToken)token[i]).
          arrayValue()) + " ] ";
    } else if (token[i].getType().isCompatible(new ArrayType(new
        ArrayType(BaseType.UNKNOWN)))) {
      returnval += " [ ";
      Token aToken[] = ((ArrayToken)token[i]).arrayValue();
      Integer arrayLength = aToken.length;
      for (int j = 0 ; j < arrayLength; j++) {
        returnval += ((j > 0) ? "; " : "") + getTokenValue(((ArrayToken
            )aToken[j]).arrayValue());
      }
      returnval += " ] ";
    } else if (token[i].getType().equals(BaseType.STRING)) {
      returnval += " '" + ((StringToken)token[i]).stringValue() + "' ";
    } else if (token[i].getType().equals(BaseType.INT)) {
      returnval += " " + ((IntToken)token[i]).intValue() + " ";
    } else if (token[i].getType().equals(BaseType.DOUBLE)) {
      returnval += " " + ((DoubleToken)token[i]).doubleValue() + " ";
```

```
    } else if (token[i].getType().equals(BaseType.BOOLEAN)) {
      returnval += " " + ((BooleanToken)token[i]).toString() + " ";
    } else if (token[i].getType().equals(BaseType.LONG)) {
      returnval += " " + ((LongToken)token[i]).longValue() + " ";
    } else if (token[i].getType().equals(BaseType.UNSIGNED_BYTE)) {
      returnval += " " + ((UnsignedByteToken)token[i]).byteValue() + "
          ";
    } else {
      throw new IllegalActionException("invalid token type: " + token[i
          ].getType().toString());
    }
  }
  return returnval;
}

private void setOutputToken(String portName, String[][] value) throws
    IllegalActionException {
  List opList = outputPortList();
  Iterator opListIt = opList.iterator();
  if (portName.equals(output.getName())) {
    throw new IllegalActionException("sending a custom token out of
        port " + output.getName() + " is bad!");
  }
  while (opListIt.hasNext()) {
    TypedIOPort tiop = (TypedIOPort)opListIt.next();
    String thisPortName = tiop.getName();
    if (thisPortName.equals(portName)) {
      int type = -1;
      for (int i = value.length-1 ; i < value.length ; i++) {
        try {
          if (value[i][0].indexOf(".") > -1) {
            Double.valueOf(value[i][0]);
            type = type > 1 ? type : 1;
          } else {
            Integer.valueOf(value[i][0]);
            type = type > 0 ? type : 0;
          }
        } catch (NumberFormatException e) {
          type = type > 2 ? type : 2;
        }
      }
      ArrayToken token = null;
      if (type == 2) {
        Token[] row = new Token[value.length];
```

```
        for (int i = 0 ; i < value.length ; i++) {
          Token[] column = new Token[value[0].length];
          for (int j = 0 ; j < value[0].length ; j++) {
            column[j] = new StringToken(value[i][j]);
          }
          row[i] = new ArrayToken(column);
        }
        token = new ArrayToken(row);
      }
      else if (type == 1) {
        Token[] row = new Token[value.length];
        for (int i = 0 ; i < value.length ; i++) {
          Token[] column = new Token[value[0].length];
          for (int j = 0 ; j < value[0].length ; j++) {
            column[j] = new DoubleToken(Double.valueOf(value[i][j]).
                doubleValue());
          }
          row[i] = new ArrayToken(column);
        }
        token = new ArrayToken(row);
      } else if (type == 0) {
        Token[] row = new Token[value.length];
        for (int i = 0 ; i < value.length ; i++) {
          Token[] column = new Token[value[0].length];
          for (int j = 0 ; j < value[0].length ; j++) {
            column[j] = new IntToken(Integer.valueOf(value[i][j]).
                intValue());
          }
          row[i] = new ArrayToken(column);
        }
      } else {
        throw new IllegalActionException("invalid value passed for
            token");
      }
      tiop.send(0, token);
      break;
    }
  }
}
```

Algoritmo A.1: Código fonte adaptado do componente *MatlabExpression*.

A.2 Partes do código fonte do componente *Matriz*

O pedaço de código fonte apresentado a seguir foi criado pelo autor para converter os dados em um formato reconhecido pelo sistema Kepler.

