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

Laryssa Aparecida Machado da Silva

COMPOSER-SCIENCE: UM *FRAMEWORK* PARA A COMPOSIÇÃO DE *WORKFLOWS* CIENTÍFICOS

Juiz de Fora
2010

COMPOSER-SCIENCE: UM *FRAMEWORK* PARA A COMPOSIÇÃO DE *WORKFLOWS* CIENTÍFICOS

Laryssa Aparecida Machado da Silva

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Modelagem Computacional da Universidade Federal de Juiz de Fora, como parte dos requisitos necessários à obtenção do grau de Mestre em Ciências em Modelagem Computacional.

Orientador: Regina Maria Maciel Braga Villela

Juiz de Fora
Julho de 2010

Silva, Laryssa Aparecida Machado da.

Composer-science: um framework para a composição de workflows científicos / Laryssa Aparecida Machado da Silva. – 2010.
170 f.

Dissertação (Mestrado em Modelagem Computacional)— Universidade Federal de Juiz de Fora, Juiz de Fora, 2010.

1. Processamento eletrônico de dados. 2. Semântica. 3. Web sites. I. Título.

CDU 681.34

Laryssa Aparecida Machado da Silva

COMPOSER-SCIENCE: UM *FRAMEWORK* PARA A COMPOSIÇÃO DE *WORKFLOWS* CIENTÍFICOS

Dissertação apresentada ao Programa de Pós-graduação em Modelagem Computacional da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do grau de Mestre.

Aprovada em 5 de julho de 2010.

BANCA EXAMINADORA

_____

Profa. Dra. Regina Maria Maciel Braga Villela – Orientadora

Universidade Federal de Juiz de Fora

_____

Profa. Dra. Fernanda Cláudia Alves Campos

Universidade Federal de Juiz de Fora

_____

Profa. Dra. Maria Cláudia Cavalcanti

Instituto Militar de Engenharia

_____

Prof. Dr. Marco Antônio Pereira Araújo

Centro de Ensino Superior de Juiz de Fora

Aos meus pais e marido.

AGRADECIMENTOS

A Deus, em primeiro lugar, pois sem Ele esta jornada não seria cumprida.

À minha orientadora Regina Braga, pela excelente orientação, por toda a dedicação, paciência e incentivo ao longo deste trabalho.

À prof. Fernanda Campos pelo auxílio nos trabalhos realizados.

Aos membros da Banca Examinadora pelo trabalho de avaliação.

Aos meus pais, Solange e Ovídio, por me darem a vida e me ensinarem a vivê-la.

Ao meu marido, Vitor, pelo amor, carinho, paciência, companheirismo, incentivo, dedicação e por tornar meus dias mais felizes.

À Universidade Federal de Juiz de Fora pelo apoio financeiro através da bolsa de mestrado.

À todos que, direta ou indiretamente, contribuíram para a realização deste trabalho.

Resumo da Dissertação apresentada à Universidade Federal de Juiz de Fora como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências

Um conceito importante nas pesquisas em *e-Science* é o de *workflows* científicos, que, em geral, são longos, compostos de várias aplicações que, em conjunto, representam um experimento científico. Uma possibilidade para auxiliar na definição destes *workflows* científicos é o uso de ferramentas que agreguem semântica para auxiliar na sua composição. Os serviços Web semânticos apresentam tecnologias altamente favoráveis à sua composição para a obtenção de processos mais complexos, tais como o uso de padrões Web, independência de plataforma, independência de linguagem de programação utilizada para o desenvolvimento, possibilidade de processamento distribuído, e, principalmente, o uso de recursos semânticos que possibilitem sua descoberta, composição e invocação automáticas. Com o objetivo de auxiliar na descoberta de serviços Web para a composição de *workflows* científicos, propomos o desenvolvimento de um *framework*, denominado Composer-Science, que realize a busca de serviços Web semânticos e componha estes, definindo assim, um *workflow* científico. O objetivo geral do *Composer-Science* é permitir que o pesquisador descreva semanticamente um *workflow* científico e, considerando essa descrição, automatize, por meio do uso de serviços Web semânticos e ontologias, a busca semântica por serviços em repositórios e a geração de *workflows* científicos a partir dessa composição. O objetivo geral do *framework* pode ser decomposto em objetivos específicos: o registro e o armazenamento, nos repositórios distribuídos (bancos de dados) do *framework,* de ontologias de domínio (OWL) e anotações dos serviços Web semânticos (OWL-S); a realização de pesquisa semântica, baseada em requisitos fornecidos pelo pesquisador, nos repositórios distribuídos, a fim de realizar a descoberta de serviços Web semânticos que atendam os requisitos semânticos fornecidos; a análise sintática, baseada em requisitos estruturais (dados de entrada e saída), além da análise semântica dos serviços descobertos por meio da pesquisa semântica, a fim de se obter possíveis composições dos mesmos; a geração de modelos de *workflows* em WS-BPEL a partir das composições possíveis. Desta forma, os modelos gerados pelo *framework* podem ser utilizados em Sistemas de Gerenciamento de Workflows Científicos (SGWfC) e serem compostos com outros modelos de *workflow*.

Palavras-chave: *Workflow* científico. Serviço Web semântico. Composição de serviços.

Abstract of Dissertation presented to Federal University of Juiz de Fora as a partial fulfillment of the requirements for the degree of Master of Science

An important concept in e-Science research is scientific workflows, which are usually long, consisting of several applications that, together, represent a scientific experiment. One possibility to assist in defining these scientific workflows is the use of tools that add semantics to the composition process. Semantic Web services have technologies that are highly favorable to their composition, in order to obtain more complex processes. Examples of these technologies are the use of Web standards, platform independence, programming language independence, possibility of distributed processing and especially the use of semantic resources that enable their discovery, automatic composition and invocation. With the aim of assisting in the discovery of Web services for scientific workflows composition, we propose the development of a framework, named Composer-Science, to conduct the search for semantic Web services and compose them, thus defining a scientific workflow. The overall objective of Composer-Science is to allow researcher to describe semantically a scientific workflow and, considering this description automatize, through the use of semantic web services and ontologies, the semantic search for services in repositories and the generation of scientific workflows from this composition. The overall objective of the framework can be broken down into specific objectives: registration and storage of domain ontologies (OWL) and semantic annotations of Web services (OWL-S), in distributed repositories (databases) of the framework; implementation of semantic search, based on requirements provided by the researcher, in distributed repositories, in order to discovery semantic Web services that match the semantic requirements provided; the syntactic analysis, based on structural requirements (input and output), and semantic analysis of services discovered using semantic search, in order to obtain their possible compositions; the generation of WS-BPEL workflow models from the possible compositions. Finally, the models generated by the framework can be used in Workflow Management Systems (WMS) and composed with other workflow models.

Keywords: Scientific Workflows. Semantic Web service. Service composition.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

O contexto de *e-Science* está fortemente relacionado ao processo de modelagem computacional, envolvendo o uso de algoritmos já conhecidos ou que foram desenvolvidos a partir de técnicas de simulação, manipulação e mineração de dados, dentre outros processos, em que o modelo é um dos resultados da própria pesquisa, sendo interpretado como um processo que filtra, transforma, aglutina e gera dados. Por meio da modelagem computacional é possível estudar o desempenho de grandes sistemas computacionais sem que seja necessário implementá-los ou ainda, executar experimentos virtuais sem a necessidade da construção de uma estrutura física para tal [SBC, 2006].

Os produtos gerados pelos experimentos assim como os próprios modelos computacionais e matemáticos que representam esses experimentos são artefatos que podem ser utilizados na realização de outros experimentos científicos. Porém, existem algumas dificuldades na utilização desses artefatos como, por exemplo, a dependência de uma determinada tecnologia, como uma linguagem de programação ou uma plataforma, que foi utilizada para o seu desenvolvimento, ou a não exposição dos artefatos de alguma maneira que permita que eles sejam aproveitados pela comunidade científica. O encapsulamento de artefatos científicos em serviços Web é capaz de solucionar o problema da dependência de determinadas tecnologias, pois um serviço Web utiliza padrões conhecidos que permitem a independência de linguagem de programação e plataforma. Além disso, os serviços podem ser disponibilizados na Web, de forma que facilite sua descoberta por cientistas interessados em reutilizá-los.

Atualmente, com a difusão dos serviços Web capazes de auxiliar nas pesquisas em diversas áreas científicas, há um crescente interesse em utilizá-los para compor *workflows* que representem procedimentos experimentais. Entretanto, como o número de serviços disponíveis na Web vem crescendo, e os *workflows* científicos são bastante complexos, envolvendo muitas tarefas, se torna difícil e custoso, em termos de tempo, buscar manualmente os serviços capazes de realizar estas tarefas, uma a uma. A fim de resolver este problema, diversos estudos vêm sendo realizados no sentido de automatizar a busca por serviços que realizam determinada tarefa. Uma forma de otimizar este trabalho é o uso de buscas semânticas.

Os artefatos gerados precisam ainda de mecanismos que facilitem a interação, a interoperabilidade e o compartilhamento entre grupos de pesquisa que estejam trabalhando cooperativamente ou mesmo que tenham como domínio a mesma área de interesse, aproveitando ao máximo os recursos que a Web pode proporcionar no âmbito do trabalho colaborativo.

1.2 JUSTIFICATIVA

Atualmente, os sistemas computacionais são considerados essenciais para a pesquisa científica, suportando todos os aspectos relacionados ao seu ciclo de vida. Neste contexto, *e-Science* pode ser utilizada para caracterizar o importante papel das tecnologias computacionais na pesquisa, colaboração, compartilhamento de dados e documentos, e uso de recursos para automatizar a execução e análise de dados de experimentos científicos [PARASTATIDIS, 2009].

O termo *e-Science* pode ser utilizado, portanto, para descrever o desenvolvimento de infraestruturas de serviços de software capazes de prover acesso a facilidades remotas, recursos computacionais distribuídos, armazenamento de informações em bancos de dados dedicados, disseminação e compartilhamento de dados, resultados e conhecimento [LAUSCHNER, 2005]. Considerando este cenário, tecnologias como agentes, *workflows* científicos, ontologias e serviços Web semânticos podem ser utilizadas como base para a composição de uma infraestrutura de apoio a *e-Science*.

Na área da Ciência da Computação, uma ontologia define uma especificação formal e explícita dos termos de um domínio e das relações entre eles [GRUBER, 1995]. Em aplicações de *e-Science* as ontologias podem ser aplicadas em diversos contextos tais como: auxiliar na troca de informações entre aplicações científicas que trabalham em domínios correlatos, permitir a descoberta semântica de serviços Web semânticos e auxiliar na composição de serviços Web em *workflows* científicos [PALAZZI *et al*., 2009]. Da mesma forma, as tecnologias de *workflow* fornecem ambiente para a resolução de problemas por cientistas, facilitando a criação e execução de experimentos a partir do uso de uma grande quantidade de dados e serviços disponíveis [PIGNOTTI *et al.,* 2008].

O uso de recursos computacionais para a realização de pesquisas científicas não é trivial, pois, em geral, os cientistas de outras áreas, que não a área de computação, têm dificuldade para lidar com as tecnologias computacionais disponíveis em baixo nível de abstração. Sendo assim, o *framework* proposto nessa dissertação tem como objetivo facilitar

a definição de *workflows* científicos, permitindo que os cientistas descrevam seus *workflows* em um alto nível de abstração, utilizando recursos semânticos.

Além da dificuldade de utilização das ferramentas computacionais por cientistas, existe também a dificuldade para se encontrar artefatos de software que realizem determinada tarefa necessária ao experimento que está sendo realizado. Considerando os serviços disponíveis na Web, por exemplo, é difícil encontrar, manualmente, um serviço para desempenhar uma dada tarefa de um *workflow* científico. Dessa forma, o *framework* proposto visa automatizar a busca por serviços em seus repositórios a partir de busca semântica.

1.3 OBJETIVOS

Esta dissertação tem como principal objetivo propor um *framework* que auxilie a comunidade científica na tarefa de especificar computacionalmente experimentos científicos, em alto nível de abstração, por meio da utilização dos recursos de repositórios de serviços Web semânticos, ontologias, composição de serviços Web semânticos, e *workflows* científicos.

O *framework,* denominado Composer-Science*,* faz parte de um contexto amplo que consiste na especificação e desenvolvimento de uma infraestrutura, denominada ASOW-Science [MATOS *et al*., 2009], para aplicações em *e-Science*, cujo objetivo é o suporte computacional a projetos de pesquisa e a pesquisadores que desejam compartilhar experiências e resultados. Esta infraestrutura visa apoiar o gerenciamento de aplicações e resultados de pesquisa relacionados a um dado domínio de aplicação.

O objetivo geral do *Composer-Science* é permitir que o pesquisador descreva semanticamente um *workflow* científico e, considerando essa descrição, automatize, por meio do uso de serviços Web semânticos e ontologias, a busca semântica por serviços em repositórios e a geração de *workflows* científicos a partir dessa composição. O objetivo geral do *framework* pode ser decomposto em objetivos específicos:

i) o registro e o armazenamento, nos repositórios distribuídos (bancos de dados) do *framework,* de ontologias de domínio (OWL) e anotações dos serviços Web semânticos (OWL-S); ii) a realização de pesquisa semântica, baseada em requisitos fornecidos pelo pesquisador, nos repositórios distribuídos, a fim de realizar a descoberta de serviços Web semânticos que atendam os requisitos semânticos fornecidos; iii) a análise sintática, baseada em requisitos estruturais (dados de entrada e saída), além da análise semântica dos serviços descobertos por meio da pesquisa semântica, a fim de se obter possíveis

composições dos mesmos; e iv) a geração de modelos de *workflows* em WS-BPEL a partir das composições possíveis.

1.4 ESTRUTURA DO TRABALHO

Esta dissertação está organizada em seis capítulos. Este capítulo apresenta uma breve introdução que engloba a apresentação da motivação do trabalho, a definição de seus objetivos globais e específicos, assim como o detalhamento de sua estrutura.

O capítulo 2 apresenta os principais conceitos relacionados a este trabalho como, a definição do contexto de *e-Science*, no qual o trabalho está inserido; a definição de *workflows* científicos e de sua importância para a pesquisa científica; o conceito de ontologia do ponto de vista de sua aplicação neste trabalho; e o conceito de Serviços Web Semânticos detalhando a linguagem OWL-S para sua descrição semântica.

O capítulo 3 apresenta os principais conceitos envolvidos no processo de composição de workflows científicos, detalhando conceitos importantes para fundamentar a abordagem escolhida, apresentando o principal formalismo utilizado neste trabalho para auxiliar na modelagem da composição dos workflows, que são as Redes de Petri [PETRI, 1962], detalhando a linguagem de composição de workflows WS-BPEL, utilizada para gerar o modelo de composição proposto na abordagem, e apresentando alguns trabalhos relacionados.

O capítulo 4 descreve os objetivos do *framework* Composer-Science e a arquitetura proposta para sua implementação, detalhando a arquitetura e apresentando as ferramentas utilizadas.

O capítulo 5 apresenta cenários de uso para a utilização do *framework* Composer-Science proposto. É detalhado um estudo de caso que consiste na definição de um *workflow* científico da área de bioinformática por meio da utilização do protótipo do *framework*. Detalhes relacionados à implementação do protótipo também são apresentados.

Finalmente, o capítulo 6 apresenta as conclusões e sugere trabalhos futuros.

2 PRESSUPOSTOS TEÓRICOS

A Computação revolucionou a pesquisa científica, sendo hoje reconhecida como o “terceiro pilar” a sustentá-la, junto com os pilares da teoria e da experimentação [CARVALHO *et al.*, 2006]. Assim, ela se faz presente em grande parte dos avanços em todas as áreas do conhecimento. Novas formas de interação entre as ciências, em vários níveis e escalas, são mediadas pela união da Ciência da Computação com diferentes domínios do conhecimento. Muitas das grandes descobertas científicas recentes são resultados do trabalho de equipes multidisciplinares que envolvem cientistas da Computação [CARVALHO *et al.*, 2006].

Atualmente, estão sendo aplicados cada vez mais recursos computacionais para auxiliar no desenvolvimento de atividades de pesquisa nas mais diversas áreas da ciência. A utilização do recurso de serviços Web é bastante propícia nesta situação, devido às suas características de interoperabilidade, escalabilidade e flexibilidade [BALBI; PIRES e MATTOSO, 2005].

Com o crescimento do número de serviços Web em várias áreas, há interesse em construir serviços compostos, que executem tarefas mais complexas. Serviços Web compostos são gerados por meio da combinação de outros serviços, e por isso, são capazes de executar funções mais complexas que um serviço simples. Entretanto, como o número de serviços disponíveis vem crescendo, é cada vez mais difícil e demorado para o usuário encontrar os serviços desejados.

Para auxiliar na resolução do problema de composição de serviços Web, vários esforços vêm sendo feitos no sentido de automatizar essa tarefa. Tecnologias como *workflows* científicos [TAYLOR *et al.*, 2006], ontologias [GRUBER, 1992; GRUBER, 1994; BORST, 1997], e serviços Web semânticos [CARDOSO, 2007] podem ser utilizadas para permitir essa automatização. Este trabalho tem como principal objetivo apresentar a proposta de um *framework* capaz de auxiliar na definição e composição automática de *workflows* científicos a partir de serviços Web semânticos.

Neste capítulo, apresentamos os principais conceitos relacionados ao contexto deste trabalho. A seção 2.1 define o contexto de *e-Science*, no qual este trabalho está inserido. A seção 2.2 define o que são *workflows* científicos e sua importância para a pesquisa científica. A seção 2.3 define o conceito de ontologia do ponto de vista de sua aplicação neste trabalho. A seção 2.4 apresenta o conceito de Serviços Web Semânticos detalhando a linguagem OWL-S para sua descrição semântica.

2.1 *E-SCIENCE*

Atualmente, os sistemas computacionais têm se tornado importantes para a pesquisa científica, suportando todos os aspectos relacionados ao seu ciclo de vida. A comunidade científica tem utilizado os termos *e-Science* e *e-Research* para englobar o importante papel das tecnologias computacionais na pesquisa, colaboração, compartilhamento de dados e documentos, e uso de recursos para automatizar a execução e análise de dados de experimentos científicos [PARASTATIDIS, 2009].

A utilização de recursos computacionais no desenvolvimento da pesquisa beneficia o trabalho das comunidades científicas facilitando o compartilhamento de dados e serviços computacionais, além de contribuir para a construção de uma infra-estrutura de dados e de uma comunidade científica distribuída [LUDÄSCHER *et al*., 2004].

Este contexto em que a computação se torna parte integrante e imprescindível para o sucesso na realização de pesquisas científicas das mais variadas áreas, é o contexto de *e-Science*, em que a ciência é realizada com o apoio computacional, se tornando assim, mais eficiente. Soluções computacionais são muito importantes para o desenvolvimento de pesquisas em áreas como, por exemplo, a biologia, que gera grande volume de dados e necessita do apoio computacional para a realização e gerenciamento de experimentos.

As atividades de *e-Science* estão crescendo por todo o mundo, acompanhadas por uma proliferação de dados e ferramentas. Isto traz novos desafios, por exemplo, como entender e organizar esses recursos, como compartilhar e reutilizar experimentos bem sucedidos (ferramentas e dados), e como prover interoperabilidade entre dados e ferramentas de diferentes locais e utilizados por usuários com perfis distintos. Estes desafios geram pesquisas na área da computação, no sentido de desenvolver mecanismos que ofereçam todo o apoio necessário à pesquisa. A computação está em busca de uma infra-estrutura que permita projetar, reusar, anotar, validar, compartilhar e documentar artefatos gerados pela pesquisa científica[DIGIAMPIETRI, 2007].

O termo *e-Science* foi introduzido, no Reino Unido, por Taylor, para encapsular as tecnologias necessárias ao suporte à pesquisa colaborativa e multidisciplinar que emergiu em vários campos da ciência [HINE, 2006]. Taylor reconheceu a importância do uso de ferramentas computacionais na pesquisa científica colaborativa, multidisciplinar e com grande volume de dados, e usou o termo *e-Science* para englobar as ferramentas e tecnologias necessárias ao suporte a esse tipo de pesquisa [HEY *et al*., 2009].

São características de *e-Science* o acesso a uma vasta coleção de dados, utilização de recursos computacionais em larga escala, utilização de recursos heterogêneos e dinâmicos de múltiplas organizações e *workflows*. No contexto de *e-Science*, temos a

utilização de cada vez mais recursos computacionais para o auxílio ao desenvolvimento de atividades de pesquisa. Muitas vezes, é necessária a combinação desses recursos para obtenção de maior poder de processamento, para a interação entre grupos de pesquisas separados geograficamente, ou simplesmente para se obter resultados finais que precisam ser gerados a partir da agregação de resultados parciais obtidos pelo uso de diversas ferramentas.

A Computação em Grade ou *Grid* Computacional (*Grid Computing*) é uma tecnologia bastante relacionada às atividades de *e-Science*. Um *Grid* Computacional trata do compartilhamento de recursos relacionados e soluções de problemas, em organizações virtuais distribuídas geograficamente. A chave deste conceito é a habilidade de negociar compartilhamento de recursos dispostos ao longo de um conjunto de partes participantes, provedores e consumidores, e então usar os resultados desta fusão de recursos para algum propósito [LAUSCHNER, 2005].

A infraestrutura oferecida pela computação em *grid* permite que instituições de pesquisa geograficamente distribuídas interajam e compartilhem dados, metadados e modelos através de ferramentas para a execução em ambientes de alto desempenho. Cada componente da grade pode atuar tanto como cliente como na condição de fornecedor de dados, informações e conhecimentos. Essas facilidades permitem maior integração de informações oriundas de diferentes grupos científicos. A computação em grade permite o desenvolvimento e a implementação de infraestruturas de gerenciamento e disponibilização contínua de dados, informações e conhecimentos relativos ao desenvolvimento e a aplicação de soluções dinâmicas em diversas áreas [LAUSCHNER, 2005].

2.2 *WORKFLOW* CIENTÍFICO

A necessidade de sistemas capazes de isolar o fluxo de tarefas da automatização de processos de escritório foi identificada nos anos 1970. Ao separar o fluxo do restante do sistema, ganhou-se flexibilidade e agilidade quando modificações eram necessárias. Além disso, aos profissionais especialistas em áreas de negócios, puderam ser atribuídas as responsabilidades por essas tarefas, sem que fossem necessários conhecimentos de programação [NARDI, 2009].

O tema ganhou a atenção de diversas empresas, que constituíram a *Workflow Management Coalition* (*WfMC*) em 1993. A *WfMC,* definiu *workflow* como a automatização de um processo de negócio ou de parte dele, englobando documentos, informações ou tarefas que são passadas entre os participantes do processo, de acordo com regras de

negócios pré-definidas. Um processo de negócio é uma instância de uma tarefa bem definida que é freqüentemente repetida como parte de uma regra de negócio [WfMC, 1995].

O conceito de *workflow* também pode ser definido, em linhas gerais, como um modelo que define o fluxo de processos ou fluxo de tarefas coordenadas e encadeadas usando um plano sistemático. Diz respeito à organização de um conjunto de procedimentos, em que documentos, informações ou tarefas são trocados entre os participantes. A distribuição de tarefas entre os participantes é baseada em regras e requisitos, constantes na definição do *workflow*. A colaboração de todos os participantes é necessária para alcançar um objetivo global de um processo, podendo ser realizada manualmente ou automaticamente [FERNANDES e NOVAIS, 2007].

Os *workflows* fornecem meios sistemáticos e automatizados para a condução de análises de conjuntos de dados e integração de aplicações. São uma forma de capturar processos cujos resultados podem ser reproduzidos e métodos podem ser revisados, validados, repetidos e adaptados. O *workflow* fornece uma interface por meio da qual os cientistas podem criar fluxos de trabalho sem a necessidade de programação de baixo nível. O *workflow* consiste em uma plataforma para integração e acesso ao crescente número recursos computacionais independentes disponíveis, por cientistas que não possuem conhecimento especializado sobre cada um dos componentes ou sobre computação [GOBLE e ROURE, 2009].

Além das aplicações em sistemas de negócios, *workflows* também são aplicados no cenário científico, que lhes acrescenta algumas complexidades como [NARDI, 2009]:

- Fluxos com grande número de etapas, comum no processamento de tarefas em áreas como bioinformática, química, astronomia, agronomia, entre outras;
- Volatilidade dos fluxos, que devem poder ser alterados frequentemente, na avaliação de hipóteses científicas;
- Necessidade de parametrização para grande número de tarefas, por exemplo, em buscas por padrões no genoma;
- Monitoramento e acompanhamento da execução, que deve levar em conta a dinamicidade dos fluxos;
- Execução em ambientes cujos recursos sejam desconhecidos a princípio.

A computação experimental em larga escala tem causado grande impacto nas práticas científicas, produzindo avanços significados em diversas áreas de pesquisa. O poder computacional vem sendo largamente explorado, por exemplo, para a criação de simulações sofisticadas relacionadas ao clima e a terremotos, para a extração de resultados de dados científicos oriundos da astronomia ou da física, e para a criação de simulações e análise de dados relacionados à biologia [GIL *et al.*, 2010]. Os avanços computacionais necessários a quase todas as áreas científicas incluem aumento dos níveis de automação e

suporte para o desenvolvimento de sistemas cada vez mais complexos [ATKINS *et al.*, 2003; WASHINGTON *et al.*, 2005]. Neste contexto, os *workflows* vêm emergindo como um paradigma útil para a descrição, gerenciamento e compartilhamento de processos científicos complexos [GIL *et al.*, 2010].

O *workflow* é um elemento importante para a pesquisa científica. Mais especificamente em *e-Science*, temos os *workflows* científicos que são voltados para explicitar representações de processos experimentais científicos, geralmente de natureza colaborativa [FERNANDES e NOVAIS, 2007]. Para facilitar o trabalho dos pesquisadores, poupando tempo e possibilitando descobertas cada vez maiores e mais confiáveis, os processos científicos são mapeados para *workflows.* Com a utilização de *workflows,* serviços são associados a cada tarefa de experimentos *in silico*[1], por exemplo [MATTOS e MATTOSO, 2007].

Os *workflows* científicos guardam uma correlação muito estreita com os experimentos científicos propriamente ditos. Os experimentos podem ser executados inúmeras vezes com ligeiras modificações nos programas ou em parâmetros operacionais, e todo esse processo, independentemente da natureza do *workflow* é interativo e incremental, o que faz com que alterações na definição conceitual de um *workflow* científico sejam constantes [MATTOS *et al.*, 2008].