```
public void fire() throws IllegalActionException {
  super.fire();
  try {
    vbuffer = ((StringToken)buffer.get(0)).stringValue();
  }
  catch (IllegalActionException e) {
    System.out.println(e.getLocalizedMessage());
  }
  catch (NoTokenException e) {
  }
  parserInput();
  matrix.send( 0, getArrayToken( vmatrix ) );
}

private void parserInput() {
  List<String> line = new ArrayList<String>();
  StringTokenizer tokenizer = new StringTokenizer(vbuffer, "\n");
  String text;
  while (tokenizer.hasMoreTokens()) {
    text = tokenizer.nextToken().trim();
    if (text.length() > 0)
      line.add(text);
  }
  String[] tokens;
  int lin, col;
  double value;
  while (line.size() > 0) {
    text = line.remove(line.size() - 1).trim();
    if ( (text.indexOf("(") >= 0) && (text.indexOf(")") >= 0) ) {
      tokens = text.split("[\\(\\),]");
      lin = Integer.parseInt(tokens[1].trim());
      col = Integer.parseInt(tokens[2].trim());
      value = Double.parseDouble(tokens[3].trim());
      if (vmatrix == null) {
        vmatrix = new double[lin][col];
```

```
        }
        vmatrix[lin-1][col-1] = value;
      }
    }
}

private ArrayToken getArrayToken(double[][] doubleVar) {
  ArrayToken array = null;
  try {
    int sizeColumn = doubleVar[0].length;
    Token[] row = new Token[doubleVar.length];
    for (int i = 0; i < doubleVar.length; i++) {
      Token[] column = new Token[sizeColumn];
      for (int j = 0; j < sizeColumn; j++) {
        column[j] = new DoubleToken( doubleVar[i][j] );
      }
      row[i] = new ArrayToken(column);
    }
    array = new ArrayToken(row);
  }
  catch (IllegalActionException e) {
  }
  return( array );
}
```

Algoritmo A.2: Trecho do código fonte do componente *Matriz*, criado pelo autor.

A.3 Partes de código fonte do componente *XYPlotterMatrix*

O pedaço do código fonte apresentado abaixo foi adaptado do código disponibilizado no componente `XYPlotter` no sistema Kepler, disponível na referência [18].

```
public void fire() throws IllegalActionException {
  super.fire();
  Token[] vX = null;
  Token[] vY = null;
  try {
    vX = ((ArrayToken)inputX.get(0)).arrayValue();
```

```
    vY = ((ArrayToken)inputY.get(0)).arrayValue();
  }
  catch (IllegalActionException e) {
    System.out.println(e.getLocalizedMessage());
  }
  catch (NoTokenException e) {
  }
  int widthX = vX.length;
  int widthY = vY.length;
  int widthYColumn = ((ArrayToken) vY[0]).length();
  if (widthX != widthY) {
    throw new IllegalActionException(this,
        " The number of input channels mismatch. (inputX = "+ widthX +"
            ; inputY = "+ widthY +","+ widthYColumn +")");
  }
  int offset = ((IntToken) startingDataset.getToken()).intValue();
  for (int j = widthX - 1; j >= 0; j--) {
    double xValue = ((DoubleToken) vX[j]).doubleValue();
    Token[] Y = ((ArrayToken) vY[j]).arrayValue();
    for (int i = widthYColumn - 1; i >= 0; i--) {
      double yValue = ((DoubleToken) Y[i]).doubleValue();
      ((Plot) plot).addPoint(i + offset, xValue, yValue, true);
    }
  }
  triggerDone.send(0, new IntToken(1));
}
```

Algoritmo A.3: Código fonte do componente *XYPlotterMatrix* adaptado do componente *XYPlotter*.

Apêndice B

Códigos utilizados no modelo de *workflow* científico com serviço Web

B.1 Código fonte do serviço web

O código apresentado a seguir é uma tarefa disponibiliza por meio de um serviço web que faz a computação dos deslocamentos verticais, das formas modais e das frequências naturais de vibração.

```
import java.util.StringTokenizer;
import Jama.EigenvalueDecomposition;
import Jama.Matrix;
public class wsDynamics {
  private static final long serialVersionUID = -516827151633036721 1L;
  private int vnmodo;
  private int vnp;
  private double vdt;
  private double[] vF = null; /* Matrix completa com a Forças */
  private double[] vFA = null; /* Matrix completa com a Força aplicada
      */
  private Matrix vM = null; /* Matrix completa de massa */
  private Matrix vK = null; /* Matrix completa de rigidez */
```

```
  private double[] vGL = null; /* Matrix completa com os Graus de
      liberdade */
  private double[] vxi = null;
  private double[][] vphi = null;
  private double[] vomega = null;
  private double[][] vv = null;
  private double[] vx = null;
  private double[] diag;
  private int vNGL = 0; /* conta quantos Graus de liberdade existem */

  public String compute(int nmodo, int np, double dt, String FA, String
      F, String M, String K, String GL, String xi) {
    vnmodo = nmodo;
    vnp = np;
    vdt = dt;
    vFA = getDoubleVector( FA );
    vF = getDoubleVector( F );
    vM = new Matrix( getDoubleMatrix(M) );
    vK = new Matrix( getDoubleMatrix(K) );
    vGL = getDoubleVector( GL );
    vxi = getDoubleVector( xi );
    prepareData();
    computeModalIntegrator();
    String result = "";
    result += getArrayString( vx ) +";";
    result += getArrayString( vv ) +";";
    result += getArrayString( vphi ) +";";
    result += getArrayString( vomega );
    return result;
  }

  private void prepareData() {
    vNGL = 0;
    for (int i = 0; i < vGL.length; i++) {
      if (vGL[i] == 1) {
        vNGL++;
      }
    }
    int vMLength = vM.getRowDimension();
    if (vFA.length < vMLength) {
      double[] tmp = vFA;
      vFA = new double[vMLength];
      for (int i = 0; i < tmp.length; i++) {
        vFA[i] = tmp[i];
```

```
55        }
        }
      }

      private void computeModalIntegrator() {
60      vphi = getEigenvalue();
        Matrix FmTranspose = new Matrix(vFA, 1).transpose();
        Matrix matrixPhi = new Matrix(vphi.length, vnmodo);
        for (int i = 0; i < vphi.length; i++) {
          for (int j = 0; j < vnmodo; j++) {
65          matrixPhi.getArray()[i][j] = vphi[i][j];
          }
        }
        Matrix Mm = (matrixPhi.transpose().times(vM)).times(matrixPhi);
        for (int i = 0; i < matrixPhi.getRowDimension(); i++) {
70        for (int j = 0; j < matrixPhi.getColumnDimension(); j++) {
            matrixPhi.getArray()[i][j] = matrixPhi.getArray()[i][j] / Math.
                pow(Mm.get(j, j), 0.5);
          }
        }
        vphi = matrixPhi.getArray();
75      Mm = (matrixPhi.transpose().times(vM)).times(matrixPhi);
        Matrix Km = (matrixPhi.transpose().times(vK)).times(matrixPhi);
        Matrix Fm = (matrixPhi.transpose().times(FmTranspose));
        vomega = new double[vnmodo];
        for (int i = 0; i < vnmodo; i++) {
80        vomega[i] = Math.pow(diag[i], 0.5);
        }
        double Cm[] = new double[vnmodo];
        for (int i = 0; i < vnmodo; i++) {
          Cm[i]= 2 * vomega[i] * vxi[i];
85      }
        int nmodo = Km.getRowDimension();
        double[] fm = Fm.getColumnPackedCopy();
        double[] X1 = new double[nmodo];
        double[] X2 = new double[nmodo];
90      double[] X = new double[nmodo];
        vv = new double[vnp][vNGL];
        vx = new double[vnp];
        for(int cont = 0; cont < vnp; cont++) {
          vx[cont] = (cont + 1) * vdt;
95        int f = 0;
          for (int g = 0; g < vGL.length; g++) {
            if (vGL[g] == 1) {
```