Os resultados produzidos pelos *workflows* científicos devem ser armazenados, pois são uma fonte de dados imprescindível para os cientistas. Além do armazenamento dos resultados, é importante manter o registro do processo de obtenção dos dados, assim é possível garantir a reprodutibilidade de um dado experimento ou ainda o reuso dos dados em outros experimentos. O mecanismo de registro dos resultados, das versões de programas e dos workflows propriamente dito chama-se proveniência e permite a economia de recursos (computacionais e humanos) além da otimização do tempo dos cientistas [MATTOS *et al.*, 2008].

Os *workflows* científicos podem ser considerados cruciais na interação dos cientistas com a infraestrutura computacional aplicada à pesquisa. Eles facilitam as atividades de *e-Science* permitindo que os cientistas modelem, executem, verifiquem, reconfigurem e re-executem as análises de dados científicos. Este tipo de *workflow* também pode ser visto como um novo paradigma de modelagem que integra o controle de dados e a computação orientada a processos [LUDÄSCHER, 2007].

Da perspectiva de um cientista, a integração das informações científicas obtidas a partir de diferentes experimentos é necessária para a formulação de conclusões e, envolve a definição de *workflows* e experimentos, sua execução e gerência (permitindo reuso, por

[1] A expressão *in silico* é, originalmente, usada para denotar simulações computacionais que modelam um processo natural ou de laboratório (LEMOS, 2004).

exemplo). A criação de novos modelos, a utilização de modelos já existentes, a manipulação de dados novos e antigos, a realização de análises sobre os dados e visualização dos resultados podem ser obtidos a partir da definição e execução de *workflows* científicos [PENNINGTON, 2007].

A modelagem de *workflows* científicos ainda é muito difícil, pois existe um maior domínio na modelagem de *workflows* para negócios (*business workflows*), visto que as características deste segundo tipo de *workflow* são bem diferentes das características do primeiro. A diferença mais notável entre os *workflows* científicos e de negócios é a escala. Quando comparados aos *workflows* científicos, os *workflow*s para negócios normalmente são bem menores. Os *workflows* científicos podem envolver centenas de instâncias de serviços, que terão que ser modeladas. Além disso, os *workflows* científicos, na maioria das vezes, vão executar centenas de invocações de serviços básicos e, consequentemente, vão enviar centenas de mensagens, que serão trocadas entre os serviços. Os *workflows* para negócios, normalmente, operam em pequena escala [TAYLOR *et al.*, 2006].

O processo de definição de um *workflow* científico não é regido por metodologias nem é independente de domínio de aplicação. Cada *workflow* requer uma ampla discussão entre os membros do grupo de pesquisa, sua definição geralmente envolve tomada de decisão e análises refinadas sobre cada uma das etapas. Entretanto, apesar dos *workflows* científicos possuírem características particulares, os cientistas desejam aumentar a eficácia e a eficiência dos experimentos científicos, portanto, faz-se necessário o uso de sistemas de gerência de *workflows* (SGWf) [MATTOS *et al.*, 2008].

Segundo a *WfMC* (*Workflow Management* Coallition) é possível definir um SGWf como um sistema para a definição, criação e gerência da execução de fluxos de trabalho através do uso de *softwares* capazes de interpretar a definição de processos, interagir com seus participantes e, quando necessário, invocar ferramentas e aplicações [WfMC, 1995].

Os sistemas de *workflows*, em geral, apresentam três componentes: uma plataforma de execução (*execution platform*), uma interface para definição visual (*visual design suite*), e um kit de desenvolvimento (*development kit*). A plataforma é responsável pela execução do *workflow*, que inclui: (1) invocação de serviços e gerenciamento da heterogeneidade de dados e interfaces entre múltiplas plataformas; (2) monitoramento e recuperação de falhas; (3) otimização de memória, armazenamento e execução, incluindo concorrência e paralelismo; (4) manipulação de dados, com seu mapeamento e transferência entre serviços; (5) armazenamento de informações relacionadas ao processo e à proveniência dos dados gerados; e (6) segurança e monitoramento de políticas de acesso [GOBLE e ROURE, 2009].

2.2.1 Sistemas de Gerência de *Workflows*

Sistemas de *workflow* são necessários à execução de processos longos, devem ser robustos e apresentar mecanismos de tolerância e recuperação de falhas [GOBLE e ROURE, 2009]. Os sistemas de gerência de *workflows* são pacotes de *software* que fornecem toda a infraestrutura para definir, executar e monitorar *workflows*. Eles podem ser classificados, de acordo com seu uso, em SGWf comerciais, SGWf científicos e SGWf em *Grid* [MATTOS *et al*, 2008].

O SGWf científico é voltado para aplicações científicas de quaisquer áreas de pesquisa, como por exemplo, biologia, física, química, ecologia, geologia, astronomia, entre outras. Essas aplicações são baseadas em experimentos científicos que requerem alto poder computacional. Os SGWf científicos são responsáveis por invocar as aplicações locais ou externas que participam de um *workflow* científico, passando os dados pertinentes entre elas. Geralmente eles oferecem para os usuários interfaces gráficas que permitem não só a definição conceitual do *workflow*, como também o monitoramento de sua execução [MATTOS *et al*, 2008].

Segundo [LEMOS, 2004] um SGWf científico deve: (1) incluir processos, dados e recursos mais utilizados e oferecer mecanismos de extensibilidade para acomodar novos processos, dados e recursos; (2) oferecer ferramentas para a validação do *workflow* definido pelo cientista; (3) otimizar e executar o *workflow* de acordo com a arquitetura que está sendo utilizada, permitindo intervenção e monitoramento pelo pesquisador em qualquer ponto da execução do *workflow*; (4) permitir agendamento da execução do *workflow*; (5) armazenar dados produzidos na execução do *workflow*, além de armazenar e produzir metadados, permitindo a consulta dos mesmos.

Existem alguns sistemas que tratam de gerência de *workflows* e exploração de dados científicos, sendo possível destacar o Taverna [TAVERNA, 2010], o VisTrails [VISTRAILS, 2010] e o Kepler [KEPLER, 2008]

- O Taverna é um ambiente integrado para desenvolvimento e execução de *workflows*. Surgiu com o propósito de facilitar as atividades de bioinformática capturando métodos científicos como modelos de processos formais, múltiplos e escaláveis, com suporte a grandes conjuntos de dados (*data sets*) [ALBRECHT, 2007].
- O VisTrails, que também é um sistema de gerenciamento de *workflows* científicos, foi desenvolvido pela Universidade de Utah, e oferece suporte para a exploração e visualização de dados.

- Kepler é um *software* para análise e modelagem de dados científicos, que tem como objetivo principal reduzir o esforço necessário para a criação de modelos executáveis, por meio do uso de uma representação visual de processos. Essas representações, chamadas de *workflows* científicos, mostram o fluxo de dados através de análise discreta e modelagem de componentes.

Visando possibilitar o reuso de serviços Web de natureza científica, é proposto neste trabalho um *framework* capaz de automatizar a criação de *workflows* científicos por meio da composição de serviços web semânticos. Em outras palavras, o *framework* proposto recebe a descrição semântica de um modelo de *workflow* científico, realiza a busca de serviços web semânticos a partir dessa descrição em repositórios de serviços, e compõem os serviços encontrados gerando um modelo de *workflow* em WS-BPEL. A arquitetura e o funcionamento do *framework* serão melhor detalhados no capítulo 4.

2.2.2 Repositórios

O uso de *workflows* vem se tornando uma prática cada vez mais comum no meio científico, dessa forma, os repositórios de *workflows* e serviços vêm sendo considerados mecanismos cruciais para o compartilhamento e o reuso, que são práticas muito importantes no contexto de *e-Science*.

Os repositórios permitem o compartilhamento de modelos de *workflows* e de serviços (computações) que podem ser reutilizados por outros desenvolvedores [GIL *et al.*, 2010]. Em *e-Science*, muitos repositórios distribuídos de dados e serviços são disponibilizados e mantidos por diversas instituições. Alguns exemplos são o *National Virtual Observatory* [NVO, 2010], o *Earth System Grid* [ESG, 2010], o *Biomedical Informatics Research Network* [BIRN, 2010], e o *Cancer Biomedical Informatics Grid* [caBIG, 2010].

Além de disponibilizarem dados, algumas organizações disponibilizam também algoritmos, serviços, modelos, ou códigos implementados capazes de processarem dados, que podem ser utilizados como componentes de *workflows*.

2.3 ONTOLOGIA

Segundo [ALMEIDA, 2003], o termo ontologia tem origem no grego *ontos*, que significa ser e *logos*, que significa palavra. Este termo foi introduzido na filosofia com o objetivo de

distinguir o estudo do ser humano como tal, do estudo de outros seres das ciências naturais. A origem é a palavra aristotélica “categoria”, que pode ser usada para classificar e caracterizar alguma coisa.

O uso de ontologias em Ciência da Computação foi iniciado por volta de 1991, no contexto da *DARPA Knowledge Sharing Effort*. Este projeto teve como objetivo conceber novas formas de construção de sistemas baseados em conhecimento, de modo que as bases de conhecimento não precisassem ser construídas a partir do zero, mas por componentes reutilizáveis.

As ontologias podem ser usadas para o desenvolvimento de aplicações relacionadas à gerência do conhecimento, processamento da linguagem natural, comércio eletrônico, integração de informação, recuperação de informação, integração de dados, e áreas de aplicação como bioinformática, educação e sistemas biológicos [CORCHO *et al.* 2007] [LAMBRIX *et al.* 2007].

Em Ciência da Computação, uma ontologia define uma especificação formal e explícita dos termos de um domínio e das relações entre eles, de uma conceitualização compartilhada [GRUBER, 1992; GRUBER, 1994; BORST, 1997]. Segundo [ALMEIDA, 2003], “formal” significa legível para computadores; “especificação explícita” diz respeito a conceitos, propriedades, relações, funções, restrições, axiomas que são explicitamente definidos; “compartilhado” quer dizer conhecimento consensual; e, “conceitualização” diz respeito a um modelo abstrato de algum fenômeno do mundo real.

Uma ontologia permite capturar a compreensão comum sobre objetos e seus relacionamentos, em um determinado domínio de interesse, e prover um modelo formal e manipulável desse domínio. A especificação formal do significado dos termos utilizados possibilita a criação de novos termos através da combinação dos já existentes, bem como permite a integração com outras ontologias [MATOS, 2008].

O estudo e o uso de ontologias na área de software foram popularizados com a idéia da Web Semântica introduzida por [BERNERS-LEE, HENDLER e LASSILA, 2001]. Ainda segundo [BERNERS-LEE, HENDLER e LASSILA, 2001]. Neste contexto, uma ontologia consiste de uma taxonomia e um conjunto de regras de inferência, que permitem capturar o conhecimento que não está explícito na taxonomia.

Segundo [CORCHO *et al.* 2007], o surgimento da Web Semântica tem causado uma crescente necessidade de se reutilizar conhecimento, e tem reforçado o seu potencial ao mesmo tempo. Por isso, ontologias têm um importante papel neste contexto e são utilizadas para fornecer uma forma de compreender o significado semântico das informações. A ontologia representa computacionalmente o conhecimento humano e permite o compartilhamento e o reuso desse conhecimento.

Ontologias fornecem um vocabulário comum sobre um domínio, promovem a interoperabilidade entre sistemas, servem como base para a integração e consulta de modelos de fontes de informações. Alguns benefícios de sua utilização são o reuso, o compartilhamento e portabilidade de conhecimento, a manutenibilidade, a documentação e a confiabilidade [LAMBRIX *et al.* 2007]. As ontologias podem estender os relacionamentos hierárquicos das taxonomias, permitindo relacionamentos horizontais entre os termos. Desta forma, facilitam a modelagem de requisitos de informação do mundo real. Sua expressividade é ampliada com o uso de máquinas de inferência.

A ontologia pode ser considerada uma das mais importantes dentre as tecnologias da Web Semântica por servir de alicerce para a construção de outras como, por exemplo, o serviço Web semântico. A fim de tornar possível a criação da Web Semântica o W3C vem trabalhando continuamente na criação de padrões abertos e vem incentivando seu uso tanto pela indústria como pelo meio acadêmico. Estes padrões são importantes para a integração e a interoperabilidade intra e inter processos [CARDOSO, 2007]. A linguagem padrão para a definição de ontologias, definida pelo W3C, é OWL [OWL, 2004].

A linguagem OWL foi projetada para ser utilizada por aplicações que necessitam processar o conteúdo da informação em vez de apenas apresentar informações para os seres humanos. Ela é uma revisão da DAML+OIL *Web Ontology Language* incorporando melhorias necessárias da aplicação de DAML+OIL [OWL, 2010].A OWL visa atender às necessidades de uma linguagem de ontologia para a Web, superando algumas limitações das linguagens que a precederam, como XML e RDF.

O objetivo da OWL é prover uma linguagem de ontologia que possa ser usada para descrever, de um modo natural, classes e relacionamentos entre classes em documentos e aplicações Web. Os termos usados em uma ontologia devem ser escritos de forma que eles possam ser usados por diferentes softwares [MATOS, 2008]. OWL distingue entre construtores e axiomas [SINGH e HUHNS, 2005]. Construtores OWL são as primitivas usadas para especificar novas classes e os axiomas são as primitivas para fazer asserções adicionais sobre as classes e propriedades. Os dialetos OWL provêem construtores de classes baseados em lógica descritiva. Estes construtores usam os tipos de dados definidos no XML *Schema*.

Como é baseada em lógica descritiva, OWL possibilita o uso de mecanismos de inferência, os quais permitem explicitar conhecimentos que estão implícitos em uma base de conhecimento. Dessa forma, um documento OWL não deve ser considerado apenas sob o ponto de vista de sua sintaxe, mas também de sua semântica. Isso significa que dois documentos superficialmente diferentes em termos sintáticos podem expressar o mesmo conhecimento, se eles legitimam as mesmas inferências. Segundo [SINGH e HUHNS, 2005] pensar em termos de inferências possibilita que conhecimentos de diferentes fontes sejam

colocados juntos. Modelos conceituais diferentes podem ser combinados e mapeados um em relação ao outro. Os mecanismos de inferência podem detectar potenciais inconsistências, ajudando a resolver erros.

O editor Protégé-OWL [PROTEGE, 2010] permite a construção de ontologias para a Web Semântica, em especial utilizando a linguagem OWL especificada pelo W3C. Uma ontologia desenvolvida em OWL pode incluir descrições de classes, propriedades e suas instâncias. Dada uma determinada ontologia, a semântica OWL especifica como derivar conseqüências lógicas, ou seja, fatos que não estão explicitamente descritos na ontologia, mas apresentados pela semântica. Essas implicações podem ser baseadas em um único documento ou em múltiplos documentos que foram combinados usando os mecanismos definidos pela OWL.

2.4 SERVIÇOS WEB SEMÂNTICOS

Um serviço Web é um sistema de software desenvolvido para dar suporte à interação entre máquinas por meio de uma rede. Possui uma interface descrita em um formato processável por máquina. Outros sistemas interagem com o serviço Web conforme especificado em sua descrição usando mensagens SOAP, tipicamente por meio do protocolo HTTP usando serialização XML juntamente com outros padrões Web [W3C, 2004].

Serviços Web são modulares, autodescritivos, e representam aplicações autocontidas acessíveis por meio da Internet [CURBERA, NAGY e WEERAWARANA, 2001]. Em geral, os serviços Web são descritos usando WSDL (*Web Service Description Language*) [CHINNICI *et al.*, 2007], que é uma linguagem que fornece informações operacionais sobre o serviço. Os serviços Web semânticos são serviços Web que apresentam, além das características normais dos serviços Web, uma descrição semântica de sua estrutura.

A linguagem WSDL não apresenta recursos para a descrição semântica de um serviço. Desta forma, uma solução para a criação de serviços Web semânticos é por meio do mapeamento dos conceitos presentes em uma descrição de serviço (especificação WSDL) para conceitos ontológicos. Entre os elementos declarados explicitamente na descrição WSDL que podem ser anotados com metadados referentes a conceitos presentes em uma ontologia estão operações, mensagens, precondições e efeitos. Algumas abordagens e iniciativas que visam especificar serviços Web usando semântica e ontologias incluem OWL-S [OWL-S, 2004], WSML [WSML, 2004], WSMO [WSMO, 2004], WSMX [WSMX, 2004], e WSDL-S [AKKIRAJU, *et al.*, 2006].

As abordagens de serviços Web semânticos podem servir como tecnologia de base para o suporte a outras questões relacionadas a serviços Web, tais como, por exemplo, políticas de modelagem e qualidade de serviços [MILLER *et al*., 2004; BRADSHAW *et al*., 2004]. Algumas abordagens relacionadas à computação distribuída, como as estruturas *Peer-to-Peer* e de *Grids* Computacionais (*Grid Computing*) podem ter a tecnologia de serviços Web inserida em seu contexto e, dessa forma, podem se tornar tecnologias semânticas podendo ser denonominadas *Semantic Peer-to-Peer* ou *Grids* Computacionais Semânticos (*Semantic Grid Computing*) [HAASE *et al*., 2004; POLLERES *et al*., 2005]

Existem diversas aplicações possíveis para os serviços Web semânticos, dentre elas podemos destacar aplicações em logística, turismo, trabalho colaborativo, finanças, telecomunicações, bioinformática, *business intelligence*, e sistemas de informações geográficas [STUDER *et al*., 2007].

Neste trabalho, a abordagem utilizada para a anotação semântica de serviços Web será a OWL-S [OWL-S, 2004], recomendada pelo W3C (*World Wide Web Consortium*) atualmente.

2.4.1 OWL-S

Para fazer uso de um serviço Web de maneira automática, um agente de software precisa de uma descrição do serviço interpretável por máquina, além de conhecer os meios para acessá-lo. A Web Semântica tem como objetivo permitir aos usuários localizar, selecionar, compor e executar os serviços Web automaticamente, por meio do uso de recursos semânticos. Assim, é necessário representar as descrições semânticas dos serviços a fim de que elas sejam processáveis por máquinas e possíveis de serem compartilhadas.

A OWL-S é uma ontologia de serviços por meio da qual é possível representar as descrições semânticas de serviços Web. A estrutura dessa ontologia está representada na Figura 1 e se baseia na necessidade de fornecer três tipos essenciais de conhecimento sobre um serviço [OWL-S, 2004]:

- A definição sobre **o que o serviço faz para potenciais clientes** é apresentada no *profile*, utilizado para anunciar o serviço. Nessa perspectiva, cada instância da classe de serviço apresenta um *ServiceProfile*.
- A **forma de funcionamento do serviço** é apresentada no *ServiceModel*. Instâncias da classe Serviço usam a propriedade *describedBy* para se referir ao *ServiceModel*.
- A **forma de interação do cliente com o serviço** é apresentada pelo *grounding*.

O *grounding* fornece as informações necessárias sobre protocolos de transporte. Instâncias da classe *Service* apresentam a propriedade *suports* a qual se refere ao *ServiceGrounding* correspondente.

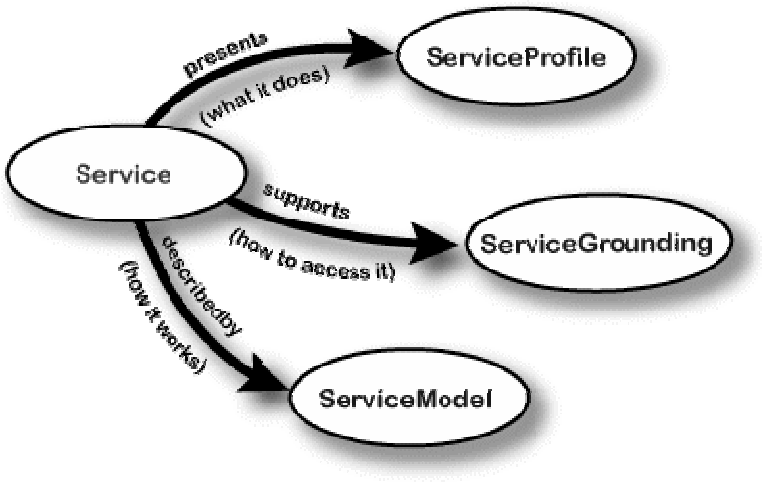


Figura 1 A ontologia *Service* [OWL-S, 2004].

A classe *Service* fornece um ponto de referência para a declaração de um serviço Web, uma instância de *Service* vai existir para cada serviço distinto publicado. A classe *Service* possui as propriedades *presents*, *describedBy*, e *supports*. Cada instancia de *Service* descreverá (*presents*) um *ServiceProfile*, será descrita (*describedBy*) por um *ServiceModel*, e suportará (*supports*) um *ServiceGrounding*.

De uma maneira simplificada, o *ServiceProfile* fornece as informações necessárias para um agente descobrir um serviço, enquanto o *ServiceModel* e *ServiceGrounding*, em conjunto, fornecem informações suficientes para um agente fazer uso do serviço, uma vez encontrado.

3. COMPOSIÇÃO DE *WORKFLOWS* CIENTÍFICOS

A experimentação computacional em larga escala tem exercido grande impacto na prática científica, produzindo avanços significativos em diversas áreas de pesquisa. Modelos computacionais de experimentos científicos permitem a execução dos experimentos de forma mais rápida e menos custosa que a execução de maneira tradicional, em laboratórios, por exemplo [GIL *et al*., 2010]. Os *workflows* são considerados um paradigma poderoso para a representação e gerenciamento de aplicações científicas complexas por fornecerem uma especificação formal para experimentos científicos. Os *workflows* fornecem meios sistemáticos e automatizados para a condução de análises de dados e aplicações, e são uma forma de capturar um processo experimental uma vez que os resultados e o próprio processo podem ser revisados, validados, repetidos e adaptados. Além disso, esse paradigma permite que os cientistas criem seus processos experimentais computacionais sem realizarem programação de baixo nível, acessando e integrando recursos disponíveis sem a necessidade de conhecer completamente seu funcionamento interno [GOBLE e ROURE, 2009].

Os *workflows* científicos permitem a representação de experimentos em diferentes níveis de abstração, e a reprodução e compartilhamento desses experimentos entre grupos de cientistas. A representação semântica pode ser considerada um nível de abstração para a representação de um *workflow*. Em geral, o processo de definição de um *workflow* científico não é regido por metodologias e depende do domínio de aplicação ao qual o *workflow* se insere. Cada *workflow* é definido a partir de ampla discussão entre os membros do grupo de pesquisa, e sua definição envolve tomada de decisão e análises refinadas sobre cada uma das etapas. Entretanto, apesar dos *workflows* científicos possuírem características particulares, os cientistas desejam aumentar a eficácia e a eficiência dos experimentos científicos [MATTOS *et al*., 2008]. Neste contexto, alguns esforços vêm sendo feitos no sentido de criar algumas abordagens para a composição de *workflows*. Entre os recursos utilizados nas abordagens existentes, podemos citar a utilização de recursos relacionados à Web Semântica.

Este capítulo apresenta os principais conceitos envolvidos no processo de composição de workflows científicos. Na seção 3.1 apresentamos os conceitos de orquestração e coreografia de serviços web, que são conceitos importantes para a fundamentação teórica de nossa abordagem. A seção 3.2 apresenta o principal formalismo utilizado neste trabalho para auxiliar na modelagem da composição dos workflows, que são as redes de petri. Na seção 3.3 detalhamos a linguagem de composição de workflows WS-

BPEL, que é utilizada para gerar o modelo de composição proposto na abordagem. Finalmente, na seção 3.4 apresentamos alguns trabalhos relacionados.

3.1 ORQUESTRAÇÃO E COREOGRAFIA

Os termos coreografia e orquestração se referem à organização de serviços Web a fim de que trabalhem em conjunto para a realização de uma tarefa comum.

A Orquestração consiste em ter um serviço Web central que controla e coordena as diferentes operações de todos os outros serviços envolvidos na operação. Deste modo, apenas o serviço central tem "conhecimento" do objetivo global que todos desempenham, sendo o responsável por definir explicitamente as operações e a ordem de invocação dos serviços envolvidos, ao passo que estes não sabem, nem precisam saber, que estão contribuindo para um objetivo maior [RIBEIRO, 2008].

Orquestração se refere a um processo executável que pode interagir com serviços Web internos ou externos ao processo. Descreve como os serviços podem interagir a nível de mensagens, incluindo a lógica da execução dos serviços e a ordem de interação entre eles. Essas interações podem abranger aplicações ou organizações e resultam em um processo. Com a orquestração, o processo é sempre controlado da perspectiva de uma das partes envolvidas [PELTZ, 2003].

A coreografia é naturalmente mais colaborativa, pois cada parte envolvida no processo descreve o papel que desempenha na interação. A coreografia inclui a sequência de mensagens que podem envolver várias partes e múltiplas fontes. Pode ser associada à troca de mensagens públicas que ocorre entre múltiplos serviços Web [PELTZ, 2003].

A coreografia não necessita de um coordenador. Isto porque cada serviço Web tem "consciência" do seu papel no processo geral – sabe quais são os serviços com os quais interage e quando executar as suas operações. Coreografia é, assim, um esforço coordenado de vários serviços Web, tendo por base a troca de mensagens. Assim, todos os participantes de uma coreografia têm "consciência" do processo de negócio, das operações que devem executar, das mensagens que devem trocar, tal como o momento em que devem fazê-lo [RIBEIRO, 2008].

Alguns requisitos técnicos são necessários para a orquestração ou coreografia de serviços Web. Para realizar a orquestração ou coreografia de serviços são necessárias uma linguagem e uma infraestrutura que ofereçam suporte a [PELTZ, 2003]:

- **Flexibilidade**. Uma das mais importantes considerações é a flexibilidade oferecida pela linguagem. A flexibilidade pode ser alcançada se houver uma

separação clara entre a lógica do processo e os serviços Web invocados. Essa separação pode ser alcançada pela manipulação do fluxo do processo.

- **Atividades básicas e estruturadas**. Uma linguagem de orquestração deve suportar atividades de comunicação com outros serviços e de manipulação da semântica do *workflow*. Podemos considerar uma atividade básica como um componente que interage com elementos externos ao processo. Atividades estruturadas gerenciam o fluxo do processo, especificando quais atividades podem ser executadas e em que ordem.
- **Composição recursiva**. Um processo simples pode interagir com diversos serviços Web. Entretanto, um processo pode ser considerado um serviço Web, permitindo que este processo seja agregado a processos mais complexos.

Tanto a orquestração como a coreografia de serviços Web devem suportar alguns requisitos básicos para o gerenciamento da integridade e consistência de suas interações. Esses requisitos incluem [PELTZ, 2003]:

- **Persistência e correlação**. A possibilidade de manter os estados entre as requisições dos serviços é um requisito importante, especialmente quando se trata da execução assíncrona de serviços Web. A linguagem e a infraestrutura devem apresentar mecanismos para gerenciar os dados de persistência e correlacionar requisições a fim de construir conversações de alto nível.
- **Manipulação de exceções e transações**. A orquestração longa de serviços Web deve gerenciar exceções e a integridade transacional. Por exemplo, recursos não podem ficar presos a uma transação que demora a ser executada.

A linguagem WS-BPEL [WS-BPEL, 2007] apresenta estruturas que possibilitam a orquestração ou coreografia de serviços Web atendendo a todos os requisitos citados. Dessa forma, essa é a linguagem utilizada para expressar o *workflow* gerado pelo *framework* proposto neste trabalho. A linguagem WS-BPEL será detalhada na seção 3.4.

3.2. REDE DE PETRI

A Rede de Petri [PETRI, 1962] é uma técnica de modelagem que permite a representação de sistemas, utilizando como alicerce uma forte base matemática. Essa técnica permite modelar sistemas paralelos, concorrentes, assíncronos e não-determinísticos [MURATA, 1989]. Podemos encontrar as Redes de Petri sendo utilizadas em áreas como economia, biologia, engenharia, computação, entre outras [DA PENHA *et al.*, 2004].

As variações de Redes de Petri tais como Redes de Petri Coloridas [JENSEN, 1990], Temporizadas [COOLAHAN e ROUSSOPOULOS, 1983] ou Estocásticas [GRANDA *et al.*, 1992] são modelos importantes e que não fazem parte da teoria original. No entanto, são representações que ganharam relevância por serem capazes de modelar sistemas de forma mais simplificada ou intuitiva ou simplesmente porque ainda não havia um modelo capaz de representar sistemas que envolviam temporizações ou comportamentos probabilísticos (estocásticos).

3.2.1. Elementos da Rede de Petri

Segundo [MURATA, 1989], as Redes de Petri são formadas por dois tipos de componentes: um ativo denominado transição, e outro passivo denominado lugar ou posição. Esses dois componentes são ligados entre si através de arcos dirigidos. Os arcos podem ser únicos ou múltiplos e interligam os lugares às transições. A realização de uma ação está associada a algumas pré-condições (condição de algumas variáveis de estado), isto é, existe uma relação entre os lugares e as transições que possibilita a realização de uma ação. De forma semelhante, após a realização de uma ação, alguns lugares terão suas informações alteradas (pós-condições). Graficamente, os lugares são representados por círculos e as transições por traços ou barras (Figura 2).

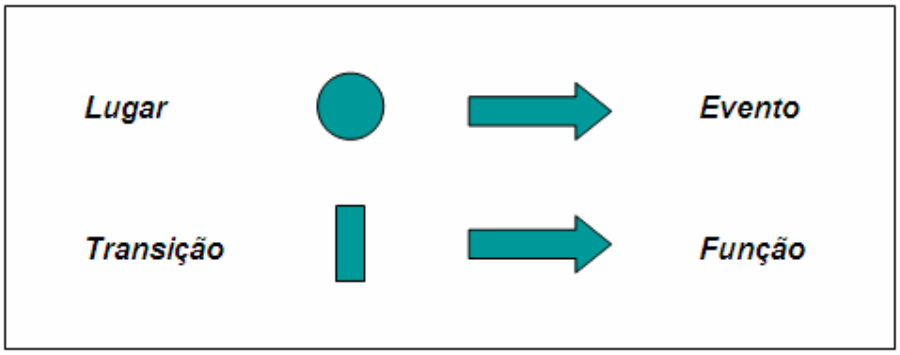


Figura 2 Elementos básicos das redes de Petri.

A teoria de Redes de Petri apresenta as seguintes propriedades: formalização matemática, representação gráfica e analisabilidade [MATTOS *et al.*, 2008]. Assim, as redes permitem a representação com facilidade de todas as relações de causalidade entre processos em situações de sequencialidade, conflito, concorrência e sincronização.

A definição formal de uma rede de Petri consiste numa tupla PN = (P, T, F, M_0), onde P é um conjunto de posições, T é um conjunto de transições, F $\subseteq$ (P x T) $\cup$ (T x P) é um conjunto de arcos (relações de fluxo) dos lugares às transições e vice-versa. M_0: P $\rightarrow$ N^+ é o marco inicial no qual para cada posição p $\in$ P existem n $\in$ N *tokens*. Os conjuntos de posições e transições são disjuntos e uma rede de Petri deve conter ao menos uma posição e uma transição: P $\cap$ T = Ø, P $\cup$ T = Ø [MATTOS *et al.*, 2008].

3.2.2. Modelagem de *Workflow*

Na modelagem de *workflow* utilizando redes de Petri, cada tarefa é representada por uma transição correspondente. Lugares representam as pré e pós-condições ou ainda, os recursos requeridos para a execução de uma determinada tarefa. Os arcos representam as relações lógicas entre as tarefas e o próprio fluxo de trabalho [SALIMIFARD e WRIGHT, 2001].

Existem algumas razões para a utilização de Redes de Petri na modelagem de *workflows*. Em [VAN DER AALST, 1998] são destacadas as seguintes:

- ***Semântica Formal.*** Um processo de *workflow* especificado em termos de uma Rede de Petri possui uma definição clara e precisa, porque a semântica de Redes de Petri e suas extensões (colorida, temporizada, hierárquica) apresentam definições formais.
- ***Natureza Gráfica.*** Rede de Petri constitui uma linguagem gráfica e, por isso, é intuitiva e fácil de aprender. A natureza gráfica também suporta a comunicação com usuários finais.
- ***Expressividade.*** Redes de Petri suportam todas as primitivas necessárias para modelar um processo de *workflow*. Todas as construções de roteamento presentes em sistemas de gerência de *workflow* podem ser modeladas. Além disso, o fato dos estados serem representados explicitamente permite a modelagem de escolhas implícitas.
- ***Propriedades.*** Existe uma grande quantidade de conhecimento, em forma de livros e artigos, que exploram essa modelagem.
- ***Análise.*** Redes de Petri são marcadas pela disponibilidade de muitas técnicas de análise, fato que, claramente, pode ser considerado uma grande vantagem em favor do uso desta modelagem para *workflows*. Estas técnicas podem ser usadas para prover propriedades e calcular as medidas de desempenho (tempo de resposta, tempo de espera, taxas de ocupação, etc.).

- ***Independência de Fornecedor.*** Redes de Petri proveem um *framework* independente de ferramenta para modelagem e análise de processos.

De acordo com [VAN DER AALST, 1998] e [RUSSEL *et al.,* 2006], um *workflow* pode ser analisado com três visões distintas: a visão do caso, a visão do processo, e a visão do recurso. A visão de caso indica que um *workflow* consiste de diferentes casos que são tratados individualmente, isto é, cada pedaço do processo é executado por caso específico. A visão de processo indica que um *workflow* consiste de diversas tarefas em alguma ordem de roteamento. A visão de recurso indica que as tarefas em um *workflow* são executadas por alguns recursos organizacionais. Um recurso pode ser uma máquina (impressora, fax etc.) ou uma pessoa (participante, empregado etc.). Tal *workflow* pode ser modelado em uma Rede de Petri, da seguinte maneira: os casos são modelados por *tokens* de alto nível, que tem uma identidade, são distinguíveis uns dos outros, e podem conter estrutura de dados; as tarefas são modeladas por transições e as condições são modeladas por lugares. Um modelo de *workflow* pode ser construído especificando como os casos são roteados juntamente com as tarefas que precisam ser executadas. De acordo com o WfMC [WfMC, 1995], são definidas as seguintes construções de roteamento: sequência, paralela, condicional e iteração. A seguir, é apresentada uma breve discussão considerando algumas dessas construções de roteamento e seu mapeamento para redes de Petri.

Além do paradigma de Redes de Petri, outros podem ser utilizados para a modelagem de *workflows*. Em [GIL *et al.*, 2010], por exemplo, os *workflows* são modelados como grafos acíclicos.

Roteamento Sequencial

É usado para lidar com relacionamentos causais entre tarefas. Considerando duas tarefas A e B. Se a tarefa B é executada depois do término da execução da tarefa A, então A e B são executadas sequencialmente. A Figura 3 mostra que o roteamento sequencial pode ser modelado através da adição de lugares. O lugar c2 modela o relacionamento causal entre a tarefa A e a tarefa B, isto é, c2 representa uma pós-condição para a tarefa A e uma pré-condição para a tarefa B.

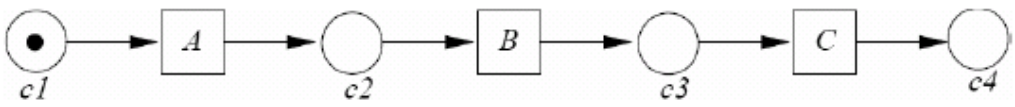


Figura 3 Roteamento sequencial.

Roteamento Paralelo

É usado em situações onde a ordem de execução é menos restrita. Por exemplo, duas tarefas B e C precisam ser executadas, mas a ordem de execução é arbitrária. Para a modelagem em Redes de Petri são construídos dois blocos: o *AND-split* e o *AND-join*. A Figura 4 exibe a rede do roteamento paralelo, onde a execução do *AND-split* habilita ambas as tarefas B e C, enquanto que *AND-join* indica que D só estará habilitado após a execução de ambos, B e C, isto é, D é usado para sincronizar dois subfluxos. Como resultado, as tarefas B e C são executadas em paralelo.

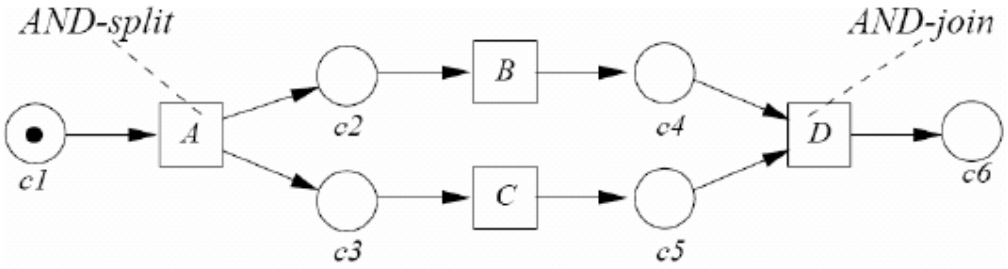


Figura 4 Roteamento paralelo.

Roteamento Condicional

É usado para permitir que um roteamento possa variar entre casos. Para modelar uma escolha entre duas ou mais alternativas, dois blocos de construção são usados: o *OR-split* e o *OR-join*. A Figura 5 ilustra a situação onde a tarefa A é seguida ou pela tarefa B ou pela tarefa C, ou seja, é feita uma escolha entre B e C. A execução de uma dessas duas tarefas é seguida pela execução da tarefa D.

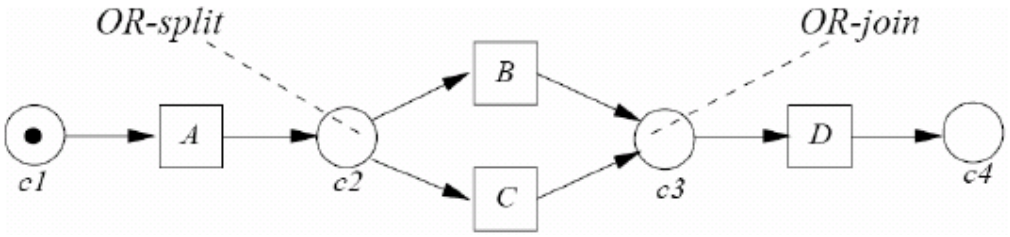


Figura 5 Roteamento condicional.

3.3. WS-BPEL

A *Web Services Business Process Execution Language* (WS-BPEL) [OASIS, 2007], quando foi criada, com a denominação BPEL4WS, veio para substituir duas linguagens para *workflow* criadas anteriormente pela IBM e pela Microsoft. A *Web Services Flow Language* (WSFL) [WSFL, 2007], que foi criada pela IBM, disponibilizava uma visão orientada a grafos para definir um *workfllow.* A XLANG [XLANG, 2007], criada pela Microsoft apresentava uma visão mais voltada para a estrutura orientada a blocos [TAYLOR *et al.*, 2006]. A continuidade de BPEL4WS ficou sob responsabilidade do órgão de padronização OASIS (*Organization for the Advancement of Structured Information Standards*) e assim criou-se o padrão WS-BPEL (*Web Services Business Process Execution Language*) que atualmente está na versão 2.0 [JORDAN *et al.*, 2007; TAYLOR *et al.*, 2006].

Os conceitos básicos de WS-BPEL podem ser aplicados de forma abstrata ou executável. Um processo WS-BPEL abstrato é um processo parcialmente especificado, que não se destina a ser executado e que deve ser expressamente declarado como “abstrato”. Considerando que os processos executáveis são totalmente especificados e, portanto, podem ser executados, um processo abstrato pode esconder alguns dos detalhes operacionais necessários concretos expressos por artefatos executáveis.

Todas as construções de processos executáveis estão disponíveis para processos abstratos, consequentemente, processos WS-BPEL executáveis e abstratos têm o mesmo poder de expressão. WS-BPEL define um modelo e uma gramática para descrever o comportamento de processos baseados em interações entre o processo e os seus parceiros. A interação com cada parceiro ocorre através das interfaces dos serviços Web, e a estrutura das relações ao nível da interface é encapsulado no que é chamado um *partnerLink*. O processo WS-BPEL define como as múltiplas interações com esses parceiros são coordenadas para alcançar o objetivo do processo assim como a lógica necessária para essa coordenação. A WS-BPEL também introduz mecanismos sistemáticos para lidar com exceções e falhas de processamento. Um processo WS-BPEL é uma definição reutilizável que pode ser utilizada de diferentes formas em diferentes cenários [OASIS, 2007].

Dois termos comumente utilizados na coordenação da execução de serviços são orquestração (*orchestration*) e coreografia (*choreography*). Embora esses termos possam parecer sinônimos à primeira vista (e muitos os tratam desta forma), na realidade são conceitos diferentes e que se complementam, sobrepondo-se apenas em alguns pontos [PELTZ, 2003]. WS-BPEL pode ser considerada uma linguagem de orquestração.

O termo orquestração é utilizado para descrever processos executáveis que interagem com serviços internos e externos, determinam regras de negócio e a ordem de

execução das tarefas de processos geralmente longos, transacionais e de múltiplos passos. Coreografia, por outro lado, trata das interações entre serviços envolvendo diversas partes, dando maior atenção às regras de interação que às partes estão submetidas, incluindo dependências de fluxo de execução, dependências no fluxo de dados, correlação de mensagens, restrições de tempo e regras de controle de transação [BARROS *et al.*, 2005]. Pode-se dizer que a orquestração tem o foco na execução de um processo do ponto de vista das ações internas de uma das partes, já a coreografia tem o foco no processo como um todo.

A WS-BPEL é uma linguagem que possui grande parte de seus conceitos semelhantes ao da teoria de *workflows*, já que esta linguagem é focada na execução de processos sequenciais e paralelos com múltiplos passos, assim como na teoria de *workflows*. O metamodelo da WS-BPEL tem como elemento central o Processo (*Process*), onde se relacionam outros elementos como Atividade (*Activity*), Correlação (*Correlation*), Compensação (*Compensation*) e Parceiro (*Partner*). É na definição do elemento *Activity* que se define o fluxo de execução de um processo através de elementos como *If*, *While*, *Pick*, *Flow*, e *Sequence*, e o fluxo de mensagens através dos elementos *Receive*, *Reply*, e *Invoke*.

A WS-BPEL permite ainda a definição de dois tipos de processos, os abstratos e os executáveis. A especificação define que ambos os processos compartilham dos mesmos recursos da linguagem, porém os abstratos permitem que determinadas etapas de um processo sejam omitidas, assumindo assim um papel de descritor ou modelo de processo [OASIS, 2007]. É possível descrever um modelo de *workflow* WS-BPEL abstrato, que é um modelo que destaca os comportamentos mais importantes do *workflow*, sem especificar todos os detalhes. A intenção é permitir uma definição a partir das características públicas visíveis do *workflow*, escondendo detalhes que podem ser variáveis em implementações diferentes do mesmo *workflow*. Isso funciona como uma interface em linguagens de programação. O modelo abstrato será então implementado por um modelo de *workflow* WS-BPEL que irá conter todos os detalhes e será chamado modelo de *workflow* WS-BPEL executável [TAYLOR *et al.*, 2006].

WS-BPEL lida essencialmente com aspectos funcionais do processo de negócio:

- Fluxos de controle (*branch*, *loop*, *parallel*);
- Comunicações assíncronas;
- Não determinismo;
- Unidades de trabalho, falhas e compensações.

WS-BPEL já é o padrão estabelecido como a linguagem para a especificação de processos de negócio, além disso, o WS-BPEL vem ganhando espaço na especificação de processos científicos [EMMERICH *et al.*, 2006].

É também uma linguagem para a especificação de processos que descreve processos assíncronos de longa duração e suas interações. De forma simplificada, a WS-BPEL pode ser vista como uma linguagem de programação que é expressa em XML e que provê as construções de controle de fluxo para a combinação de serviços WSDL. A linguagem é focada em serviços Web, entretanto outros elementos podem ser incorporados a um fluxo de processos WS-BPEL (por exemplo, métodos Java, EJBs etc.).Um processo descrito em WS-BPEL representa um conjunto de comportamentos públicos observáveis, inclui informações de quando esperar uma mensagem, quando enviar uma mensagem e quando tratar operações que falharam. A especificação WS-BPEL baseia-se em três conceitos [OASIS, 2007]:

- Protocolos de descrição de processos invariavelmente expressam comportamentos dependentes de dados. Neste caso a linguagem necessita de construtores condicionais e de expiração de tempo (*time-out*);
- A habilidade de especificar condições de exceção e suas conseqüências, tais como seqüências de recuperação, são tão importantes para a linguagem de processos como as seqüências que tratam fluxos corretos de execução;
- Interações de longa duração incluem com freqüência unidades de trabalho múltiplas, aninhadas e cada uma com seus próprios requisitos de dados.

Na especificação WS-BPEL, todas as interações internas a um *workflow* ocorrem através de interfaces de serviços Web definidas em WSDL. Dessa forma, os processos podem apresentar dois comportamentos: (1) o processo interage com o serviço Web através da interface dos serviços Web descrita em WSDL e (2) o processo define a si mesmo como um serviço Web descrevendo sua interface por meio de WSDL [LEYMANN *et al.*, 2007].

Como WS-BPEL foi desenvolvida para trabalhar com serviços Web, cada *workflow* WS-BPEL pode ser considerado um serviço Web. Isso faz WS-BPEL se ajustar perfeitamente ao *middleware* de serviços Web e torna fácil a composição de *workflows* hierárquicos. Um *workflow* WS-BPEL é um serviço Web que pode ser usado dentro de um outro *workflow* WS-BPEL que, por sua vez, também pode ser usado dentro de um outro *workflow* WS-BPEL e assim por diante. Essa estrutura permite a composição de um *workflow* a partir de vários outros [TAYLOR *et al.*, 2006]. Na linguagem WS-BPEL, o resultado de uma composição de serviços Web é chamado processo (*Process*), os serviços participantes são chamados de parceiros (*Partners*) e a troca de mensagens ou a transformação de resultados intermediários é chamada de atividade (*Activity*).

A especificação de WS-BPEL oferece um vasto conjunto de ferramentas para manipulação de mensagens XML, extração e combinação de partes de mensagens XML, descrição e validação do conteúdo dessas mensagens e roteamento de mensagens para serviços Web [TAYLOR *et al.*, 2006]. WS-BPEL define processos de negócios usando uma

linguagem baseada em XML e não se concentra em representações gráficas dos processos nem especifica uma metodologia de definição particular para os processos [LEYMANN *et al*., 2007].

3.3.1. Estrutura da Linguagem WS-BPEL

A especificação WS-BPEL possui atividades básicas e estruturadas. Uma atividade básica é uma instrução que não interfere no fluxo de execução e executa uma operação simples, como invocar um serviço Web ou fazer tratamento de dados. Uma atividade estruturada gerencia o fluxo do processo, especificando a ordem de execução, como por exemplo, laços, execução sequencial, execução em paralelo e desvios condicionais. No Quadro 1 são apresentadas atividades de WS-BPEL separadas em atividades básicas e estruturadas.

Quadro 1 Atividades WS-BPEL.

Atividades básicas	
invoke	Invoca um serviço Web
reply	Envia uma mensagem de resposta a um cliente
receive	Recebe uma mensagem de um cliente
assign	Comando de atribuição
throw	Sinaliza explicitamente um erro interno
wait	Especifica um tempo de espera
empty	Uma atividade que não tem nenhum efeito
rethrow	Relança um erro que foi capturado
Atividades estruturadas	
sequence	Atividades dentro de um bloco *sequence* são executadas sequencialmente
if	Desvio condicional
while	Execução iterativa (laços)
repeatUntil	Execução iterativa (é executado ao menos uma vez)
pick	Associa ações a determinados eventos
flow	Atividades dentro de um bloco *flow* são executadas em paralelo
foreach	Execução

A linguagem tem um forte conjunto de estruturas de controle (laços, condições etc.) e um bom suporte para capturar e manipular exceções (falhas) e reversão de mudanças pelo uso de compensações. Compensações são importantes, em particular, para *workflows* com execuções longas, que necessitam de operações de *undo* (desfazer) em casos de erros irrecuperáveis em serviços que estão sendo usados pelo *workflow*, em que

consistências globais têm que ser restauradas antes que o *workflow* seja finalizado [TAYLOR *et al*., 2006].

Um processo WS-BPEL é normalmente divido em duas seções: uma seção de declaração e uma seção contendo os processos de "atividades". Envolto em um elemento XML <process> que identifica o nome do processo em si.

As declarações globais são tipicamente a primeira seção de um processo que contém as declarações globais do *workflow*, isto inclui os serviços Web que serão utilizados, os parceiros do *workflow* (<partnerLinks>), quais variáveis serão utilizadas, associadas a *tag* <variable>, além das definições dos tipos complexos que serão utilizados, *tag* <types>.

As variáveis *inputClient* e *outputClient* são necessárias para a entrada e saída do *workflow*. Outros construtores de alto nível também são declarados nesta primeira seção. São eles os construtores de erro global, expressos na *tag* <faultHandlers>, e os manipuladores de falha de transação global, expresso por <compensationHandlers>.

A segunda parte do processo WS-BPEL, que contém a lógica de processamento e a área de definição do processo, contém os passos com chamadas a serviços Web que são interconectados para compor um processo útil ao usuário. Nesta seção existem dois tipos de atividades. O primeiro tipo é referente às atividades de processo e atividades de dados. As atividades de processo atuam na chamada e recepção dos serviços Web, como exemplos destas primitivas têm-se as *tags*: <invoke>, <receive> e <reply>. Ainda nas atividades de processo podemos controlar o mesmo, através das *tags*: <wait> e <terminate>. Para a manipulação de dados temos a *tag* <assign>.

As atividades estruturadas definem o controle programático sobre quais passos serão executados no *workflow*, incluem o tratamento de condições com a *tag* <case>; execução de laços, <while>; construtores de execução em paralelo como o <flow>; e construtores de execução sequencial com o <sequence>.

A Figura 6 abaixo ilustra um código WS-BPEL para a invocação do serviço *entrezSpell* e *entrezFetch*. O primeiro retorna sugestões de escrita para um dado termo biológico; o segundo retorna uma lista de registros dado um identificador primário. Para a execução de cada serviço temos as *tags* <invoke> e <receive> dentro de uma seqüência (<sequence>), garantindo que eles serão executados um após o outro. Mas as duas seqüências de atividades estão envoltas por uma atividade de fluxo (<flow>), o que permite que as subseqüências sejam executadas em paralelo.

```
<flow>
      <sequence>
        <invoke name="invokeEntrezFetch"
                partnerLink="EntrezFetchService"
                portType="services:eUtilsServiceSoap"
                operation="run_eFetch"
                inputVariable="queryKey"/>
        <receive name="receive_invokeEntrezFetch"
                partnerLink="EntrezFetchService"
                portType="services:eUtilsServiceSoapCallback"
                operation="run_eFetch_MS"
                variable="outputFetch"/>
      </sequence>
      <sequence>
      <invoke name="invokeEntrezSpell" partnerLink="EntrezSpellService"
                portType="services:eUtilsServiceSoap"
                operation="run_eSpell"
                inputVariable="term"/>
        <receive name="receive_invokeEntrezSpell"
                partnerLink="EntrezSpellService"
                portType="services:eUtilsServiceSoapCallback"
                operation="run_eSpell_MS"
                variable="outputSpell"/>
      </sequence>
    </flow>
```

Figura 6 Código WS-BPEL de execução paralela [ENTREZ, 2008]

Todas as mensagens em WS-BPEL estão contidas em variáveis. As variáveis são passadas entre atividades WS-BPEL. Para copiar e modificar o conteúdo de variáveis pode ser usada a atividade de assinalar (<assign>). É suportada a linguagem XPath para selecionar e modificar o conteúdo XML [TAYLOR *et al.*, 2006].

A WS-BPEL apresenta ainda um mecanismo de identificação para instâncias de processos, que permite a definição de identificadores de instâncias no nível do envio de mensagens da aplicação. Além do controle de identificadores, também oferece suporte à criação e destruição de instâncias de processos por meio de um mecanismo básico de controle de ciclo de vida [TAYLOR *et al.*, 2006].

A linguagem apresenta o recurso de definir *workflows* baseados em grafos. É fácil iniciar diversas atividades em paralelo com <flow> e WS-BPEL permite definir dependências, de forma gráfica, entre as atividades. Cada atividade (um nó no grafo) pode ter diversos *links* de chegada e de saída. Para uma atividade iniciar sua execução, todos os *links* de chegada devem estar habilitados. Quando uma atividade é finalizada, todos os seus *links* de saída ficarão habilitados. Sendo estes *links* de entrada de outras atividades, estas atividades poderão ser iniciadas. Este recurso permite que sejam construídos diversos tipos de grafos em WS-BPEL e a parte mais interessante é que WS-BPEL permite ao programador misturar abordagens estruturadas e gráficas em um mesmo *workflow* [TAYLOR *et al.*, 2006].