```
            double P1, P2, P3;
            for (int j = 0; j < fm.length; j++) {
              double vFvalue = (cont < vF.length) ? vF[cont] : 0.0;
              P1 = (2 * vFvalue * vdt * vdt) * fm[j];
              P2 = (-2 * vdt * vdt) * X1[j] * Km.getArray()[j][j];
              P3 = 4 * X1[j] - 2 * X2[j] + X2[j] * Cm[j] * vdt;
              X[j] = (P1 + P2 + P3) / (2 + vdt * Cm[j]);
            }
            double auxX[] = new double[vphi.length];
            for (int i = 0; i < vphi.length; i++) {
              for (int j = 0; j < vphi[0].length; j++) {
                auxX[i] += vphi[i][j] * X[j];
              }
            }
            vv[cont][f] = auxX[g];
            f += 1;
          }
        }
        for (int i=0; i < X2.length; i++){
          X2[i] = X1[i];
          X1[i] = X[i];
        }
      }
    }

    private double[][] getEigenvalue() {
      int size = vK.getColumnDimension();
      Matrix matrixAux = new Matrix(size, size);
      matrixAux = vM.inverse().times(vK);
      EigenvalueDecomposition E = matrixAux.eig();
      double[][] phi = E.getV().getArrayCopy();
      diag = E.getD().getDiagonalCopy();
      int i, j, t;
      double p;
      int n = phi.length;
      for (i = 0; i < n; i++) {
        p = diag[t = i];
        for (j = i + 1; j < n; j++) {
          if (diag[j] <= p) {
            p = diag[t = j];
          }
        }
        if (t != j) { // troca
          diag[t] = diag[i];
```

```
        diag[i] = p;
        for (j = 0; j < n; j++) { // troca
          p = phi[j][i];
          phi[j][i] = phi[j][t];
          phi[j][t] = p;
        }
      }
    }
    return phi;
  }

  private double[][] getDoubleMatrix(String tokenMatrix) {
    String buffer = tokenMatrix.substring(2, tokenMatrix.length()-2 ).
        replace("},{", "|");
    StringTokenizer tokenizer = new StringTokenizer(buffer, "|");
    double[][] matrix = null;
    int i = 0;
    while (tokenizer.hasMoreTokens()) {
      String bufferRow = tokenizer.nextToken();
      StringTokenizer tokenizerRow = new StringTokenizer(bufferRow, ","
          );
      if (matrix == null) {
        matrix = new double[tokenizer.countTokens()+1][tokenizerRow.
            countTokens()];
      }
      int j = 0;
      while (tokenizerRow.hasMoreTokens()) {
        matrix[i][j] = Double.parseDouble(tokenizerRow.nextToken());
        j++;
      }
      i++;
    }
    return(matrix);
  }

  private double[] getDoubleVector(String tokenVector) {
    String buffer = tokenVector.substring(1, tokenVector.length()-1 );
    StringTokenizer tokenizer = new StringTokenizer(buffer, ",");
    double[] vector = new double[tokenizer.countTokens()];
    int count = 0;
    while (tokenizer.hasMoreTokens()) {
      vector[count] = Double.parseDouble(tokenizer.nextToken());
      count++;
    }
```