A WS-BPEL não tem *loop* paralelo. Este fator é particularmente importante quando se trata de uma aplicação científica. Se o número de iterações é constante, é possível usar

<flow> para iniciar múltiplas atividades em paralelo, mas essa abordagem não funciona se o número de iterações depende de um dado de entrada do *workflow*, por exemplo. Um *loop* paralelo pode ser simulado com invocações não bloqueantes de um serviço Web (que será um *sub-workflow* WS-BPEL). Mas esse tipo de invocação é difícil de definir e, em geral, estabelecer canais de comunicação entre o *sub-workflow* e o *workflow* principal, assim como detectar se todos os *sub-workflows* foram finalizados com sucesso, pode ser uma tarefa difícil [TAYLOR *et al*., 2006].

Existem diversas ferramentas que podem ser usadas para definir *workflows* WS-BPEL. Elas podem variar de editores de XML simples ou mais sofisticados até ferramentas gráficas com uma interface para compor *workflows* através da conexão de serviços Web por meio de grafos, escondendo do usuário o texto XML dos processos de WS-BPEL e gerando o XML automaticamente quando necessário. Como as ferramentas gráficas operam em um alto nível de abstração elas, normalmente, suportam apenas um subconjunto das funcionalidades de WS-BPEL. Essas funcionalidades costumam estar relacionadas a um determinado grupo de usuários ao qual a ferramenta se aplica [TAYLOR *et al*., 2006].

A WS-BPEL vem se tornando o principal padrão para *workflows* baseados em serviços Web e pode ser integrada à arquitetura Web e a portais. Esta linguagem é uma escolha viável para *workflows* científicos. WS-BPEL suporta extensões como BPELJ, que permite interações entre *workflows* WS-BPEL e componentes Java, e é possível que algumas das extensões existentes se tornem padrões para *workflows* WS-BPEL científicos [TAYLOR *et al*., 2006].

Por apresentar características muito ligadas a *workflows* científicos como, por exemplo, recursos para o tratamento de exceções e facilidades para representação de *workflows* complexos, esta linguagem vem sendo amplamente utilizada para a representação deste tipo de *workflow.* WS-BPEL é uma boa linguagem para compartilhamento de *workflows* entre diferentes projetos, o que facilita atividades de *e-Science*, fortalecendo a idéia de que esta linguagem pode ser usada no apoio à pesquisa científica. A WS-BPEL pode ser vista como uma ferramenta que permite que um *workflow* importe e exporte outros *workflows* WS-BPEL em projetos científicos [TAYLOR *et al*., 2006]. Existem, atualmente, muitos esforços no sentido de desenvolver ferramentas que ofereçam facilidades ao desenvolvimento de *workflows* científicos.

No contexto deste trabalho, a linguagem WS-BPEL é utilizada para a definição de *workflows* científicos compostos por serviços Web semânticos. Um documento WS-BPEL que representa um *workflow* científico é apresentado como resultado da execução do *framework* proposto neste trabalho.

3.3.2. Ferramentas para Definição e Execução de *Workflows* WS-BPEL

Os *engines* de execução de modelos de *workflow* WS-BPEL são na realidade a versão atualizada, para serviços Web, dos sistemas de gerenciamento de *workflow*. Estes sistemas aderem às especificações da *Workflow Management Coalition* [WfMC, 1995] e possuem arquitetura baseada em seus padrões.

Alguns *engines* que se destacam comercialmente e academicamente para a execução de *workflows* em WS-BPEL são:

- **ActiveBPEL:** O ActiveBPEL [ActiveBPEL, 2006] tem sua distribuição gratuita. Os principais benefícios associados ao ActiveBPEL incluem: completude, pois implementa toda a especificação BPEL 1.1; força da indústria, pois suporta características como persistência de processos, notificação de eventos e console das APIs; e trilha de crescimento, o modelo de desenvolvimento gratuito permite a evolução rápida através das contribuições da comunidade. O *engine* do ActiveBPEL roda sobre um *container* de *servlet* padrão tal como o Tomcat. Isto significa que os servidores de aplicações J2EE podem executar o ActiveBPEL. O *engine* também acompanha uma console de administração que permite redefinir parâmetros do *engine* assim como verificar informações dos *workflows* disponíveis. As bases de dados de persistência dos processos BPEL são implementados por um SGBD em memória que acompanha o ActiveBPEL.
- **JBoss jBPM:** O JBoss jBPM [JBoss jBPM, 2009] é uma ferramenta para a construção de *workflows* gratuita que utiliza a infraestrutura J2EE. Apesar do *engine* ser suportado por qualquer infraestrutura J2EE, ele possui uma integração natural com o JBoss. O JBoss utiliza uma linguagem própria para a construção de *workflows*, a jPdl (*JBoss jBPM Process Definition Language*). A jPdl se propõe a ser uma linguagem de especificação de processos flexível o suficiente para lidar com pequenos *workflows* e também *workflows* corporativos e intercorporativos. Para atender as ligações entre corporações a linguagem BPEL pode ser utilizada. O *engine* de processo mantém o controle dos estados e das variáveis de todos os processos ativos. O *engine* de processo é composto de um manipulador de solicitações (*request handler*) que provê a infraestrutura de comunicação que aloca as tarefas para os processos adequados. Os serviços de interação (*Interaction Service*) expõem as aplicações existentes como funções ou dados nos processos finais. O gerente de estados (*state manager*) manipula os processos em execução no *engine* alocando os registros e dados além de preparar os acessos às bases de dados

para as ações resultantes dos processos. O monitor de processo (*monitor process*) provê a visibilidade dos estados finais dos processos em que os usuários e processos estão interagindo. A linguagem de processos é o núcleo do *engine* e se baseia em grafos diretos. Sobre este núcleo do *engine* de manipulação de grafos outros padrões são suportados: BPEL, BPELJ, BPML, ebXMLs, BPSS e WfMC's XPDL.

- **Microsoft BizTalk Server:** As capacidades de orquestração do BizTalk [BizTalk, 2009] são focadas na comunicação entre sistemas, suportando processos de negócio que dependem da integração de diversos softwares. Possui como principais focos a integração de arquiteturas corporativas (EAI), B2B e gerência de processos de negócio (BPM). O suporte a serviços Web do BizTalk é provido pela arquitetura ASP.NET que se beneficia da infraestrutura Microsoft para a construção de *workflows* em conformidade com a WCF (*Windows Communication Foundation*) para a criação de aplicações orientadas a serviço. Além disso, suporta especificações padrão da indústria como: WSSecurity, WS-ReliableMessaging e WS-AtomicTransaction. O *engine* do BizTalk é calcado em duas características básicas: uma ferramenta para especificar e implementar a lógica do *workflow* e diversos mecanismos para a comunicação entre as aplicações participantes. Para implementar esta diversidade de interfaces de comunicação o BizTalk define um conjunto de adaptadores (*adapters*): adaptador de serviço Web, que permite a comunicação SOAP/http; adaptador de arquivo, que permite a leitura e escrita de arquivos no sistema de arquivo Windows; adaptador HTTP, que permite o envio e recebimento de informação usando o http; adaptador MSMQ, usado para a comunicação utilizando o Microsoft Message Queuing; adaptador WebSphere MQ, utilizado para a comunicação como o IBM WebSphere Message Queuing; adaptadores SMTP, que permitem o uso de mensagens usando o SMTP; adaptador POP, que habilita o recebimento e envio de mensagens através do POP3; e adaptador SQL, que permite a leitura e escrita em um banco de dados Microsoft SQL Server. Ao contrário dos outros *engine* BPEL o BizTalk não utiliza a infra-estrutura J2EE como suporte ao *engine.*
- **IBM WebSphere Process Server:** O IBM WebSphere Process Server [WebSphere, 2009] é construído em cima da infraestrutura SOA da IBM, desta forma incorpora todas as características padrões para integrar aplicações e serviços. Destacam-se na implementação do *engine* de execução de *workflows* o suporte a serviços Web sobre diversos protocolos, como SOAP/http e SOAP/JMS; o suporte a WSDL 1.1 e aos padrões de segurança e transações,

WS-Security e WS-Atomic Transactions; o suporte aos protocolos de mensagens JMS, WebSphere MQ e permitindo a comunicação *multicast*; inclui pacotes para construções de clientes em C/C++ e .NET além de Java. O seu *engine* possui como destaque a portabilidade de instalação em diversos sistemas operacionais. Este pode ser instalado em servidores com diferentes sistemas operacionais. Junto com o *engine*, o IBM WebSphere integra um editor de processos visual para a definição dos *workflows*. Este editor incorpora um depurador para acompanhar cada passo da execução de um *workflow* BPEL, permitindo assim, a interação humana nos processos em execução, que podem ser interrompidos e persistidos.

- **Oracle BPEL Process Manager:** O Oracle BPEL Process Manager [ORACLE, 2009] tem como foco principal a integração de serviços de forma colaborativa e transacional para processos de negócio. Ele incorpora suporte nativo ao BPEL; integra serviços Web, mensagens JMS, componentes JCA e tarefas do usuário; permite monitoração e auditoria; apresenta compatibilidade com a especificação BPEL4WS 1.1; permite a troca de mensagens síncronas e assíncronas. O núcleo do *engine* BPEL da Oracle agrega outras características que tornam a ferramenta bastante robusta. A primeira característica é a "desidratação do contexto" de execução do *workflow* que é a persistência dos estados da execução do *workflow*. Esta característica torna a execução do *workflow* mais robusta, pois caso haja falha na execução de algum processo, o *workflow* pode ser reiniciado a partir do ponto onde houve a falha. Outra característica não menos importante é a capacidade de manipular arquivos XML grandes. Esta característica introduz um componente de gerenciamento de banco de dados ao *engine*, expandindo as capacidades da ferramenta. Por fim, a infraestrutura do Oracle BPEL ainda conta com um servidor UDDI para facilitar a localização de serviços Web conhecidos. Como infraestrutura de suporte ao *engine* Oracle BPEL, as possibilidades são os servidores de aplicação J2EE mais tradicionais, tal como, o Oracle AS, o Weblogic da BEA, IBM Websphere Application Server e o JBoss. A possibilidade de instalação do *engine* em diversos servidores de aplicação aumenta o espectro de instalações existentes em que a ferramenta pode ser instalada.

No conjunto acima podemos separar os *engines* em dois grandes grupos: comerciais e gratuitos. Entre os comercias temos o BizTalk, IBM WebSphere Process Server e o Oracle BPEL PM. Os gratuitos são o ActiveBPEL e o JBoss jBPM; que são gratuitos para o uso, sob a Licença LGPL que permite acesso ao fonte do *engine*, mas não restringindo o uso deste dentro de aplicações comerciais.

3.4. ABORDAGENS PARA COMPOSIÇÃO DE WORKFLOWS CIENTÍFICOS

Diversos esforços vêm sendo realizados no sentido de facilitar a criação (composição) de *workflows* pelos cientistas. Algumas ferramentas vêm sendo desenvolvidas com o objetivo de automatizar este processo.

A maioria dos esforços relacionados a serviços Web vem sendo feitos no sentido de automatizar sua utilização. Os esforços iniciais fracassaram em suas promessas de automatizar a interação e composição dos serviços dinamicamente. A razão para esse fracasso foi que a tecnologia de serviços Web não oferecia meios suficientes para descrever os serviços de maneira que permitisse a construção de mecanismos genéricos de descoberta, execução e composição dos mesmos [CHARIF e SABOURET, 2006]. Porém o uso da Web semântica juntamente com os serviços Web veio para modificar este cenário. A Web semântica transforma a Web em um repositório de dados legíveis e inteligíveis por computador enquanto os serviços Web fornecem as ferramentas para o uso automático desses dados.

O trabalho de [GIL *et al*., 2010] propõe um algoritmo para a geração automática de um *workflow* a partir de modelos (*templates*) de *workflows* de alto nível disponíveis em um repositório de *workflows*. Nessa abordagem, o usuário pode definir seu *workflow* por meio da seleção de um modelo do repositório e da customização desse modelo através da adição de restrições nos resultados desejados ou nos recursos a serem utilizados. Esse algoritmo proporciona ao usuário uma grande flexibilidade em termos do nível de abstração da informação que ele fornece, assim como da quantidade de informações fornecidas.

Dada a definição de um *workflow*, o algoritmo deve: 1) interpretar a definição do *workflow* como um conjunto de restrições nos componentes e nos dados do modelo utilizado para sua definição; 2) consultar um catálogo externo de componentes para analisar as implicações dessas restrições nos componentes individuais, e um catálogo externo de dados para analisar as restrições de dados; 3) analisar o *workflow* como um todo por meio da propagação de restrições ao longo de sua execução; 4) considerar conjuntos de dados, componentes e escolhas de parâmetros alternativos por meio da criação de possíveis *workflows* candidatos; 5) gerar *workflows* prontos para execução. Os componentes do *workflow* gerado podem ser serviços Web ou códigos implementados disponíveis para execução remota. O *workflow* é estruturado como um grafo acíclico.

O trabalho de [CHARIF e SABOURET, 2006] propõe uma abordagem para a seleção e composição dinâmica de serviços. Os serviços são selecionados e compostos de acordo com as restrições expressas por um usuário. Essa abordagem é baseada em

sistemas multi-agentes (MAS – *Multi-Agent Systems*) e pode ser decomposta em três passos:

- Formalizar e decompor as solicitações do usuário em uma ou mais solicitações interdependentes, que são enviadas a um agente mediador responsável pela descoberta de serviços candidatos;
- Descobrir e recuperar os serviços candidatos a partir de um registro usando palavras chave extraídas das solicitações do usuário;
- Selecionar e compor os serviços por meio da interação entre o agente mediador e os agentes que provêem os serviços candidatos até que as solicitações do cliente sejam satisfeitas.

O trabalho de [SHIN *et al.*, 2009] propõe um método de composição de serviços Web que busca garantir a corretude funcional e a complexidade temporal da composição. Nesse trabalho, a semântica funcional é definida como a descrição exata da funcionalidade de um serviço. A funcionalidade do serviço é representada por um par constituído por uma ação e o objeto alvo dessa ação.

As informações sobre serviços são organizadas e armazenadas em um modelo gráfico de duas camadas proposto. Dada a requisição de um usuário, pacotes de composições são buscados no modelo gráfico e serviços compostos são construídos a partir dos pacotes descobertos. Para avaliar o desempenho do método proposto, o trabalho apresenta a realização de experiências para a avaliação da velocidade e exatidão da composição.

O trabalho de [SÁNCHEZ *et al.*, 2009] tem como objetivo principal a obtenção de um sistema que, a partir do emprego de agentes inteligentes e do uso intensivo de ontologias, melhore o rendimento no consumo de serviços Web e incremente a automatização das tarefas de gestão dos mesmos. Em outras palavras, o objetivo deste trabalho é a obtenção de um ambiente de execução de serviços Web semânticos baseado na tecnologia de agentes.

O sistema apresenta independência do domínio e da aplicação, atendendo assim a diversas aplicações em domínios distintos. Uma das tarefas fundamentais da plataforma é a gestão e manejo dos serviços Web semânticos realizando um conjunto de tarefas básicas: descoberta, seleção, composição e invocação de serviços. O funcionamento do sistema se dá completamente por meio da ação de agentes, sendo constituído por sete agentes, quatro bases de conhecimento que contêm as ontologias necessárias ao funcionamento da plataforma e três interfaces para relacionamento com entidades externas: consumidores de serviços, provedores de serviços, e desenvolvedores de software.

O trabalho de [KARAKOC e SENKUL, 2009] apresenta uma abordagem de programação baseada em restrições para a resolução do problema da composição de

serviços Web. A principal propriedade dessa abordagem é a habilidade de expressar e resolver um rico conjunto de restrições para modelar requisitos de qualidade de serviço em serviços Web compostos, e prover um *framework* unificado para realizar esta tarefa.

Para modelar e satisfazer as restrições do usuário, técnicas de gerenciamento de alocação de recursos de *workflows* são adaptadas para a seleção de serviços. A abordagem proposta é realizada em um *framework* de composição de serviços Web chamado *Composite Web Service Framework* (CWSF). Usando esse *framework*, o problema da composição de serviços Web é transformado em um problema de satisfação de restrições, e os planos de composição são gerados por um solucionador de restrições (*constraint solver*).

O CWSF utiliza informações semânticas dos serviços Web para realizar a sua composição. Técnicas semânticas são utilizadas para descobrir serviços, selecioná-los e realizar o controle de tarefas de composição do processo. Dessa forma, a seleção de serviços que satisfazem as restrições não se dá a partir de análise sintática e sim a partir do uso de modelos semânticos dos serviços. A saída do *framework* (*output*) é um esquema da composição de serviços Web na qual o tempo e a forma de execução dos serviços estão definidos.

3.4.1. Comparação com Composer-Science

O trabalho de [GIL *et al*., 2010] não faz uso de tecnologias semânticas para obter um *workflow* executável ao final do processo de composição. Desta forma, esse trabalho difere do Composer-Science no sentido de que a o trabalho detalhado nesta dissertação propõe o uso de tecnologias semânticas, como serviços Web semânticos e ontologias, para a composição do *workflow* e gera um modelo de *workflow* em uma linguagem padrão (no caso, WS-BPEL) que pode ser executado em qualquer *engine* existente capaz de interpretá-la. O uso de tecnologias semânticas auxilia na descoberta automática de serviços Web para compor o *workflow*, reduzindo, dessa forma, o custo relacionado ao tempo gasto para a criação do *workflow* científico. A geração de um modelo em uma linguagem padrão confere maior facilidade de uso dos resultados obtidos pelo pesquisador, por existirem ferramentas disponíveis que interpretam a linguagem e documentação disponível referente à mesma.

O Composer-Science utiliza o paradigma de Redes de Petri como base para a geração de um modelo de *workflow*, enquanto o trabalho de [GIL *et al*., 2010] gera um *workflow* como um grafo acíclico.

O trabalho de [CHARIF e SABOURET, 2006] é semelhante ao Composer-Science no sentido de que propõe uma abordagem para a seleção e composição dinâmica de

serviços. O Composer-Science faz uso de tecnologias semânticas como serviços Web semânticos e ontologias para descobrir serviços que se enquadrem na composição de um *workflow*, enquanto o trabalho de [CHARIF e SABOURET, 2006] realiza a busca de serviços a partir de palavras-chave fornecidas pelo usuário, porém, sem utilizar recursos semânticos que ampliem o poder da descrição do serviço pelo usuário e da busca de candidatos.

O trabalho de [SHIN *et al.*, 2009] propõe um método de composição de serviços Web que busca garantir a corretude funcional e a complexidade temporal da composição, utilizando semântica funcional. Dessa forma, esse trabalho tem como principal objetivo aprimorar a forma de descrição da funcionalidade de um serviço, de forma que essa descrição possibilite a composição correta, do ponto de vista funcional e do ponto de vista temporal. O objetivo do Composer-Science não é obter uma descrição funcional aprimorada e também não é a verificação da corretude da composição gerada do ponto de vista funcional e temporal. O Composer-Science tem o objetivo de obter serviços e compor estes serviços formando um *workflow* científico a partir de informações semânticas, o que facilita o trabalho do cientista, considerando que este não necessita lidar com questões computacionais complexas que atendam os parâmetros semânticos definidos pelo usuário.

O trabalho de [SÁNCHEZ *et al.*, 2009] visa a obtenção de um ambiente de execução de serviços Web semânticos baseado na tecnologia de agentes. Assim como o Composer-Science, esse trabalho faz uso da tecnologia de serviços Web semânticos para auxiliar a busca de serviços e apresenta independência de domínio de aplicação.

Porém, uma de suas tarefas fundamentais é a gestão e manejo dos serviços Web semânticos realizando descoberta, seleção, composição e invocação de serviços. Dessa forma, difere do Composer-Science, que tem o objetivo de permitir a descrição semântica de um modelo abstrato de *workflow* por um usuário e, a partir dele, descobrir, selecionar e compor serviços Web semânticos obtendo um modelo executável de *workflow* expresso em WS-BPEL. O trabalho de [SÁNCHEZ et al., 2009] não é focado na composição de serviços Web, não possuindo portanto nenhum mecanismo que auxilie o usuário na tarefa de modelagem da composição.

O trabalho de [KARAKOC e SENKUL, 2009] se baseia no uso de restrições para permitir que o usuário defina o serviço composto desejado. O *framework* proposto nesse trabalho é capaz de interpretar essas restrições e, utilizando recursos semânticos, encontrar e compor serviços, gerando um esquema ao final de sua execução que representa um serviço composto.

O Composer-Science tem o objetivo de gerar um modelo de *workflow* ao final de sua execução, e não um modelo de serviço composto, como o trabalho de [KARAKOC e SENKUL, 2009]. A geração de um modelo de *workflow* difere da geração de um serviço Web composto no sentido de que o modelo do *workflow* é expresso em uma linguagem

específica para essa finalidade, apresentando cada serviço que o compõe, podendo ser modificado e controlado, no momento de sua execução, pelo pesquisador. Porém, assim como em [KARAKOC e SENKUL, 2009], o Composer-Science faz uso de semântica para descobrir serviços e analisar sua compatibilidade com o modelo do *workflow* e com os outros serviços encontrados, que também serão utilizados na composição do *workflow*.

Considerando os SGWf (Sistemas de Gerenciamento de Workflows), o Composer-Science tem um objetivo diferente. A maioria dos SGWf existentes, tais como [TAVERNA, 2010], [VISTRAILS, 2010] e [KEPLER, 2008], por exemplo, visam principalmente a definição e execução dos *workflows*. O *framework* proposto tem como objetivo permitir a definição do *workflow* de maneira semântica e, a partir dessa definição, encontrar serviços para compor esse *workflow*. O Composer-Science não permite a execução do *workflow* obtido, como a maioria dos SGWf, mas auxilia na sua criação por meio do uso de recursos semânticos, o que permite um alto nível de abstração na definição do workflow, facilitando a tarefa do cientista.

4. COMPOSER-SCIENCE

No contexto de *e-Science*, as tecnologias de *workflow* fornecem ambiente para a resolução de problemas por cientistas, facilitando a criação e execução de experimentos a partir do uso de uma grande quantidade de dados e serviços disponíveis [PIGNOTTI *et al.,* 2008].

Este *framework* faz parte de um contexto amplo que consiste na especificação e desenvolvimento de uma infraestrutura para aplicações em *e-Science*, cujo objetivo é o suporte computacional a projetos de pesquisa e a pesquisadores que desejam compartilhar experiências e resultados. Esta infraestrutura visa apoiar o gerenciamento de aplicações e resultados de pesquisa relacionados a um dado domínio de aplicação. O *framework* proposto nessa dissertação usa os conceitos de repositório de serviços Web semânticos, ontologias, composição de serviços Web semânticos, e *workflows* científicos.

Este capítulo descreve os objetivos do *framework* Composer-Science e a arquitetura proposta para sua implementação. É apresentada a descrição arquitetural do Composer-Science e as ferramentas utilizadas.

4.1. OBJETIVOS

Os serviços Web semânticos apresentam tecnologias favoráveis à sua composição para a obtenção de processos mais complexos, tais como o uso de padrões Web, independência de plataforma, independência de linguagem de programação utilizada para o desenvolvimento, possibilidade de processamento distribuído, e, principalmente, o uso de recursos semânticos que possibilitam sua descoberta, composição e invocação automáticas. Com o objetivo de auxiliar na descoberta de serviços Web para a composição de *workflows* científicos, propomos o desenvolvimento de um *framework* que realize a busca de serviços Web semânticos e componha estes, definindo assim, um *workflow* científico. Especificamente, o *framework* deve permitir:

a. Registro e armazenamento, nos repositórios distribuídos (bancos de dados) do *framework,* de ontologias de domínio (OWL) e anotações dos serviços Web semânticos (OWL-S).
b. Realização de pesquisa semântica, baseada em requisitos fornecidos pelo pesquisador, nos repositórios distribuídos, a fim de realizar a descoberta de serviços Web semânticos que atendam os requisitos semânticos fornecidos.

c. Análise sintática, baseada em requisitos estruturais (dados de entrada e saída), além da análise semântica dos serviços descobertos por meio da pesquisa semântica, a fim de se obter possíveis composições dos mesmos.
d. Geração de modelos de *workflows* em WS-BPEL a partir das composições possíveis.

A geração de modelos em WS-BPEL permite sua composição com outros *workflows.* Conforme apresentado no capítulo 3, um *workflow* WS-BPEL é um serviço Web que pode ser usado dentro de outro *workflow* WS-BPEL e assim por diante. Essa estrutura permite a composição de um *workflow* a partir de vários outros [TAYLOR *et al*., 2006]. Além disso, o uso desta linguagem padrão permite que este *workflow* possa ser executado em Sistemas de Gerenciamento de *Workflow* capazes de executar processos definidos em WS-BPEL.

4.2. ONTOLOGIAS

Desde a introdução do uso de ontologias na Ciência da Computação progressos consideráveis vêm sendo feitos nessa área. O surgimento da Web Semântica trouxe consigo a crescente necessidade de reuso de conhecimento e, as ontologias e métodos de resolução de problemas (que podem ser considerados os precursores dos serviços Web semânticos) desempenham um papel muito importante nesse contexto.

O *Composer-Science* usa, além dos conceitos de repositório de serviços Web semânticos, composição de serviços Web semânticos, e *workflows* científicos, o conceito de ontologias, para descobrir serviços e compor *workflows* científicos a partir deles. As ontologias de domínio são utilizadas para, segundo o contexto científico em que o *workflow* se insere, i) descrever o domínio de aplicação dos *workflows* a serem compostos, ii) servir de base para a anotação de serviços Web semânticos, e também, iii) para a realização de inferências que auxiliem na busca de serviços nos repositórios.

Além das ontologias de domínio, que relacionam o *workflow* ao contexto científico em que ele está inserido, ao longo do desenvolvimento deste trabalho, surgiu a necessidade de descrever o domínio de *workflows* científicos*,* segundo o modelo que será gerado para representá-lo. Desta forma, considerando que, ao final do processo de composição do *workflow*, é gerado um modelo WS-BPEL que o representa, com o objetivo de tornar a descrição inicial do *workflow* mais precisa, foi desenvolvida uma ontologia inicial que descreve este modelo WS-BPEL. Esta ontologia, chamada WS-BPELOnto (*WS-BPEL Ontology*), foi desenvolvida para ser utilizada juntamente com as ontologias de domínio na

descrição do *workflow* científico cuja composição se deseja obter ao final da execução do *framework*.

As ontologias de domínio utilizadas no *framework* podem ser relacionadas a qualquer área científica relacionada ao contexto de *e-Science*. É permitido ao usuário do sistema acrescentar ontologias de domínio para auxiliá-lo a descrever seu *workflow*. Por outro lado, a ontologia WS-BPELOnto não pode ser modificada e é utilizada como base para a geração do modelo WS-BPEL que representa o *workflow* final.

4.2.1. Ontologia WS-BPELOnto

Neste trabalho é proposto o uso da ontologia WS-BPELOnto (*WS-BPEL Ontology*), aplicada à representação de *workflows* compostos por serviços Web por meio da linguagem WS-BPEL. Objetivos, características e limitações da ontologia são apresentados, bem como uma visão geral e uma descrição detalhada dos seus elementos.

A linguagem escolhida para definição da ontologia foi OWL por prover expressividade suficiente para representar os conceitos desejados, e ao mesmo tempo permitir o uso de mecanismos de inferência e a definição de regras semânticas. A escolha da linguagem se deu com o objetivo de usar um formato padronizado e recomendado pelo W3C, além disso, um número crescente de ferramentas para trabalhar com OWL, como Protégé-OWL [PROTÉGÉ, 2010] e Jena [JENA, 2010], e a implementação de mecanismos de inferência como Racer [RACER, 2010] e Pellet [PELLET, 2010] facilitam seu uso e permitem a criação de aplicações customizadas.

O principal objetivo da ontologia WS-BPELOnto é prover uma base para a conceitualização, a partir de um modelo WS-BPEL, das partes que compõem um *workflow*. A ontologia visa facilitar e tornar mais clara a descrição semântica do modelo do *workflow* desejado pelo usuário do sistema, e liga de forma direta a descrição inicial do *workflow* ao modelo WS-BPEL obtido ao final da execução do *framework*. Neste sentido, a WS-BPELOnto é apresentada em uma versão inicial. Axiomas, regras semânticas serão detalhados em versões futuras. Considerando os objetivos propostos inicialmente, que são relacionados ao suporte a composição de serviços web no Composer-Science, a versão inicial da ontologia atende aos seus propósitos.

A *Workflow Management Coalition* (WfMC) é uma organização que tem como objetivo ampliar a exploração da tecnologia de *workflow*, por meio do desenvolvimento de normas e de uma terminologia comum. A ontologia proposta neste trabalho busca atender as necessidades de descrição encontradas ao longo do desenvolvimento do *framework*

Composer-Science, e foi desenvolvida utilizando como base as definições técnicas de terminologias relacionadas a *workflows* propostas pela WfMC [WfMC, 1999] associando-as às terminologias relacionadas ao modelo de *workflow* WS-BPEL.

Os conceitos básicos e terminologias associadas a *workflows*, assim como as relações entre eles, propostos em [WfMC, 1999], são apresentados na Figura 7.

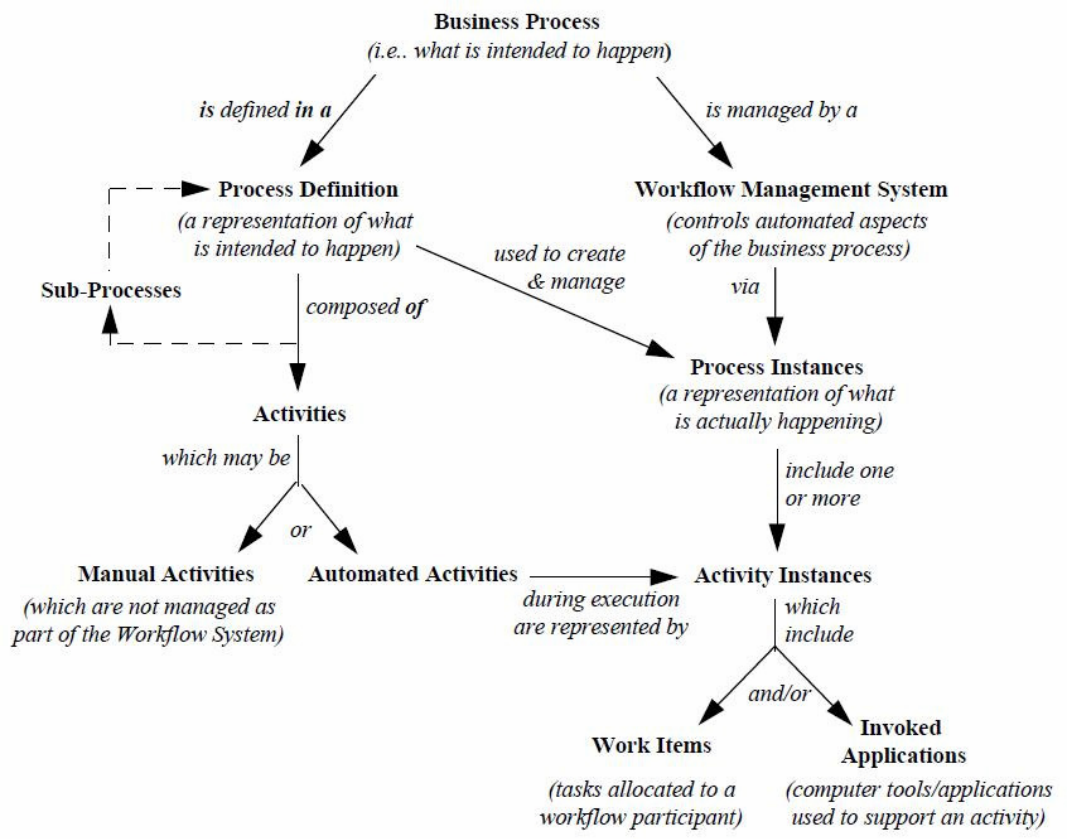


Figura 7 Terminologia básica e relações [WfMC, 1999].

Os termos básicos presentes na terminologia proposta em [WfMC, 1999] utilizados para o desenvolvimento da ontologia WS-BPELOnto foram adaptados para o contexto de *workflows* científicos, considerando que a proposta de [WfMC, 1999] visa principalmente a nomenclatura utilizada na definição de *workflows* de negócios.

Em [WfMC, 1999], além dos conceitos básicos, são apresentados conceitos similares e suas ligações com esses conceitos básicos. Dessa forma, utilizamos também os conceitos similares apresentados para desenvolvimento da ontologia WS-BPELOnto, de forma a ampliar seu poder de expressão.

A partir de um modelo WS-BPEL, foram extraídos os conceitos básicos relacionados a um *workflow* expresso nessa linguagem, e considerados indispensáveis para a descrição de um *workflow* no *framework* Composer-Science proposto neste trabalho.

Partindo dos conceitos básicos relacionados ao modelo WS-BPEL, foram inseridos os conceitos propostos em [WfMC, 1999], relacionando-os aos primeiros sempre que possível. Os conceitos básicos oriundos de WS-BPEL utilizados na ontologia WS-BPELOnto são *Process*, *PartnerLink* e *Variable*, apresentados na Figura 8.

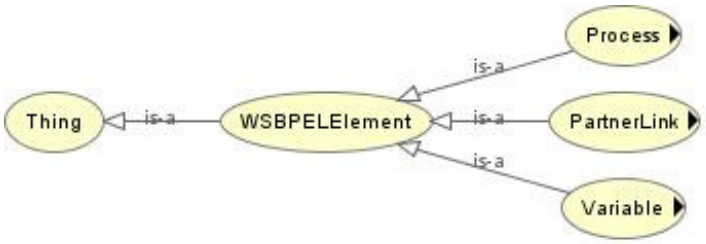


Figura 8 Nível topo da ontologia WS-BPELOnto.

A classe *Process* e suas subclasses representam um processo WS-BPEL. Um *workflow* é representado, em WS-BPEL, por um processo.

Um documento WS-BPEL contém um elemento XML <process> que identifica o processo representado. Esse elemento envolve a lógica de processamento e a definição do processo em si, além dos passos com chamadas a serviços Web que são interconectados para compor o processo como um todo. A Figura 9 apresenta a estrutura da classe *Process*.

Considerando que um *Process* em WS-BPEL representa um *workflow*, consideramos essa ligação semântica na estrutura da ontologia proposta. Sendo assim, segundo a ontologia WS-BPELOnto, um *Workflow* pode ser de negócios (*BusinessWorkflow* ou *BusinessProcess*) ou científico (*CientificWorkflow* ou *CientificProcess*). No contexto deste trabalho, se insere somente a definição de *workflow* científico.

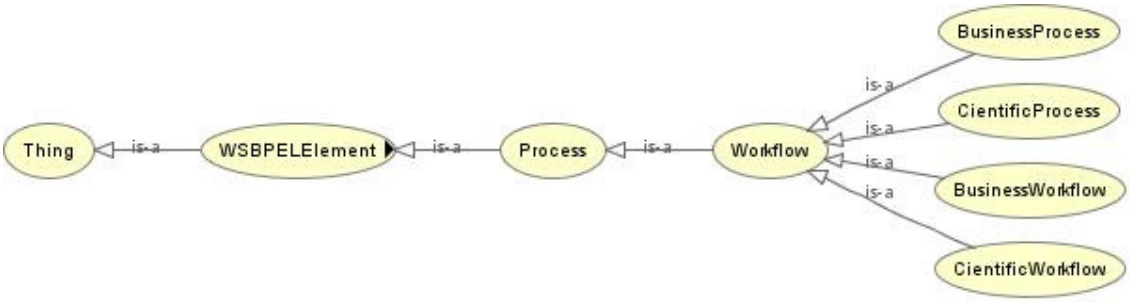


Figura 9 Classe *Process* da ontologia WS-BPELOnto.

A Classe *PartnerLink* representa um serviço. No caso de um modelo WS-BPEL de *workflow*, cada tarefa que compõem o *workflow* ou *Process* é representada por um *PartnerLink*. Em WS-BPEL cada *PartnerLink* representa um serviço Web e, dessa forma, o

workflow é composto por serviços Web. A Figura 10 apresenta a estrutura da classe *PartnerLink*.

Sabendo que cada *PartnerLink* corresponde a uma tarefa do *workflow*, podemos considerar que um *PartnerLink* é uma atividade (*Activity*) ou uma tarefa (*Task*) ou um passo (*Step*) do *workflow*. Com base na terminologia fornecida por [WfMC, 1999], uma atividade (*Activity*) pode ser automatizada (*AutomatedActivity*) ou manual (*ManualActivity*).

Uma atividade automatizada pode ser uma função (*Function*), uma aplicação que é invocada (*InvokedApplication*), um serviço (*Service*), ou mais especificamente, um serviço Web (*WebService*), que é o tipo de tarefa que compõe um *workflow* no contexto do Composer-Science.

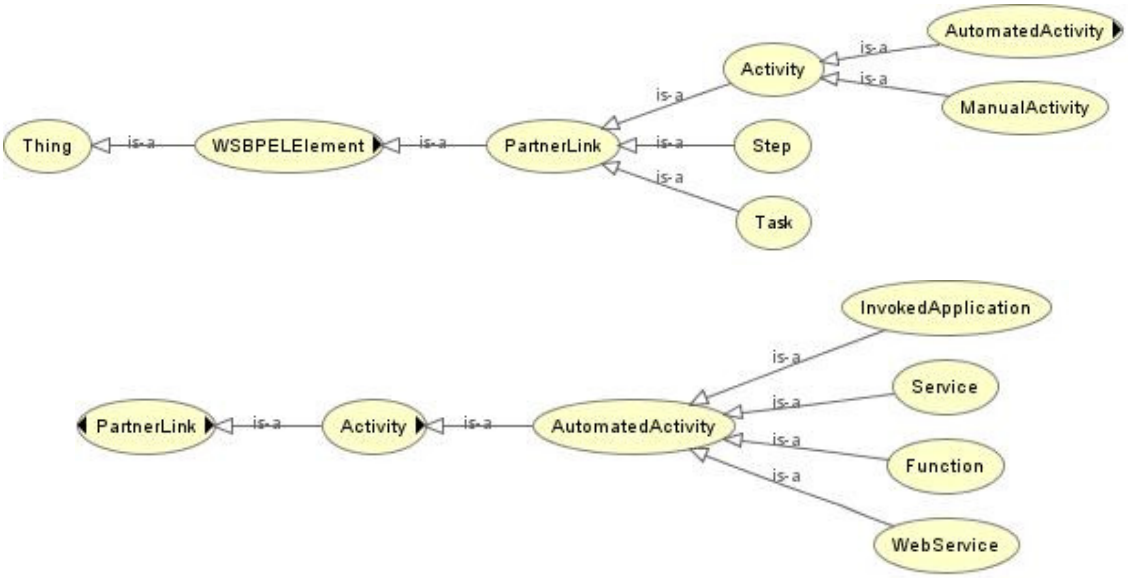


Figura 10 Classe *PartnerLink* e expansão da classe *Activity* da ontologia WS-BPELOnto.

A classe *Variable* Representa uma variável, sendo uma entrada ou saída de um serviço ou do *workflow*. A Figura 11 apresenta a estrutura da classe *Variable*.

Uma variável pode ser de entrada (*InputVariable*) ou pode ser uma saída/resultado (*OutputVariable*) de um serviço (*InputPartnerLinkVariable*). O escopo da variável também pode alcançar o *workflow* como um todo, sendo ela de entrada (*InputProcessVariable*) ou de saída/resultado (*OutputProcessVariable*) do *workflow*.

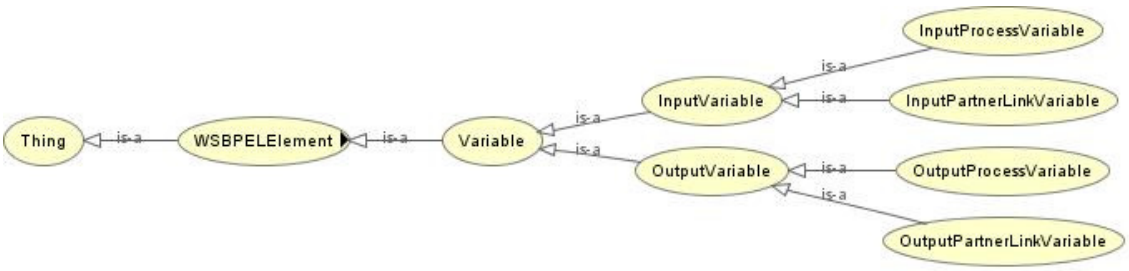


Figura 11 Classe *Variable* da ontologia WS-BPELOnto.

A ontologia WS-BPELOnto proposta é uma versão inicial e, como tal, apresenta algumas limitações:

- Não foi considerado nenhum tipo de integração ou alinhamento com outras ontologias de domínio. No entanto, a existência de elementos semanticamente ricos na ontologia WS-BPELOnto indica a possibilidade de integração com outras ontologias do domínio de *workflows*.
- A ontologia foi testada apenas com os modelos de *workflows* definidos no contexto do *framework* Composer-Science, tendo sido criada especificamente para uso no âmbito deste trabalho. No entanto, ela pode ser estendida de forma a possibilitar sua utilização em outro contexto.
- Como ocorre com qualquer ontologia expressa em OWL, não é praticável a edição da ontologia diretamente no arquivo texto. OWL possui diversas representações diferentes e os editores OWL estão preparados para trabalhar com essas diversas representações. A complexidade da sintaxe torna inviável lidar diretamente com os marcadores OWL. Uma vez que a ontologia WS-BPELOnto é expressa em OWL (bem como os modelos gerados a partir dela), pressupõe-se o uso de ferramentas específicas para se trabalhar com os modelos. Um editor OWL como, por exemplo, o Protege [PROTÉGÉ, 2010], pode ser utilizado para acessar os modelos. No entanto, esses editores possibilitam o acesso a diversas classes, indivíduos e propriedades que são elementos na ontologia e que não deveriam ser visíveis ao usuário que está construindo o modelo.

4.3. ARQUITETURA

Na Figura 12 é representada a arquitetura do Composer-Science, o que permite um melhor entendimento das etapas realizadas pelo mesmo, para alcançar seu objetivo de descobrir serviços Web semânticos e compor um *workflow* científico a partir destes.

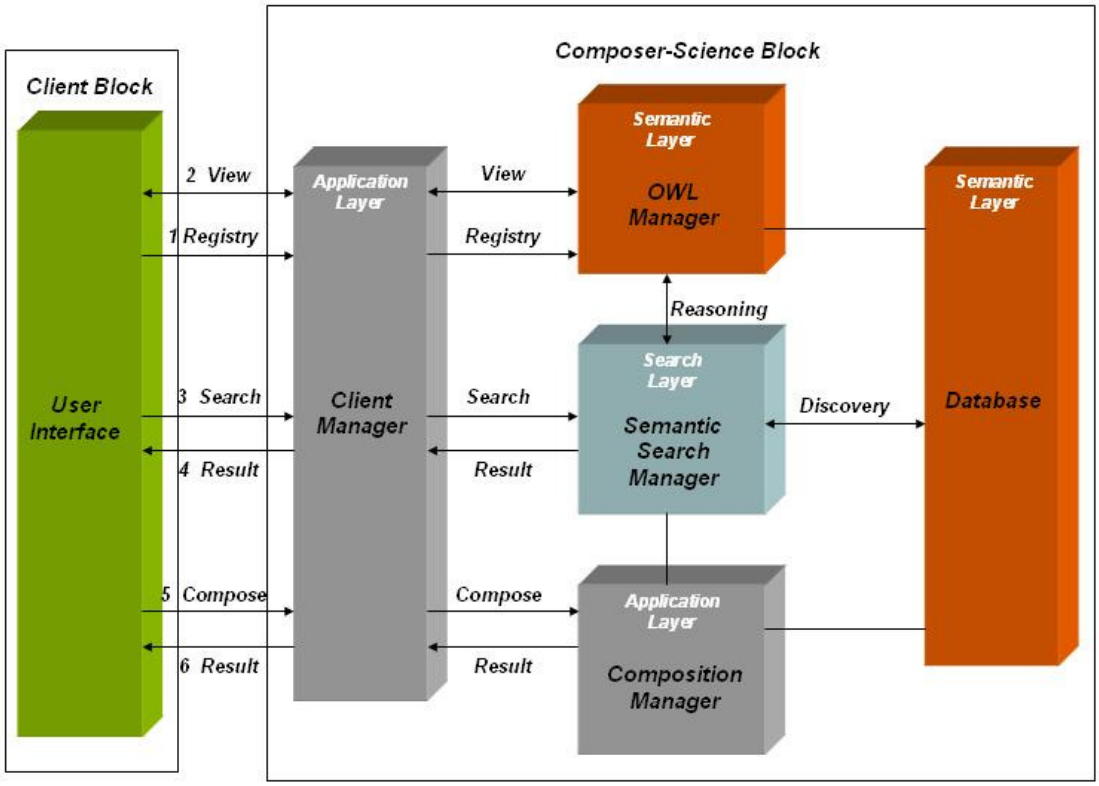


Figura 12 Arquitetura do *framework* proposto.

Podemos dividir o *framework* em dois blocos: o bloco do cliente (*Client Block* na Figura 12), que invoca o serviço, e o bloco que contém as camadas que compõem o *framework* (*Composer-Science Block* na Figura 12).

O bloco do cliente implementa uma interface (*User Interface* na Figura 12) por meio da qual o usuário interage com o *framework*, detalhando as especificidades relacionadas ao *workflow* que ele pretende criar. Esta interface consiste em uma aplicação Web que permite o detalhamento das características semânticas do *workflow* científico que está sendo definido pelo usuário do sistema.

Por meio da interface, o usuário tem acesso a representações gráficas (imagens) simplificadas das ontologias de domínio disponibilizadas pelo *framework* e pode definir quais das ontologias disponíveis serão utilizadas na descrição semântica do *workflow* e de suas

tarefas. Após definir quais ontologias de domínio serão utilizadas para descrever seu *workflow*, é possível que o usuário defina semanticamente, por meio do uso de termos presentes nessas ontologias, as funcionalidades a serem realizadas por cada tarefa que compõe o *workflow* e também o objetivo a ser alcançado pelo *workflow* como um todo. O outro bloco, denominado *Composer-Science Block*, realiza a busca dos serviços Web semânticos que atendam às especificações fornecidas pelo usuário, e fazem a composição destes serviços a fim de obter o *workflow* desejado ao final do processo.

O *framework* oferece quatro serviços para seus usuários:

- **View**: visualização do modelo da ontologia escolhida pelo usuário através da interface. O usuário seleciona uma das ontologias disponíveis no *framework* e solicita a sua visualização para melhor compreender sua estrutura.
- **Registry**: registro de ontologia de domínio (OWL) e descrição semântica (OWL-S) de um serviço Web. O usuário pode acrescentar novas ontologias e novos serviços Web semânticos ao sistema. As ontologias têm algumas de suas estruturas referenciadas em bancos de dados relacionais, e sua representação OWL armazenada nos repositórios do *framework*. As descrições semânticas OWL-S são armazenadas diretamente nos repositórios do *framework*.
- **Search**: busca de serviços que atendam às especificações de cada tarefa definida semanticamente pelo usuário. Com os dados semânticos fornecidos pelo usuário são realizadas inferências que ampliam o poder da pesquisa semântica, facilitando a descoberta de serviços compatíveis. A busca semântica é realizada com base na comparação entre as descrições semânticas (OWL-S) dos serviços disponíveis nos repositórios e, as descrições das tarefas do *workflow* definidas pelo usuário, incluindo os resultados obtidos a partir das inferências realizadas sobre estas descrições.
- **Compose**: com base na definição do *workflow* fornecida pelo usuário, é definido um modelo abstrato do *workflow.* A partir do modelo abstrato, utilizando os serviços descobertos nos repositórios, a partir da busca semântica, são gerados modelos executáveis do *workflow* considerados semanticamente possíveis. Os serviços dos modelos semanticamente possíveis têm suas estruturas sintáticas (incluindo os parâmetros de entrada e saída) analisadas e, então, são obtidos os modelos que são semântica e estruturalmente possíveis. Os modelos semântica e estruturalmente possíveis são apresentados ao usuário, que seleciona um deles para que seja gerado o documento WS-BPEL correspondente, que é então apresentado como resultado final do processo de execução do *framework*.

O serviço *View* disponibilizado pelo *framework* é importante para facilitar a sua utilização pelo usuário que irá definir o modelo de *workflow*. Este usuário pode não conhecer as ontologias de domínio disponibilizadas pelo Composer-Science e, portanto, é necessário que ele tenha acesso a representações das mesmas para que possa utilizá-las como base para descrever seu *workflow*.

O serviço *Registry* auxilia na utilização de novos domínios de aplicação, facilitando ainda a reutilização de um número maior de serviços existentes. A possibilidade de inserção de novos serviços nos repositórios do *framework* permite que mais serviços sejam descobertos e utilizados nas composições.

Desta forma, apesar de importantes para o funcionamento do *framework*, estes serviços não influenciam diretamente na sua principal funcionalidade, que consiste em descobrir serviços semanticamente compatíveis com a especificação fornecida pelo usuário e compor estes serviços a fim de obter um *workflow* científico. Os serviços *Search* e *Compose* estão mais diretamente ligados ao objetivo principal do *framework* e, por isso, serão detalhados nas seções 4.4 e 4.5, respectivamente.

Considerando os serviços *Search* e *Compose*, o *Composer-Science Block* é composto por três camadas: *Semantic Layer*, *Search Layer*, e *Application Layer*.

4.3.1. Semantic Layer

A *Semantic Layer* é a camada onde são gerenciados os acessos às ontologias (OWL) e descrições semânticas de serviços, também chamadas anotações semânticas de serviços (OWL-S).

Nesta camada é implementado o *OWL Manager*, que é o módulo gerencial responsável pelo armazenamento das anotações semânticas em repositórios e pelo mecanismo que permite a realização de inferências sobre as ontologias de domínio. Utilizamos o *framework* SOR [IBM, 2010] para auxiliar na realização de inferências semânticas sobre as ontologias de domínio. Por meio do uso do SOR, é possível realizar inferências diretamente a partir do arquivo OWL de uma ontologia. As rotinas de inferências semânticas sobre ontologias e descrições semânticas são implementadas por esta camada. No capítulo 5 será detalhada sua implementação.

4.3.2. Search Layer

A *Search Layer* é composta pelo módulo gerencial *Semantic Search Manager* que é responsável pela busca semântica e descoberta de serviços compatíveis com as descrições fornecidas pelo usuário e com os resultados das inferências obtidos a partir das mesmas.

Esta camada solicita a realização de inferências (*reasoning*) nas ontologias de domínio à *Semantic Layer*, enviando como parâmetros as especificações semânticas definidas, com base nas ontologias disponíveis, pelo usuário para cada tarefa do *workflow*. Desta forma, são obtidos, como resultados desta inferência, termos relacionados à especificação inicial, ampliando assim, as possibilidades de busca de serviços.

As informações fornecidas pelo usuário e as obtidas por meio de inferências são utilizadas para a realização de pesquisa semântica (*discovery*) em repositórios distribuídos para encontrar serviços Web compatíveis semanticamente com cada tarefa do *workflow.* Para encontrar os serviços, as descrições semânticas (OWL-S) dos serviços disponíveis nos repositórios são analisadas e comparadas com os dados semânticos relacionados a cada tarefa.

Informações semânticas são extraídas da descrição semântica (OWL-S) de cada serviço que estão armazenadas nos repositórios, por meio do uso da OWL-S API [SIRIN, 2004], que fornece uma API[2] (*Application Programming Interface*) Java que possibilita a leitura, execução e escrita de OWL-S. As informações obtidas a partir do OWL-S de um serviço são comparadas com as informações fornecidas pelo usuário para uma determinada tarefa para definir se o serviço descrito pelo OWL-S pode ser utilizado para realizar a tarefa.

Como resultados da busca realizada nos repositórios podem ser encontrados vários serviços que atendam às especificações de uma dada tarefa. A *Application Layer* é responsável por selecionar o serviço mais adequado a dada tarefa. Na seção 4.4 é detalhado como é feita a busca semântica realizada pela *Search Layer*.