```
    return(vector);
  }

  private String getArrayString(double[][] doubleVar) {
    String array = "";
    int sizeColumn = doubleVar[0].length;
    for (int i = 0; i < doubleVar.length; i++) {
      String row = "";
      for (int j = 0; j < sizeColumn; j++) {
        row += doubleVar[i][j] + ",";
      }
      array += "{"+ row.substring(0, row.length()-1) +"},";
    }
    array = "{"+ array.substring(0, array.length()-1) +"}";
    return( array );
  }

  private String getArrayString(double[] doubleVar) {
    String array = "";
    for (int i = 0; i < doubleVar.length; i++) {
      array += doubleVar[i] + ",";
    }
    array = "{"+ array.substring(0, array.length()-1) +"}";
    return( array );
  }
}
```

Algoritmo B.1: Código fonte do serviço web.

B.2 Partes do código fonte do componente *Sequência* Matriz

O pedaço de código fonte apresentado a seguir foi criado pelo autor para converter os dados em um formato para ser enviado para o serviço web. O fonte deste componente é idêntico ao apresentado no Apêndice A.2, a única diferença é a função getArrayString().

```
private String getArrayString(double[][] doubleVar) {
```

```
  String array = "";
  int sizeColumn = doubleVar[0].length;
  for (int i = 0; i < doubleVar.length; i++) {
    String row = "";
    for (int j = 0; j < sizeColumn; j++) {
      row += doubleVar[i][j] + ",";
    }
    array += "{"+ row.substring(0, row.length()-1) +"},";
  }
  array = "{"+ array.substring(0, array.length()-1) +"}";
  return( array );
}
```

Algoritmo B.2: Código fonte do componente *Sequência* Matriz, criado pelo autor.

Apêndice C

Códigos utilizados no modelo de *workflow* científico com novo componente

C.1 Código fonte do novo componente

O código fonte apresentado a seguir faz a computação dos deslocamentos verticais, das formas modais e das frequências naturais de vibração.

```
package br.ufjf.mmc.dynamicstructure;

import ptolemy.actor.NoTokenException;
import ptolemy.actor.TypedAtomicActor;
import ptolemy.actor.TypedIOPort;
import ptolemy.data.ArrayToken;
import ptolemy.data.DoubleToken;
import ptolemy.data.IntToken;
import ptolemy.data.Token;
import ptolemy.data.type.ArrayType;
import ptolemy.data.type.BaseType;
import ptolemy.kernel.CompositeEntity;
import ptolemy.kernel.util.IllegalActionException;
import ptolemy.kernel.util.NameDuplicationException;
```

```
15 import Jama.EigenvalueDecomposition;
   import Jama.Matrix;

   public class Dynamics extends TypedAtomicActor {
     private static final long serialVersionUID = -5168271516330367211L;
20   public TypedIOPort nmodo = null;
     public TypedIOPort np = null;
     public TypedIOPort dt = null;
     public TypedIOPort F = null;
     public TypedIOPort FA = null;
25   public TypedIOPort M = null;
     public TypedIOPort K = null;
     public TypedIOPort GL = null;
     public TypedIOPort xi = null;
     public TypedIOPort phi = null;
30   public TypedIOPort omega = null;
     public TypedIOPort pt = null;
     public TypedIOPort x = null;
     private int vnmodo;
     private int vnp;
35   private double vdt;
     private double[] vFA = null; /* Força aplicada */
     private double[] vF = null; /* Forças */
     private Matrix vK = null; /* Rigidez */
     private Matrix vM = null; /* Massa */
40   private double[] vGL = null; /* Graus de liberdade */
     private double[] vxi = null;
     private double[][] vphi = null;
     private double[] vomega = null;
     private double[][] vx = null;
45   private double[] vpt = null;
     private double[] diag;
     private int vNGL = 0;

     public Dynamics(CompositeEntity container, String name) throws
         NameDuplicationException, IllegalActionException {
50     super(container, name);
       nmodo  = new TypedIOPort(this, "nmodos", true, false);
       np   = new TypedIOPort(this, "np", true, false);
       dt   = new TypedIOPort(this, "dt", true, false);
       FA   = new TypedIOPort(this, "FA", true, false);
55     F  = new TypedIOPort(this, "F", true, false);
       M  = new TypedIOPort(this, "M", true, false);
       K  = new TypedIOPort(this, "K", true, false);
```