4.3.3. Application Layer

Esta camada é composta pelos módulos gerenciais *Composition Manager* e *Client Manager*. O *Composition Manager* é responsável pela geração de composições formadas com os

[2] Uma API define e descreve uma interface para a interação com um conjunto de funções prontas. O usuário da API não necessita conhecer a implementação das funções, apenas deve saber utilizá-las por meio da interface disponível.

serviços descobertos, que representem o *workflow* especificado pelo usuário, assim como pela geração do modelo WS-BPEL que é apresentado como resultado do processo de composição ao usuário final. O *Client Manager* é responsável por toda interação com os usuários do *framework*, tendo como principal finalidade impedir que os pesquisadores tenham acesso à estrutura interna do *framework*, fornecendo um ponto único de entrada/saída no sistema.

Para compreender o funcionamento da *Application Layer* temos que considerar que, após a obtenção de serviços que realizem cada tarefa definida inicialmente pelo usuário, como parte do *workflow* pela *Search Layer*, é iniciado o processo de composição dos serviços a fim de se obter possíveis composições capazes de representar o *workflow* definido pelo usuário. A composição dos serviços é realizada utilizando o paradigma de Redes de Petri [HAMADI e BENATALLAH, 2003], que auxilia na análise da compatibilidade estrutural e semântica de cada serviço para definir sua composição com outros. A composição dos serviços, assim como a forma de inserção do paradigma de Redes de Petri neste contexto serão detalhadas nas seções 4.4 e 4.5.

A definição do *workflow* fornecida pelo usuário é armazenada em uma base de dados. A partir dos dados armazenados nesta base, uma Rede de Petri é definida internamente pelo sistema para representar o *workflow* de maneira abstrata. Esta rede não apresenta a definição dos serviços que irão executar cada tarefa, contendo apenas as tarefas, sua estrutura de execução e os termos ontológicos relacionados a cada tarefa, entradas e saídas. No mínimo, duas ontologias são utilizadas para anotar semanticamente as tarefas abstratas do *workflow*, ou seja, uma ontologia relacionada ao domínio de aplicação e uma ontologia relacionada à definição de *workflows* científicos em WS-BPEL, no caso, a ontologia WS-BPELOnto.

Com base na rede abstrata do *workflow*, os serviços que atendam às especificações semânticas de cada tarefa são descobertos nos repositórios e, então, são definidas redes executáveis para cada possível composição dos serviços descobertos. As redes que representam o *workflow* executável são definidas respeitando a estrutura abstrata. Em uma primeira etapa, estas redes executáveis são formadas com base apenas na compatibilidade semântica dos serviços em relação às anotações semânticas das tarefas do *workflow* sendo, por este motivo, chamadas, no contexto deste trabalho, de redes executáveis semanticamente possíveis. Em uma segunda etapa, cada rede semanticamente possível tem a compatibilidade estrutural (dados de entrada e saída) de cada um de seus serviços definidos para cada tarefa do *workflow* analisada e, então, é definido se cada rede é semântica e também estruturalmente possível.

As redes executáveis semântica e estruturalmente possíveis são exibidas para o usuário, que escolhe aquela que deseja obter como resultado do processo de composição

do seu *workflow*. Após a escolha da composição, o usuário solicita a geração do modelo WS-BPEL que a represente. Este modelo WS-BPEL é então disponibilizado como saída do sistema para o usuário final. O processo de composição, assim como a geração do modelo WS-BPEL serão detalhados na seção 4.5.

4.4. BUSCA SEMÂNTICA DE SERVIÇOS

Com a difusão dos serviços Web capazes de auxiliar nas pesquisas em diversas áreas científicas, há um crescente interesse em utilizar estes serviços para compor *workflows* que representem procedimentos experimentais. Entretanto, como o número de serviços disponíveis na Web vem crescendo, e os *workflows* científicos são bastante complexos, envolvendo muitas tarefas, se torna difícil e custoso, em termos de tempo, buscar manualmente os serviços capazes de realizar estas tarefas, uma a uma. A fim de resolver este problema, diversos estudos vêm sendo realizados no sentido de automatizar a busca por serviços que realizam determinada tarefa. Uma forma de otimizar este trabalho é o uso de buscas semânticas.

A busca semântica de serviços Web se baseia no uso de ontologias de domínio e serviços Web semânticos. Uma ontologia de domínio define a terminologia utilizada em um determinado domínio, enquanto que os serviços Web semânticos são serviços Web anotados semanticamente. Esta anotação semântica é baseada em uma ontologia de domínio, contendo as características semânticas do serviço.

Neste trabalho, é realizada a busca semântica por serviços Web que atendam às especificações semânticas de um *workflow* científico fornecidas por um usuário. O usuário descreve, a partir das ontologias disponibilizadas pelo Composer-Science, cada tarefa do *workflow* científico que pretende compor e, a partir destas informações fornecidas por ele, é realizada a busca de serviços capazes de executar cada tarefa descrita.

Utilizando as descrições semânticas das tarefas do *workflow* selecionadas pelo usuário, são realizadas inferências nas ontologias de domínio definidas para o *workflow*. Desta forma, são obtidos termos diferentes dos utilizados pelo usuário na descrição da tarefa, porém, de significado semelhante a eles, segundo as ontologias de domínio. Estas inferências ampliam o poder de busca de serviços, uma vez que são descobertos serviços compatíveis com um número maior de termos.

Para a realização de inferências nas ontologias de domínio, utilizamos o framework SOR. Na Figura 13, temos um trecho de código Java que apresenta um exemplo simplificado de utilização do SOR para a obtenção de termos relacionados a um termo

especificado. O método apresentado no código recebe como parâmetros um *String* contendo a localização do arquivo da ontologia de domínio sobre a qual será realizada a inferência (*ontologyFile*) e um *String* contendo o termo a partir do qual serão realizadas inferências para se obter termos relacionados a ele (*mainClass*). Como retorno deste método é obtida uma lista de *Strings* que contém em cada uma de suas posições um termo relacionado ao termo utilizado como referência para a inferência na ontologia. Se nenhum termo relacionado for encontrado, a lista retornada apresenta apenas uma posição contendo o *String* que indica que não existem resultados (*No descendant classes*). Os termos relacionados ao termo inicial fornecido, no método apresentado no exemplo, são classes descendentes da classe (termo) fornecida como parâmetro segundo a ontologia de domínio utilizada para a realização das inferências.

A partir dos termos utilizados pelo usuário para descrever a tarefa e também dos termos adicionais obtidos a partir das inferências nas ontologias de domínio, os repositórios são inspecionados, buscando cada um dos serviços disponíveis (através dos arquivos OWL-S armazenados que descrevem semanticamente os serviços), e comparando sua descrição semântica com os termos que descrevem as tarefas. Os termos utilizados na descrição dos serviços são extraídos de sua anotação semântica (OWL-S) por meio do uso da OWL-S API [SIRIN, 2004]. A OWL-S API é uma API Java que permite o acesso para leitura, execução e escrita de anotações semânticas OWL-S. Esta API suporta a leitura de diferentes versões de OWL-S, sendo elas: OWL-S 1.0, OWL-S 0.9 e DAML-S 0.7. A OWL-S API faz parte de um projeto ainda em desenvolvimento, apresentando código aberto, o que permite que suas funcionalidades sejam estendidas de acordo com as necessidades encontradas durante sua utilização [SIRIN, 2004].

```
public List<String> descendantClasses(String ontologyFile,
                                       String mainClass){
    String classe = mainClass;
    OWLOntology ontology = null;
    OWLParser parser = new OWLParserImpl();
    String file = ontologyFile;
    OWLDocument owlDoc = new OWLDocumentImpl(file, null, true);
    parser.addOWLDocument(owlDoc);
    ontology = parser.parseOWLDocument(owlDoc);
    OWLTaxonomyReasoner reasoner =
           StructuralReasonerFactory.instance().createOWLTaxonmyReasoner();
    reasoner.initialize(ontology);
    OWLClass c1 = (OWLClass) ontology.getContainedResource(null, classe);
    List<String> resp = new ArrayList<String>();
    try{
        List l = reasoner.getDescendantClasses(c1);
        for (Iterator iter = l.iterator(); iter.hasNext();) {
            OWLClass clazz = (OWLClass) iter.next();
            resp.add(clazz.getURI().toString());
        }
    } catch(Exception e){
        resp.add("No descendant classes.");
    }
    return resp;
}
```

Figura 13 Código para inferência usando SOR em Java.

Na Figura 14 temos um trecho de código Java que exemplifica o uso da OWL-S API para a extração de informações relevantes a partir da anotação semântica (OWL-S) de um serviço. O método apresentado no código recebe como parâmetro um *String* que contém a localização do arquivo OWL-S que descreve um dado serviço (*owlsFile*). A partir desta descrição semântica, são impressas informações como o nome do serviço (*Service Name*), os seus parâmetros de descrição (*Service Parameter*), os nomes de seus parâmetros de entrada e saída (*Input* e *Output*), e o texto descritivo (*Text Description*) apresentado na anotação semântica do serviço.

Cada termo que descreve uma dada tarefa é comparado com cada parâmetro de descrição de um dado serviço para que seja verificado se ambos são compatíveis. Por meio do uso da OWL-S API, como apresentado no código da Figura 14, são extraídas informações da descrição semântica de um dado serviço disponível nos repositórios do *framework*. A Figura 15 demonstra de forma simplificada a extração das informações da anotação semântica do serviço. Estas informações extraídas são comparadas uma a uma com os termos utilizados pelo usuário para descrever uma dada tarefa do *workflow* especificado por ele, de forma que se possa definir quais serviços atendem à descrição da tarefa em questão. Este processo é repetido para cada tarefa, comparando com cada serviço encontrado nos repositórios.

```
public void owlsData(String owlsFile) throws FileNotFoundException{
    OWLKnowledgeBase kb = OWLFactory.createKB();
    kb.setReasoner("Pellet");
    org.mindswap.owls.service.Service s =
            (Service) kb.readService(URI.create(owlsFile));
    //Service
    System.out.println("Service: "+s.getLocalName());
    //Profile
    Profile p = s.getProfile();
    System.out.println("Profile: "+p.getLocalName());
    //Service Parameters
    OWLIndividualList oil = p.getServiceParameters();
    for (Iterator it = oil.iterator(); it.hasNext();) {
        OWLIndividual oi = (OWLIndividual) it.next();
        System.out.println("Service parameter: "+oi.getLabel());
    }
    //Inputs
    InputList il = p.getInputs();
    for (Iterator it = il.iterator(); it.hasNext();) {
        Input i = (Input)it.next();
        System.out.println("Input: "+i.getLocalName());
    }
    //Outputs
    OutputList ol = p.getOutputs();
    for (Iterator it = ol.iterator(); it.hasNext();) {
        Output o = (Output)it.next();
        System.out.println("Output: "+o.getLocalName());
    }
    //ServiceName
    System.out.println("Service Name: "+p.getServiceName());
    //TextDescription
    System.out.println("Text Description: "+p.getTextDescription());
    //Process
    Process ps = s.getProcess();
    System.out.println("Process: "+ps.getLocalName());
    //Grounding
    Grounding g = s.getGrounding();
    System.out.println("Grounding: "+g.getLocalName());
    AtomicGroundingList agl = g.getAtomicGroundings();
    for (Iterator it = agl.iterator(); it.hasNext();) {
        AtomicGrounding ag = (AtomicGrounding) it.next();
        System.out.println("Atomic Grounding: "+ag.getLocalName());
    }
}
```

Figura 14 Código Java para extração de dados de OWL-S usando OWL-S API.

Comparando o código Java apresentado na Figura 14 com os dados obtidos a partir da anotação semântica na Figura 15, temos que os métodos do objeto *s* (*Service*) *getProfile*, *getProcess* e *getGrounding* recuperam, respectivamente, os valores dos atributos *presents*, *describedBy* e *supports*, de *Service* do documento OWL-S representado na Figura 15. Temos ainda que os métodos do objeto *p* (*Profile*) *getServiceParameters*, *getInputs*, *getOutputs*, *getServiceName* e *getTextDescription* recuperam, respectivamente, os valores dos atributos *serviceParameter*, *hasInput*, *hasOutput*, *presentedBy* e *textDescription*, de *Profile* do documento OWL-S. Com esses valores obtidos do documento OWL-S que descreve o serviço, é possível fazer algumas comparações com os dados fornecidos pelo usuário do sistema para verificar se o serviço atende uma determinada tarefa do *workflow* que está sendo composto.

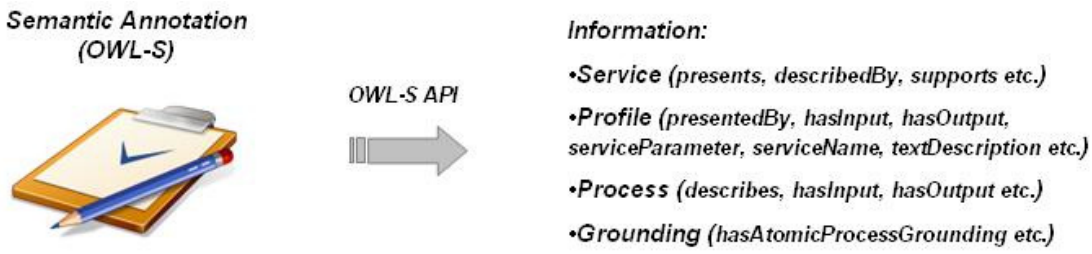


Figura 15 Esquema simplificado de extração de informações de OWL-S.

Para encontrar os serviços disponíveis no repositório que atendem à descrição semântica de uma determinada tarefa fornecida pelo usuário, as descrições do usuário são comparadas com cada informação obtida do OWL-S de cada serviço do repositório. Esse método de busca deve ser analisado em versões futuras, considerando questões de desempenho. Esta questão de desempenho não foi analisada por não ser o principal foco deste trabalho, que busca apenas uma solução inicial para o problema da busca semântica de serviços. Sendo assim, é possível, em trabalhos futuros, aprimorar o método de busca dos serviços de forma que se obtenha resultados com melhor desempenho na busca semântica.

4.5. COMPOSIÇÃO DE SERVIÇOS

A maioria dos estudos sobre a composição automática de serviços Web cria composições considerando apenas os dados de entrada e saída (*input* e *output*) dos serviços disponíveis, sem considerar sua semântica [SHIN, LEE e SUDA, 2009]. Desta forma, resultados que não satisfazem as intenções do usuário podem ser gerados. Além disso, a complexidade do processamento se torna muito alta, pois todas as combinações possíveis de serviços disponíveis devem ser consideradas. O *framework* proposto neste trabalho tem como objetivo considerar estes problemas e permitir que *workflows* científicos sejam criados por meio da composição de serviços Web semânticos.

Diversas abordagens têm sido propostas para tratar os mais variados aspectos da composição de serviços Web, tais como: linguagens de composição [WS-BPEL, 2007; WSFL, 2007], formalização da composição [RIBEIRO, 2008; SALIMIFARD e WRIGHT, 2001], ambientes de suporte à composição [TAVERNA, 2010; ORACLE, 2009]. Entre as linguagens para composição de serviços Web, WS-BPEL [WS-BPEL, 2007], descrita no capítulo 3, tem se destacado pelo fato de ser uma linguagem baseada em XML e

especificamente projetada para suportar a composição de serviços. Em termos de formalização, o objetivo básico é o uso de técnicas de descrição formal, como a Rede de Petri [HAMADI e BENATALLAH, 2003], também descrita no capítulo 3, para a verificação de propriedades da composição. Por fim, os ambientes mencionados consistem de sistemas especificamente projetados para suportar a composição de serviços Web. O *framework* proposto neste trabalho visa constituir-se em um ambiente de suporte à composição de serviços Web para a obtenção de *workflows* científicos.

Segundo [BENATALLAH *et al* 2002], os serviços podem ser compostos de maneira estática ou dinâmica. Esta escolha depende do tipo do processo que está sendo composto. Se os serviços que serão compostos têm uma natureza fixa, ou mudam raramente as suas interfaces e semântica, a composição estática satisfaz as necessidades. Contudo, um processo pode conter um conjunto de funções fracamente definidas a ser realizado, ou então ele tem que ter a possibilidade de se adaptar dinamicamente a mudanças não previstas no ambiente. Para estes casos, a composição estática pode não ser a abordagem mais adequada, pois alterações em um sistema baseado em composição estática requerem a interrupção da continuidade do processo. Composição dinâmica é uma alternativa importante neste caso, dado que existem processos críticos que não podem sofrer interrupções. Além da natureza estática e dinâmica da composição, os serviços *Web* podem ser compostos de diversas formas [HAMADI e BENATALLAH, 2003]:

- **Composição em seqüência**: executa, de forma serial, um serviço (S1) seguido de outro (S2). A principal característica deste tipo de composição é que, para se iniciar o serviço S2, S1 deverá terminar a sua execução;
- **Composição em paralelo**: executa, simultaneamente e independentemente, dois serviços arbitrários (S1 e S2). Como variação, este tipo de composição pode se dar de forma com ou sem comunicação entre os serviços envolvidos;
- **Composição com escolha**: serviços compostos desta forma resultam na execução de apenas um dos mesmos, descartando a execução do restante. A escolha é feita de maneira aleatória;
- **Composição seletiva**: similar a composição com escolha, diferindo na forma da escolha. Aqui, a mesma é realizada baseada em critérios e informações, e não de maneira aleatória;
- **Composição iterativa**: neste caso, um mesmo serviço S1 é invocado diversas e consecutivas vezes.

Conforme descrito no capítulo 3, Redes de Petri constituem um formalismo matemático que permite representação gráfica, e possui métodos para análise formal de um sistema. Os lugares e transições das Redes de Petri são usados para modelar uma visão lógica de pontos dos sistemas. Nesta seção será abordado como Redes de Petri podem ser

utilizadas tendo em vista a composição de serviços Web. A metodologia consiste de três passos: o primeiro consiste na construção da topologia da Rede de Petri condizente com a especificação semântica do *workflow* definida pelo usuário; o segundo passo consiste na definição das composições semanticamente possíveis realizadas com os serviços disponíveis; e o terceiro passo visa a análise estrutural dos serviços e a apresentação das composições semântica e estruturalmente possíveis, que serão apresentadas ao usuário.

Passo 1: Definição da topologia da Rede de Petri

A partir da definição do *workflow* fornecida pelo usuário do sistema, uma rede é definida para representar o *workflow* abstrato correspondente àquela definição inicial. A definição do *workflow* é armazenada em uma base de dados relacional e, com base nestes dados armazenados, o *framework* é capaz de definir a rede abstrata que representa o *workflow*.

A Figura 16 apresenta os modelos lógico e conceitual da base de dados utilizada para o armazenamento da definição do *workflow* fornecida pelo usuário do sistema. O modelo criado pelo usuário é armazenado com um nome (*name*), definido por ele, na tabela *workflow*.

As tarefas que compõem o *workflow* são armazenadas na tabela *task*, que contém informações como a ordem de execução das tarefas (armazenada no campo *sequence*), o *workflow* ao qual a tarefa pertence (*workflow_id*), o nome dado à tarefa (*name*) e a relação (as relações são armazenadas na tabela *relation*) da tarefa com a próxima tarefa pertencente ao mesmo *workflow* (*relation_id*).

A ontologia de *workflow* WS-BPELOnto e as ontologias de domínio disponíveis no *framework* estão armazenadas na tabela *ontology*. A tabela *ontology* apresenta um campo *ontologytype* que funciona como uma *flag*, sendo preenchido com valor “D”, quando o registro do banco de dados trata de uma ontologia de domínio, e sendo preenchido com o valor “W”, para identificar o registro da ontologia de *workflow* WS-BPELOnto.

As definições de quais ontologias de domínio serão utilizadas para descrever o *workflow* e suas tarefas (*tasks*) são armazenadas na tabela *workflow_ontology*, que relaciona um dado *workflow* com uma ou mais ontologias de domínio.

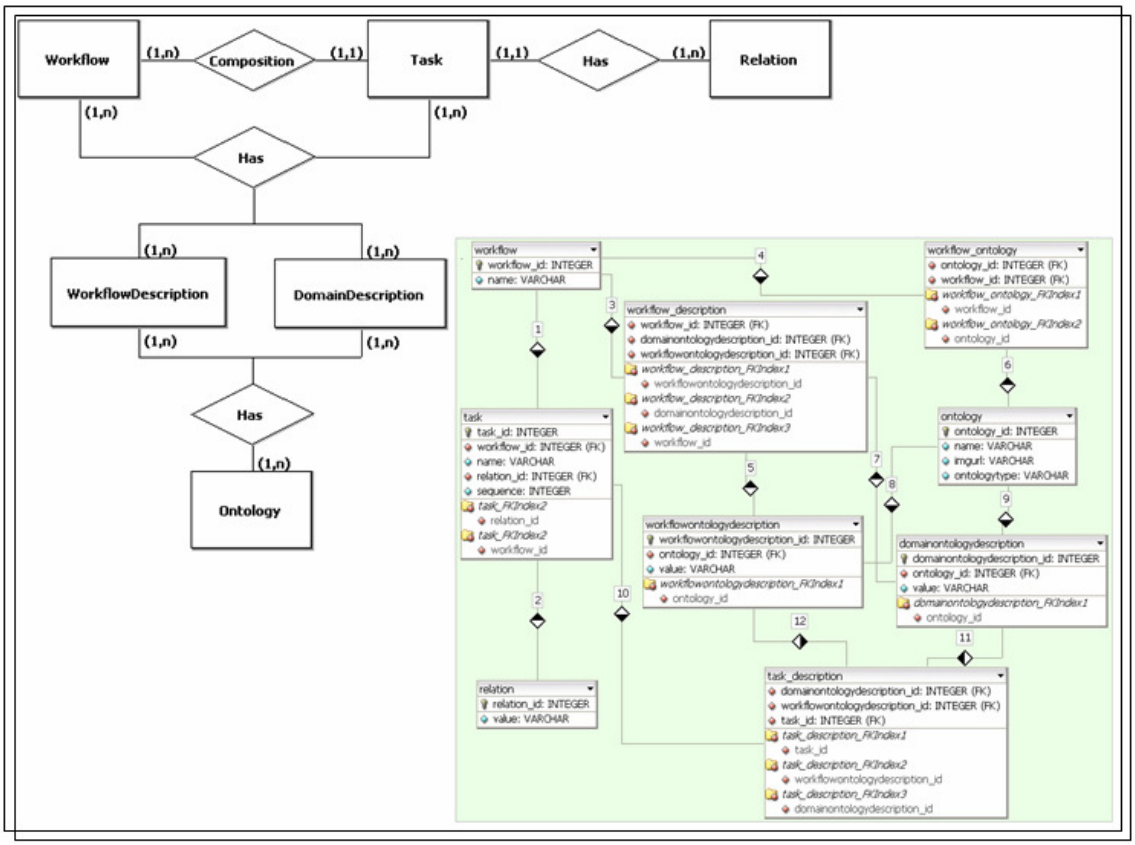


Figura 16 Estrutura para armazenamento de modelo de *workflow* abstrato.

As descrições semânticas (termos) das ontologias de domínio disponibilizadas pelo *framework* são armazenadas na tabela *domainontologydescription* no momento em que a dada ontologia é registrada no *framework*, o que permite que uma ou mais descrições possam ser associadas a cada tarefa do *workflow* por meio do campo *domainontologydescription_id* da tabela *task_description*.

As descrições semânticas (termos) da ontologia WS-BPELOnto se encontram armazenados na tabela *workflowontologydescription* e se relacionam com as tarefas do *workflow* por meio do campo *workflowontologydescription* da tabela *task_description.*

Os termos das ontologias de domínio e da ontologia WS-BPELOnto se relacionam com o *workflow*, respectivamente por meio dos campos *domainontologydescription_id* e *workflowontologydescription_id* da tabela *workflow_description*.

Considerando o modelo do banco de dados da Figura 16, temos um exemplo de modelo de *workflow* armazenado na base de dados, representado na Figura 17. A partir do modelo representado na Figura 17 é gerada a rede representada na Figura 18.

workflow	
workflow_id	*name*
1	Model

task				
task_id	*workflow_id*	*name*	*relation_id*	*sequence*
1	1	A	2	1
2	1	B	3	2
3	1	C	3	2
4	1	D	6	3

relation	
relation_id	*value*
1	SEQUENCIAL
2	OR-SPLIT
3	OR-JOIN
4	AND-SPLIT
5	AND-JOIN
6	FINAL

Figura 17 Modelo de *workflow* armazenado na base de dados.

Na Figura 18 temos o exemplo de uma rede que representa um *workflow* abstrato definido por um usuário. Este *workflow* executa uma tarefa A, em seguida, executa uma tarefa B ou uma tarefa C e, finalmente, executa a tarefa D, alcançando assim, seu resultado final.

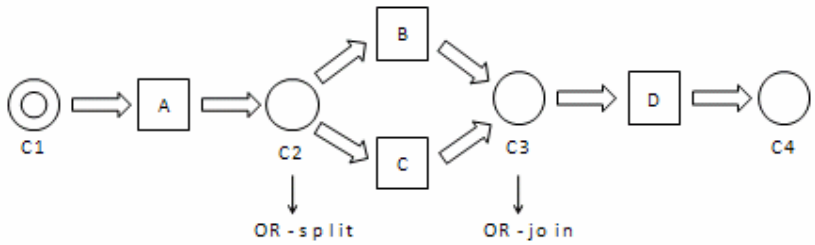


Figura 18 Rede de Petri que representa um *workflow* abstrato.

Passo 2: Composições semanticamente possíveis com os serviços encontrados

É realizada a busca semântica de serviços nos repositórios do *framework*, baseada nas descrições semânticas das tarefas do *workflow* definidas pelo usuário do sistema.

Uma das ontologias de domínio especificadas para serem utilizadas na descrição do *workflow* é a CelO [MATOS *et al*, 2009]. Então, a partir dos termos que descrevem cada tarefa, são realizadas inferências na ontologia para obter mais termos relacionados e, desta forma, ampliar o poder da busca semântica por serviços nos repositórios do *framework*. Utilizamos alguns exemplos de termos extraídos da CelO para exemplificar alguns trechos do passo 2 da composição.

Na Figura 19, a partir do modelo abstrato do *workflow* (*Abstract Workflow Model*), são selecionadas na base de dados as descrições (*descriptions*) relacionadas a cada tarefa que compõem o *workflow*. Conforme o exemplo apresentado, a tarefa A apresenta o termo *CellElement* como descrição relacionada ao domínio (termo da ontologia CelO) e *PartnerLink* como descrição relacionada a WS-BPEL (ontologia WS-BPELOnto), a tarefa B apresenta os termos *CellProcess* e *PartnerLink*, a tarefa C apresenta os termos *CellStructure* e *PartnerLink*, e a tarefa D apresenta os termos *CellMLModel* e *PartnerLink*.

As informações relevantes (*Information*) contidas nas descrições semânticas (OWL-S) dos serviços dos repositórios são extraídas de suas descrições por meio do uso da OWL-S API e, então, comparando os termos que descrevem as tarefas e os termos obtidos a partir das inferências com as informações extraídas de cada serviço, é realizada a busca semântica (*Semantic Search*) por serviços compatíveis com estes termos. Com os serviços descobertos são gerados modelos executáveis semântica e estruturalmente possíveis do *workflow* (*Executable Workflow Model*), que são apresentados para o usuário do sistema para que este escolha um modelo para gerar o WS-BPEL correspondente.

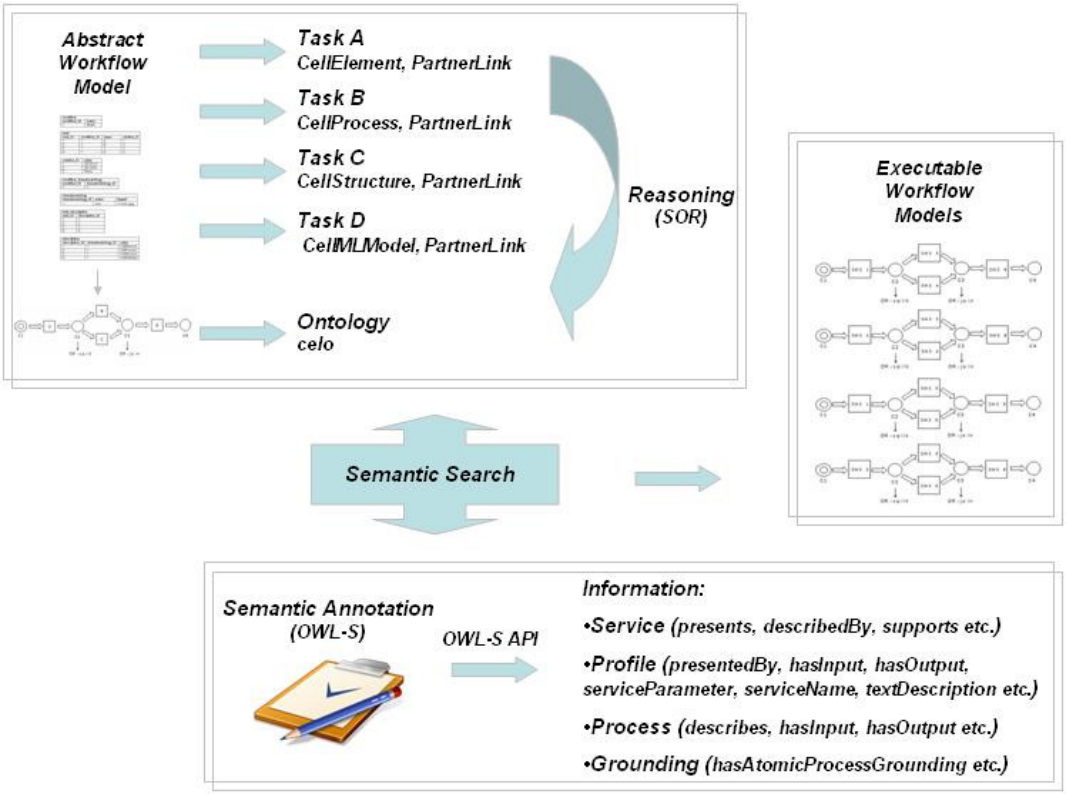


Figura 19 Obtenção de modelos de *workflows* executáveis a partir do modelo abstrato.

Os serviços encontrados são atribuídos a cada tarefa da rede que representa o *workflow* abstrato, conforme podemos observar na Figura 20.

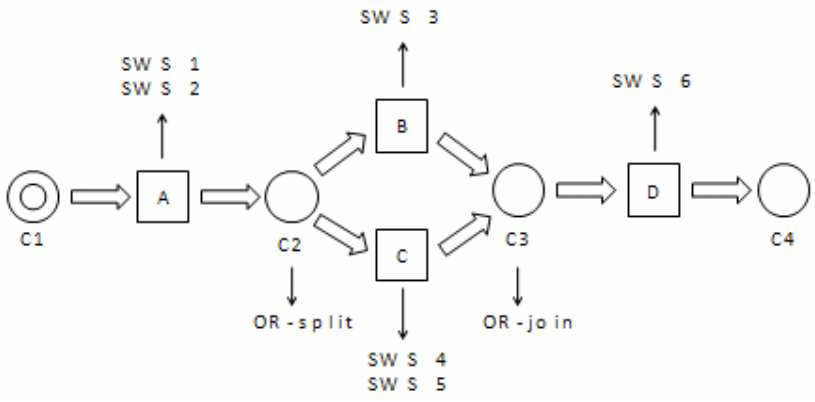


Figura 20 Rede de Petri que representa *workflow* abstrato, com atribuição de serviços.

Após a atribuição de serviços às tarefas do *workflow* abstrato, as composições, utilizando os serviços descobertos são definidas como redes que representam *workflows* executáveis. Estas redes são consideradas, no contexto deste trabalho, composições

semanticamente possíveis. Podemos visualizar as redes que representam as composições na Figura 21.

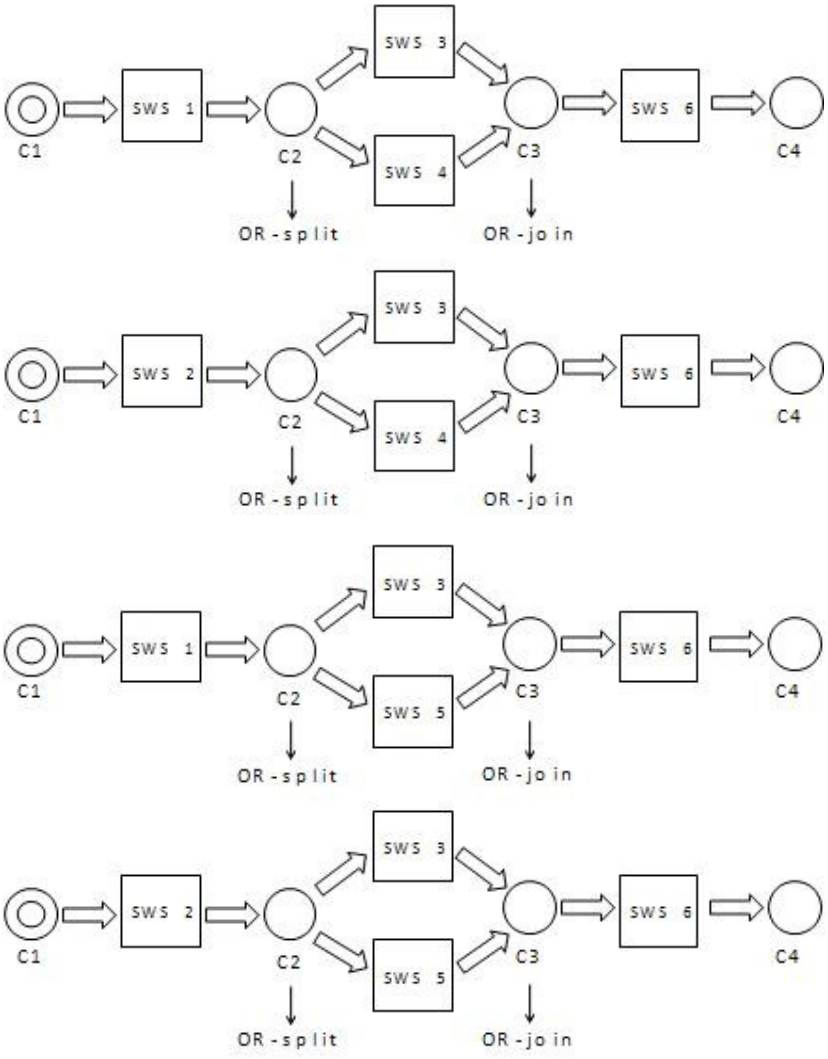


Figura 21 Redes de Petri que representam *workflows* executáveis.

Os modelos executáveis definidos (composições semanticamente possíveis) são armazenados na base de dados de forma que possam ser acessados posteriormente para a realização da análise estrutural dos serviços e definição das composições semântica e estruturalmente possíveis.

Na Figura 22, temos as tabelas da base de dados em que são armazenados os dados referentes às composições executáveis. Cada modelo semanticamente possível é armazenado na tabela *workflowexe*, referenciando o modelo abstrato ao qual se refere (*workflow_id*) e recebe o nome do modelo abstrato que o originou seguido de um número seqüencial. Por exemplo, os modelos semanticamente possíveis gerados a partir de um

modelo abstrato chamado *Teste* vão receber os nomes de *Teste1*, *Teste2*, *Teste3* e assim por diante. Esta tabela *workflowexe* possui o campo *possible* que indica se o modelo é também estruturalmente possível, e este campo tem como valor *default* o "Y", que significa que, a princípio, todos os modelos são, além de semanticamente possíveis, também estruturalmente possíveis. O valor deste campo é alterado posteriormente, após a análise estrutural dos serviços, quando o modelo em questão é classificado como semântica e estruturalmente possível ("Y") ou não ("N").

As tarefas que compõem o modelo semanticamente possível são armazenadas na tabela *taskexe* e referenciam as tarefas do modelo abstrato correspondentes (*task_id*). No campo *service* é armazenada a localização (URL) do serviço que foi descoberto e atribuído para realizar aquela tarefa. Os parâmetros de entrada e saída de cada serviço atribuído para a realização de cada tarefa são armazenados na tabela *io*, que contém o nome (*name*) do parâmetro, seu tipo (*iotype*) e a indicação (*inputoutput*) se o parâmetro é de entrada ("I") ou saída ("O"). A ligação entre o serviço (*taskexe*) e seus parâmetros (*io*) é feita por meio da tabela *taskexe_io*.

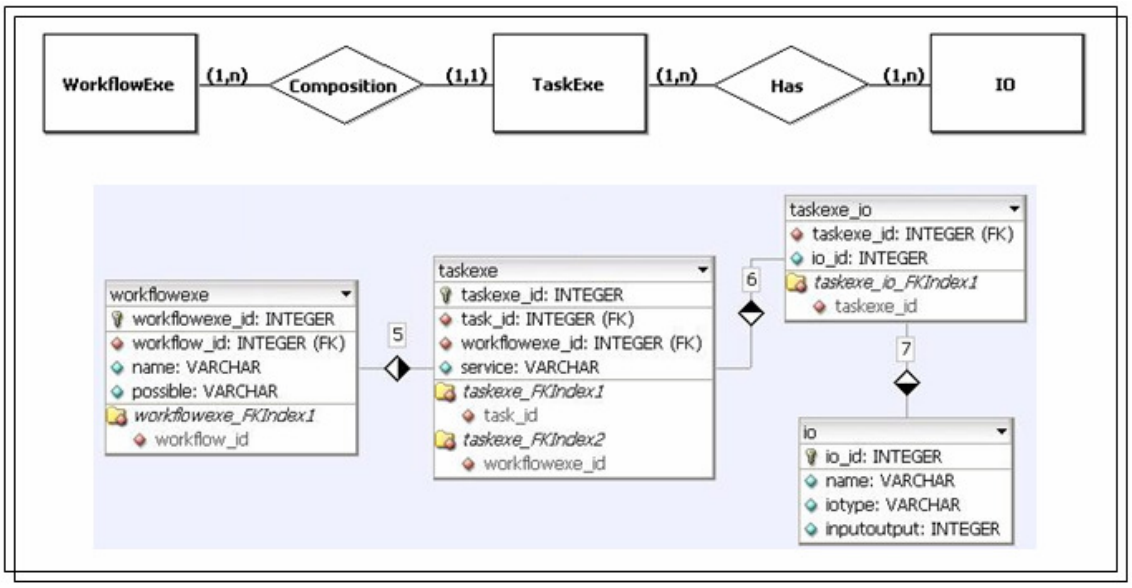


Figura 22 Estrutura de armazenamento de *workflows* executáveis.

Passo 3: Definição das composições semântica e estruturalmente possíveis

Cada serviço de cada composição semanticamente possível tem sua estrutura analisada de forma que possa ser definido se os serviços da composição são compatíveis

estruturalmente entre si e se a composição do *workflow* formada por eles é estruturalmente possível ou não. A análise estrutural está representada na Figura 23.

Na Figura 23, os parâmetros de saída do serviço atribuído à tarefa A do *workflow* abstrato devem ter tipos compatíveis com os parâmetros de entrada dos serviços atribuídos à tarefa B e à tarefa C. A relação da tarefa A com as tarefas seguintes B e C é *OR-Split*, ou seja, após a execução da tarefa A será executada a tarefa B ou a tarefa C, porém, no momento da composição, não há possibilidade de avaliar qual das duas tarefas (B ou C) será executada após a tarefa A, dessa forma, é necessário que a saída de A seja compatível com a entrada de B e de C, considerando a possibilidade de qualquer uma dessas tarefas serem executada.

Os parâmetros de saída dos serviços atribuídos à tarefa B e C devem apresentar tipos compatíveis com os parâmetros de entrada do serviço atribuído à tarefa D. Esta análise também é feita com base na relação das tarefas B e C com a tarefa seguinte D, a relação *OR-join*. A tarefa D será executada após a execução da tarefa B ou da tarefa C, então, seus parâmetros de entrada devem ser compatíveis com os parâmetros de saída das duas tarefas separadamente, considerando que qualquer uma das duas tarefas (B ou C) pode ser executada antes da tarefa D.

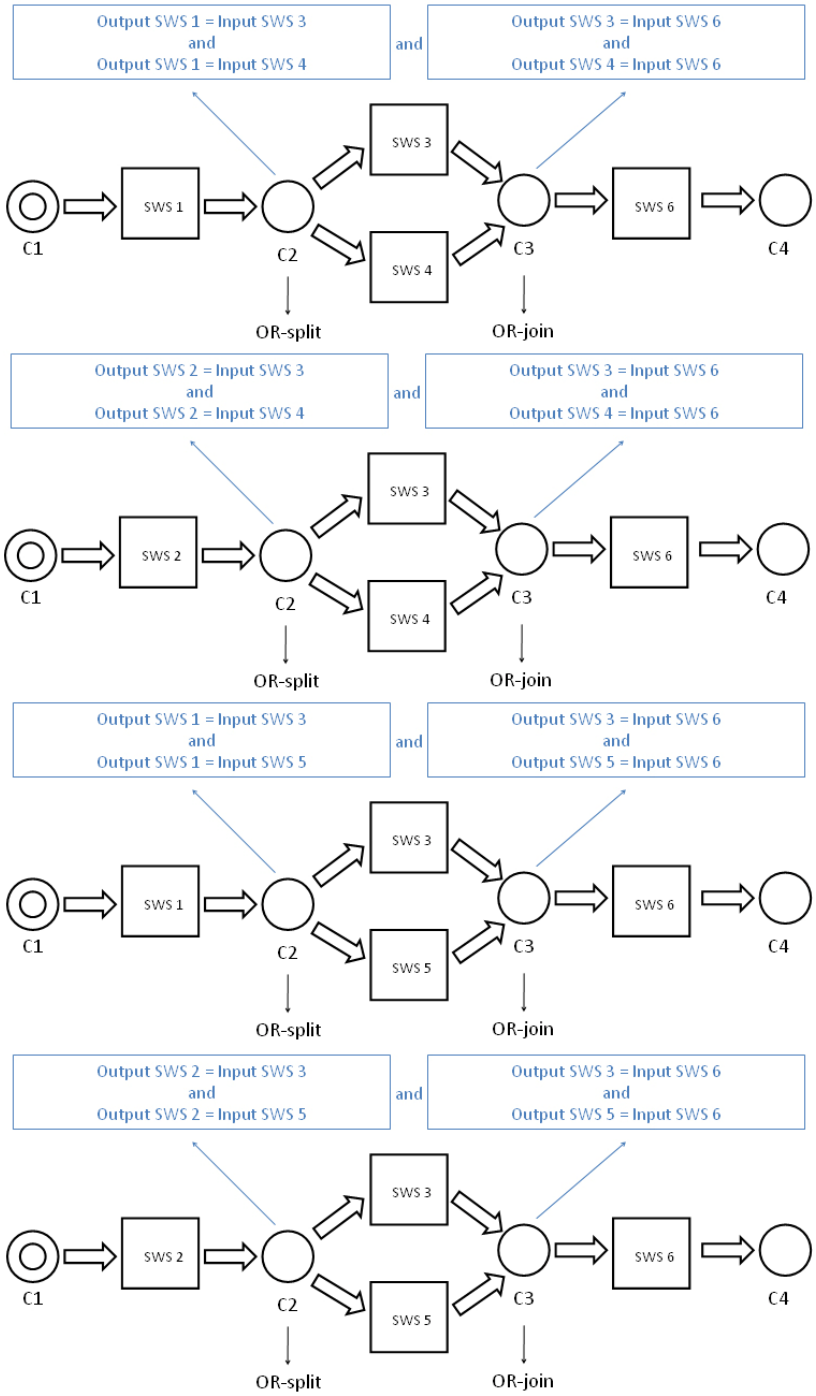


Figura 23 Análise estrutural dos serviços encontrados.

A análise da compatibilidade dos parâmetros de entrada e saída dos serviços da composição é feita levando em consideração apenas modelos simples de dados. No protótipo do *framework* desenvolvido foi realizado o batimento estrutural exato entre os tipos de dados de entrada e saída dos serviços, sem considerar questões relacionadas a

divergências de grandezas para tipos iguais e as transformações necessárias para contornar esse problema.

Neste trabalho serão abordados seis tipos de relações simples para as tarefas do *workflow*. Vamos considerar que uma tarefa poderá possuir uma das seguintes relações com a tarefa que a segue na definição do *workflow*:

- **Sequencial**. Se a tarefa A possui a relação *sequencial* com a tarefa B seguinte e a tarefa B possui qualquer relação com sua subsequente, a tarefa B só poderá ser executada ao fim da execução da tarefa A, e os parâmetros de saída (O_A) da tarefa A deverão ter tipos compatíveis com os parâmetros de entrada (I_B) da tarefa B. Teremos $\mathbf{O_A = I_B}$.
- **AND-Split** e **AND-Join**. Se a tarefa A possui a relação *And-Split,* é seguida da execução da tarefa B que possui a relação *AND-Join* que, por sua vez, é seguida da tarefa C que possui a relação *AND-Join* que, por sua vez, é seguida da tarefa D que possui a relação *final*, as tarefas B e C serão executadas ao mesmo tempo, ao fim da execução da tarefa A, e a tarefa D será executada ao fim da execução das tarefas B e C. Desta forma, os parâmetros de saída de A (O_A) deverão ter tipos compatíveis com os parâmetros de entrada de B (I_B) e C (I_C) somados e os parâmetros de B e C somados deverão ter tipos compatíveis com os parâmetros de entrada de D (I_D), ou seja, $\mathbf{O_A = I_B + I_C}$ **e** $\mathbf{O_B + O_C = I_D}$.
- **OR-Split** e **OR-Join**. Considerando o cenário em que a tarefa A possui a relação *OR-Split*, é seguida pela tarefa B com relação *OR-Join*, que é seguida pela tarefa C com relação *OR-Join* que, por fim, é seguida da tarefa D que possui relação *final*. Os parâmetros de saída de A (O_A) devem ter tipos compatíveis com os parâmetros de entrada de B (I_B) e com os parâmetros de entrada de C (I_C), e os parâmetros de saída de B (O_B) devem ter tipos compatíveis com os parâmetros de entrada de D (I_D) e os parâmetros de saída de C (O_C) devem ter tipos compatíveis com os parâmetros de entrada de D (I_D), ou seja, $\mathbf{(O_A = I_B}$ **e** $\mathbf{O_A = I_C)}$ **e** $\mathbf{(O_B = I_D}$ **e** $\mathbf{O_C = I_D)}$.
- **Final**. Se uma tarefa A possui a relação *final* significa que esta tarefa não é seguida pela execução de nenhuma outra. Desta forma, seus parâmetros de entrada serão analisados de acordo com a sua relação (uma das relações citadas anteriormente) com a tarefa anterior a ela, e seus parâmetros de saída não têm restrições.

O esquema de análise estrutural dos serviços é apresentado na Figura 24.

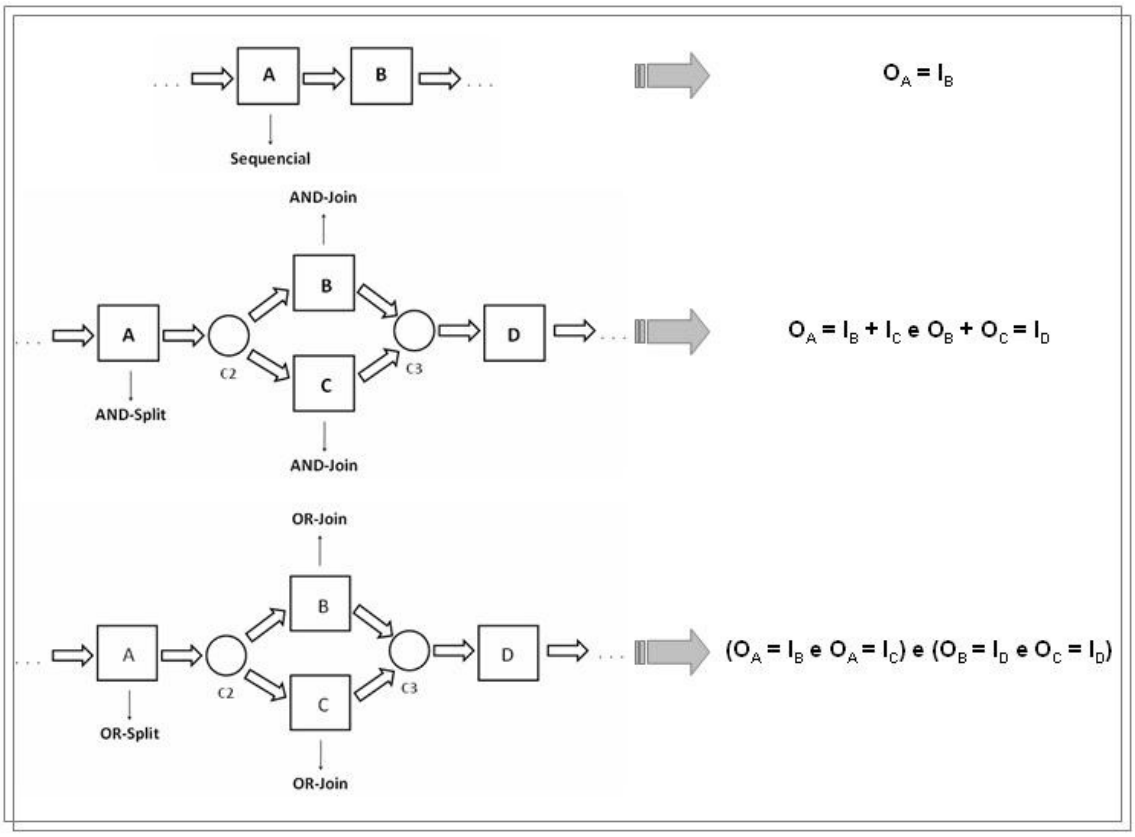


Figura 24 Esquema de análise estrutural.

No capítulo 5 é detalhado o mecanismo de análise estrutural dos serviços Web encontrados para compor um determinado *workflow*. É apresentado um exemplo e os códigos correspondentes para a realização dessa análise.

A partir de cada modelo executável do *workflow*, os dados de entrada e saída de cada serviço do modelo são obtidos a partir do WSDL do serviço. Os dados são extraídos do WSDL por meio do uso da biblioteca Java WSDL4J [WSDL4J, 2005]. O projeto WSDL4J é *open source* e permite a criação, representação e manipulação de documentos WSDL.

O código Java do exemplo da Figura 25 extrai os nomes das mensagens (*Message*) "*request*" e "*response*" do WSDL e de seus parâmetros (*PartName*) de entrada (parâmetros de *request*) e saída (parâmetros de *response*), com seus respectivos tipos de dados (*PartTypeName*).

Na Figura 26 temos um esquema geral da análise estrutural dos serviços para a definição da possibilidade estrutural dos modelos semanticamente possíveis. As informações (nome, tipo) sobre os parâmetros de entrada e saída de um serviço são extraídos de seu WSDL. Com os tipos dos dados, é realizada a análise estrutural da compatibilidade entre os serviços dentro do modelo do *workflow* e, desta forma, são definidos os modelos semântica e estruturalmente possíveis.

```
public void wsdlData(String wsdlFile) throws WSDLException{
    WSDLFactory factory = WSDLFactory.newInstance();
    WSDLReader reader = factory.newWSDLReader();
    Definition wsdlInstance = reader.readWSDL(null, wsdlFile);
    Map messages = wsdlInstance.getMessages();
    Iterator msgIterator = messages.values().iterator();
    while (msgIterator.hasNext())
    {
        Message msg = (Message)msgIterator.next();
        if (!msg.isUndefined())
        {
            System.out.println("Message: "+msg.getQName());
            Map parts = msg.getParts();
            for(Iterator pIt = parts.values().iterator();
                pIt.hasNext();){
                Part part = (Part) pIt.next();
                System.out.println("PartName: "+part.getName());
                System.out.println("PartTypeName: "+
                        part.getTypeName().toString());
            }
        }
    }
}
```

Figura 25 Código Java usando WSDL4J para extrair informações de documento WSDL.