```
    GL  = new TypedIOPort(this, "GL", true, false);
    xi   = new TypedIOPort(this, "xi", true, false);
    phi = new TypedIOPort(this, "phi", false, true);
    omega = new TypedIOPort(this, "omega", false, true);
    pt = new TypedIOPort(this, "X", false, true);
    x = new TypedIOPort(this, "V", false, true);
    nmodo.setTypeEquals(BaseType.INT);
    np.setTypeEquals(BaseType.INT);
    dt.setTypeEquals(BaseType.DOUBLE);
    FA.setTypeEquals(new ArrayType(BaseType.DOUBLE));
    F.setTypeEquals(new ArrayType(BaseType.DOUBLE));
    M.setTypeEquals(new ArrayType(new ArrayType(BaseType.DOUBLE)));
    K.setTypeEquals(new ArrayType(new ArrayType(BaseType.DOUBLE)));
    GL.setTypeEquals(new ArrayType(BaseType.DOUBLE));
    xi.setTypeEquals(new ArrayType(BaseType.DOUBLE));
    phi.setTypeEquals(new ArrayType(new ArrayType(BaseType.DOUBLE)));
    omega.setTypeEquals(new ArrayType(BaseType.DOUBLE));
    pt.setTypeEquals(new ArrayType(BaseType.DOUBLE));
    x.setTypeEquals(new ArrayType(new ArrayType(BaseType.DOUBLE)));
    _attachText("_iconDescription", "<svg>\n" +
        "<rect x=\"0\" y=\"0\" "
        + "width=\"60\" height=\"20\" "
        + "style=\"fill:white\"/>\n" +
        "</svg>\n");
  }

  public void fire() throws IllegalActionException {
    super.fire();
    prepareData();
    computeModalIntegrator();
    phi.send( 0, getArrayToken( vphi ) );
    omega.send( 0, getArrayToken( vomega ) );
    x.send( 0, getArrayToken( vx ) );
    pt.send( 0, getArrayToken( vpt ) );
  }

  private double[][] getDoubleMatrix(Token[] tokenLine) {
    Token[] tokenColumn = ((ArrayToken)tokenLine[0]).arrayValue();
    double[][] matrix = new double[tokenLine.length][tokenColumn.length
        ];
    for (int i = 0; i < tokenLine.length; i++) {
      tokenColumn = ((ArrayToken)tokenLine[i]).arrayValue();
      for (int j = 0; j < tokenColumn.length; j++) {
        matrix[i][j] = ((DoubleToken)tokenColumn[j]).doubleValue();
```

```
        }
      }
      return(matrix);
    }

    private double[] getDoubleVector(Token[] tokenVector) {
      double[] vector = new double[tokenVector.length];
      for (int i = 0; i < tokenVector.length; i++) {
        vector[i] = ((DoubleToken)tokenVector[i]).doubleValue();
      }
      return(vector);
    }

    private void prepareData() {
      // igual à função do Apêndice B.1, na linha 42.
    }
    private void computeModalIntegrator() {
      // igual à função do Apêndice B.1, na linha 59
    }
    private double[][] getEigenvalue() {
      // igual à função do Apêndice B.1, na linha 123.
    }
    private ArrayToken getArrayToken(double[][] doubleVar) {
      // igual à função do Apêndice A.2, na linha 42.
    }

    private ArrayToken getArrayToken(double[] doubleVar) {
      ArrayToken array = null;
      try {
        Token[] row = new Token[doubleVar.length];
        for (int i = 0; i < doubleVar.length; i++) {
          row[i] = new DoubleToken( doubleVar[i] );
        }
        array = new ArrayToken(row);
      }
      catch (IllegalActionException e) {
      }
      return( array );
    }

}
```

Algoritmo C.1: Código fonte do novo componente inserido no sistema Kepler.