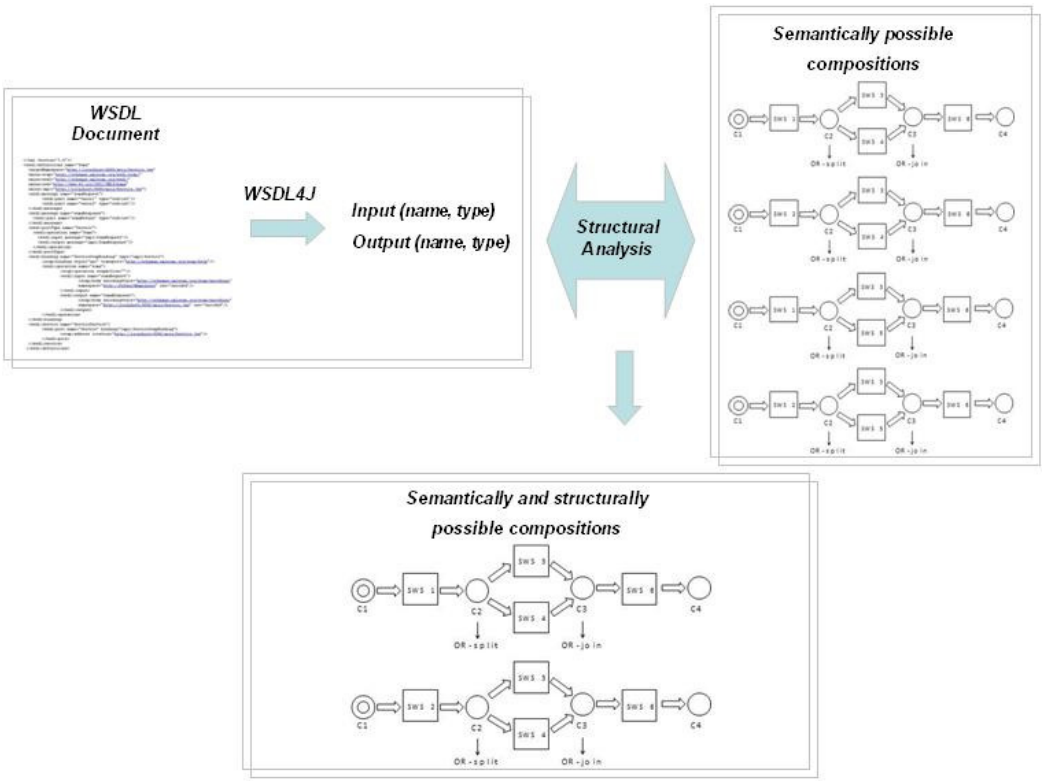


Figura 26 Esquema da análise estrutural dos serviços para a definição de composições semântica e estruturalmente possíveis.

5. ESTUDO DE CASO

Neste capítulo serão apresentados cenários de uso para a utilização do *framework* Composer-Science. Os cenários são apresentados levando em consideração a forma de utilização do *framework* por um usuário final (cientista ou grupo de cientistas) para auxiliar na tarefa de criação de *workflows* científicos capazes de representar seus experimentos.

Um *workflow* científico da área de bioinformática, definido em [WILLIAMS, 2009], será utilizado para ilustrar o primeiro cenário de uso (que representa o principal objetivo do *framework* proposto) e para detalhar o funcionamento do *framework*.

Cenário de uso 1

Um usuário, que pode ser um pesquisador ou grupo de pesquisadores, deseja realizar a simulação computacional de um experimento científico, mas não dispõe de aplicativos que executem todas as etapas do mesmo.

O usuário faz uso do *framework* Composer-Science para obter um *workflow* científico composto por serviços Web que represente seu experimento. O uso do *framework* tem como principal objetivo reduzir o tempo gasto na implementação computacional do experimento. O *framework* possibilita definir semanticamente o modelo de workflow científico desejado de maneira simples e intuitiva, reduzindo o tempo gasto com análises e discussões que irão levar a correta modelagem do experimento. O *framework* realiza a busca semântica de serviços Web que irão compor o *workflow*, especificando para o usuário um modelo WS-BPEL que pode ser executado em uma *engine* capaz de interpretar essa linguagem. A busca semântica automática de serviços para compor o *workflow* dispensa o trabalho manual do usuário para realizar a descoberta de serviços dentre um número elevado de serviços disponíveis na Web ou em um repositório, reduzindo o tempo de busca.

Por meio da interface do *framework*, o usuário cria seu modelo de *workflow* definindo um nome para identificá-lo. A Figura 27 apresenta a interface do protótipo para criação de um novo modelo de *workflow*. O usuário define um nome para identificar seu *workflow* no campo *Workflow name* e salva o registro pressionando o botão *Save*. O botão *New* permite que outro modelo seja criado. Os *links Associate Domain Ontology to Workflow* e *Back to Menu* direcionam, respectivamente, para as páginas que permitem a associação de ontologias de domínio ao modelo de *workflow* (as ontologias de domínio que serão

utilizadas para descrever semanticamente o *workflow* e suas tarefas), e para o menu principal do *framework*.

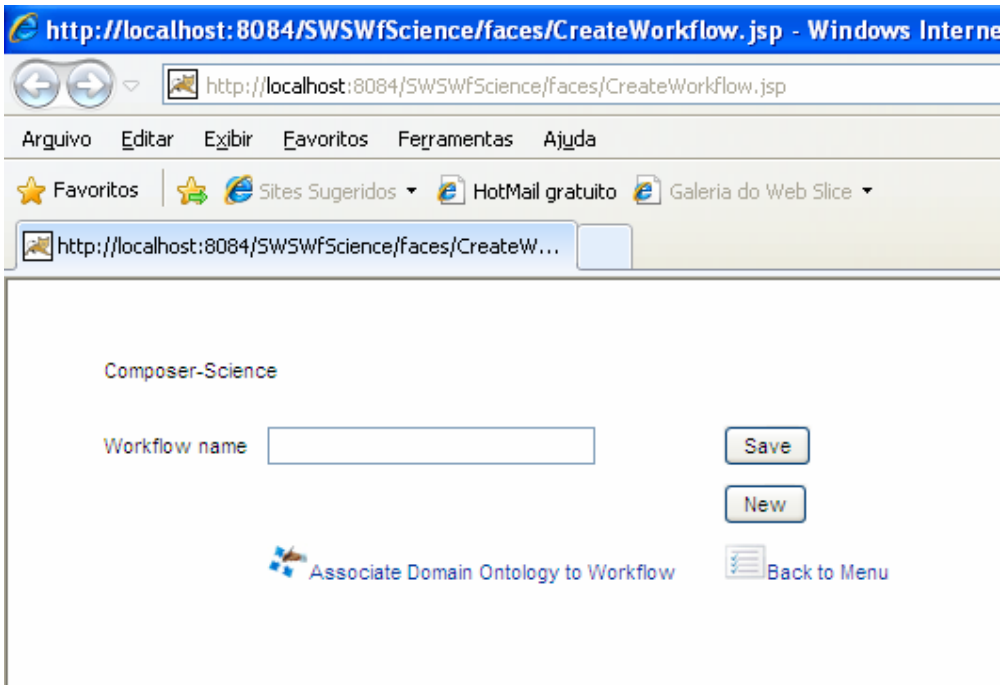


Figura 27 Interface para criação de modelo de *workflow*.

O usuário pode selecionar as ontologias de domínio disponíveis no *framework* e visualizar suas estruturas de forma a compreender a terminologia proposta por cada uma delas e utilizá-las para descrever o modelo do *workflow* científico que deseja obter ao final da execução do sistema. É possível também visualizar a estrutura da ontologia WS-BPELOnto descrita no capítulo 4 e das ontologias de domínio disponíveis, como é o caso da CelO [MATOS et al., 2009]. A Figura 28 apresenta a interface do protótipo para a visualização da estrutura de uma ontologia. O usuário seleciona uma das ontologias disponibilizadas pelo *framework* em uma lista (campo *Ontology*) e pressiona o botão *View* que exibe a visualização da ontologia escolhida. O *link Back to Menu* direciona para o menu principal do *framework*.

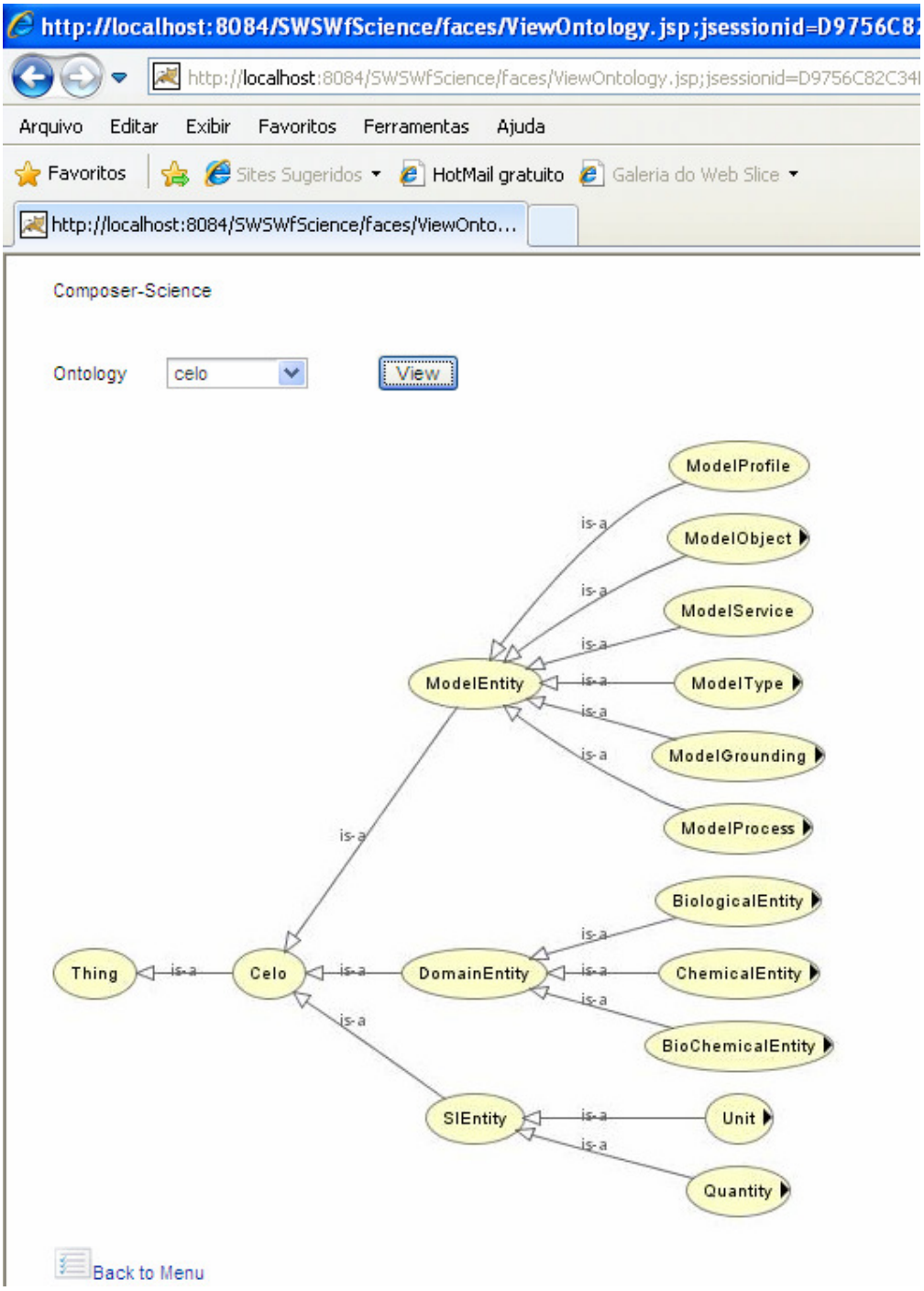


Figura 28 Interface para visualização de ontologia.

O usuário associa as ontologias que descrevem o domínio ao qual seu experimento se insere ao modelo de *workflow* que criou. Por meio de termos presentes nessas ontologias associadas, é possível descrever o objetivo global do *workflow*.

A Figura 29 apresenta a interface por meio da qual é possível associar ontologias de domínio ao modelo de *workflow* criado (a ontologia de modelagem de *workflow* WS-

BPELOnto é associada automaticamente). O usuário seleciona o modelo de *workflow* em uma lista (campo *Workflow*) e pressiona o botão *Charge Ontologies* para que as ontologias de domínio disponíveis no *framework* sejam carregadas na lista *All Ontologies* (esquerda), e para que as ontologias que já estão associadas ao modelo sejam carregadas na lista *Selected Ontologies* (direita). Os botões >> (superior) e << (inferior), entre as duas listas de ontologias (*All Ontologies* e *Selected Ontologies*), tem as funções, respectivamente, de associar uma nova ontologia ao modelo e de desfazer a associação de uma ontologia ao modelo. O *link Associate Description to Workflow* direciona para a página por meio da qual é possível realizar a descrição semântica do *workflow*. O *link Back to Menu* direciona para o menu principal, que contém *links* para todas as páginas do *framework*.

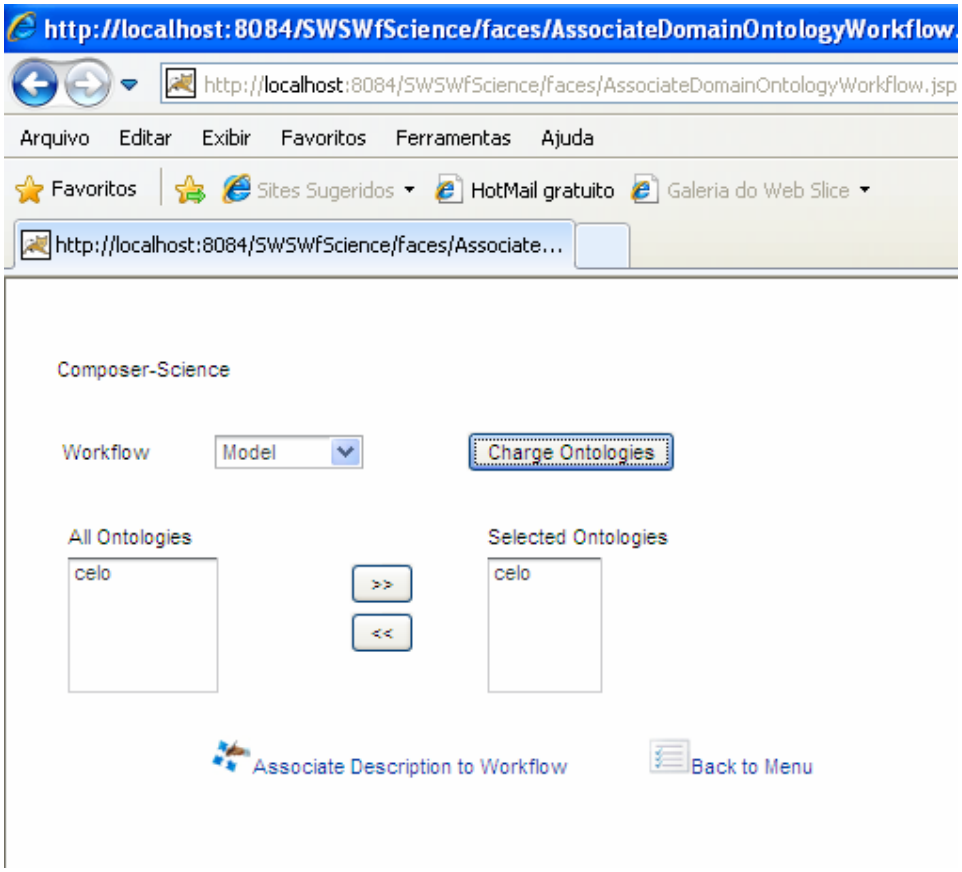


Figura 29 Interface para associação de ontologias de domínio a um modelo de *workflow*.

A Figura 30 apresenta a interface por meio da qual é possível utilizar termos das ontologias de domínio associadas ao modelo criado e da ontologia WS-BPELOnto para descrever o *workflow* como um todo. O campo *Workflow* apresenta a lista dos modelos de *workflows* registrados no *framework*. O usuário seleciona o modelo e pressiona o botão *Charge Data* para que sejam carregados nos campos *Domain Description* e *Workflow*

Description os termos, respectivamente, das ontologias de domínio associadas ao modelo escolhido e da ontologia WS-BPELOnto.

Uma vez definidos os termos para descrever o *workflow*, o botão *Save* é pressionado para salvar as descrições do modelo. Se for necessário adicionar alguma nova descrição ao modelo de *workflow*, o botão *New* esvazia os campos e permite a criação de um novo registro.

Os *links Create Task* e *Back to Menu* direcionam, respectivamente, para as páginas de criação de novas tarefas para o *workflow* e para o menu principal do *framework*.

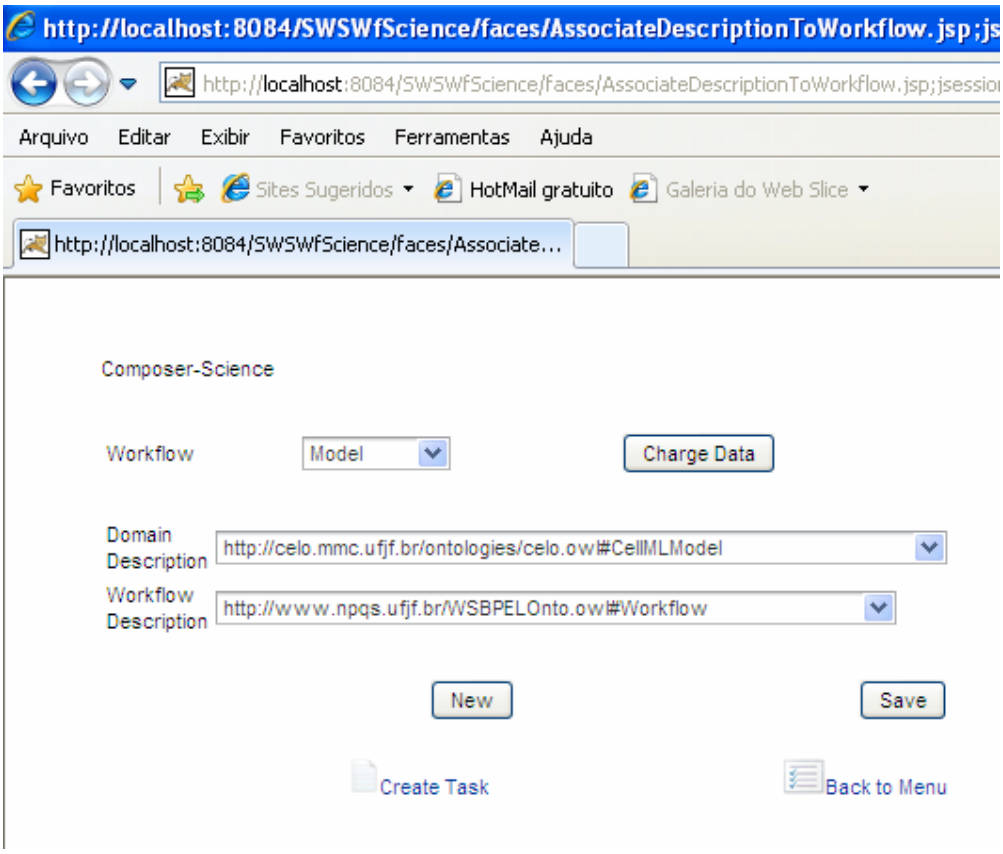


Figura 30 Descrição semântica de *workflow*.

O usuário registra cada uma das tarefas que irão compor seu *workflow*, indicando seu nome, a relação da tarefa com a próxima tarefa do *workflow* e a sequência em que ela será executada. A Figura 31 apresenta a interface para a criação das tarefas que irão compor o *workflow*. No campo *Workflow* são listados os modelos de *workflows* existentes no sistema, e então o usuário escolhe aquele que deseja modificar por meio da inserção de uma nova tarefa. O usuário pode criar uma nova tarefa definindo seu nome no campo *Task*, sua relação com a próxima tarefa do *workflow* no campo *Relation with next task*, e sua sequência de execução no campo *Sequence*. Para salvar a tarefa criada, é necessário usar

o botão *Save*, e para criar uma nova tarefa, o botão *New*. Os *links Associate Domain Description to Task* e *Back to Menu* direcionam, respectivamente, para a página que permite a descrição semântica da tarefa e para o menu principal do sistema.

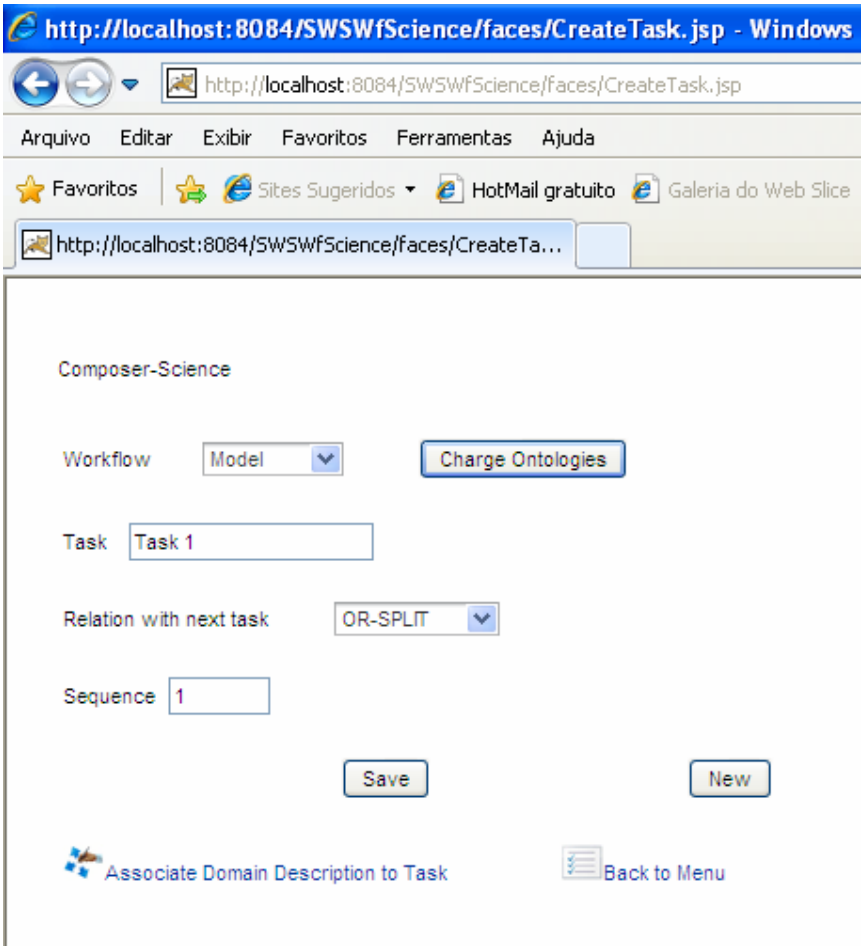


Figura 31 Interface para criação de tarefa de *workflow*.

A essas tarefas que irão compor o *workflow*, é possível associar descrições semânticas, oriundas das ontologias de domínio associadas ao modelo, que definam seu objetivo, seus dados de entrada e de saída. A Figura 32 apresenta a interface que permite a descrição semântica de cada tarefa por meio da utilização de termos oriundos das ontologias de domínio associadas ao modelo de *workflow* (campo *Domain Description*) ao qual a tarefa pertence, e também por meio da utilização de termos da ontologia WS-BPELOnto (campo *Workflow Description*). Pode-se associar, por meio desta interface, mais de um termo da ontologia para descrever semanticamente a mesma tarefa.

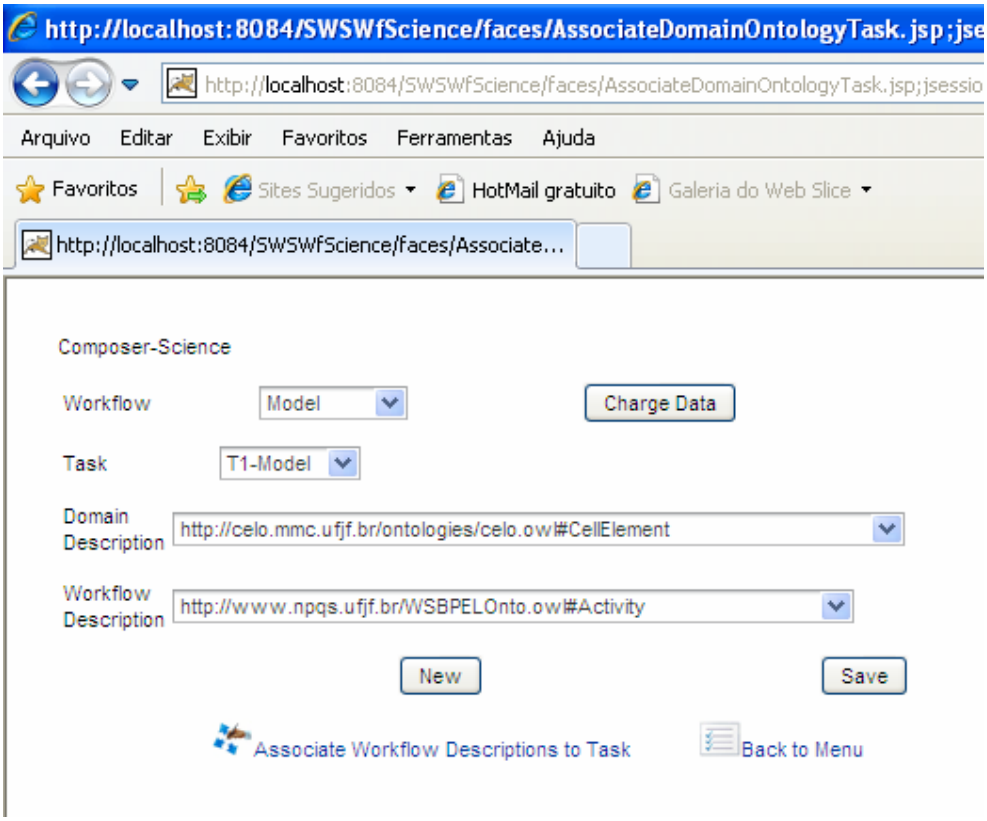


Figura 32 Interface para a descrição semântica de tarefa.

A cada uma das descrições usadas para definir as tarefas, devem ser obrigatoriamente associadas descrições oriundas da ontologia WS-BPELOnto, que irão definir o contexto do modelo WS-BPEL ao qual cada descrição associada à tarefa se insere.

Terminada a etapa de criação do modelo abstrato do *workflow* científico, o usuário do sistema solicita o processamento do sistema, que busca os serviços que atendam ao modelo especificado, e realiza as composições dos serviços.

Após o processamento do sistema, as três primeiras composições realizadas são apresentadas ao usuário para que ele escolha uma das composições e solicite a geração do modelo WS-BPEL correspondente a ela. A Figura 33 apresenta a interface de visualização das composições realizadas e solicitação da geração do modelo WS-BPEL. O usuário seleciona o modelo de *workflow* na lista do campo *Workflow* e solicita o processamento por meio do botão *Search*. As composições semântica e estruturalmente possíveis são apresentadas nas tabelas e o usuário solicita a geração do modelo WS-BPEL de uma das composições por meio do botão *Generate WS-BPEL* localizado a sua direita.

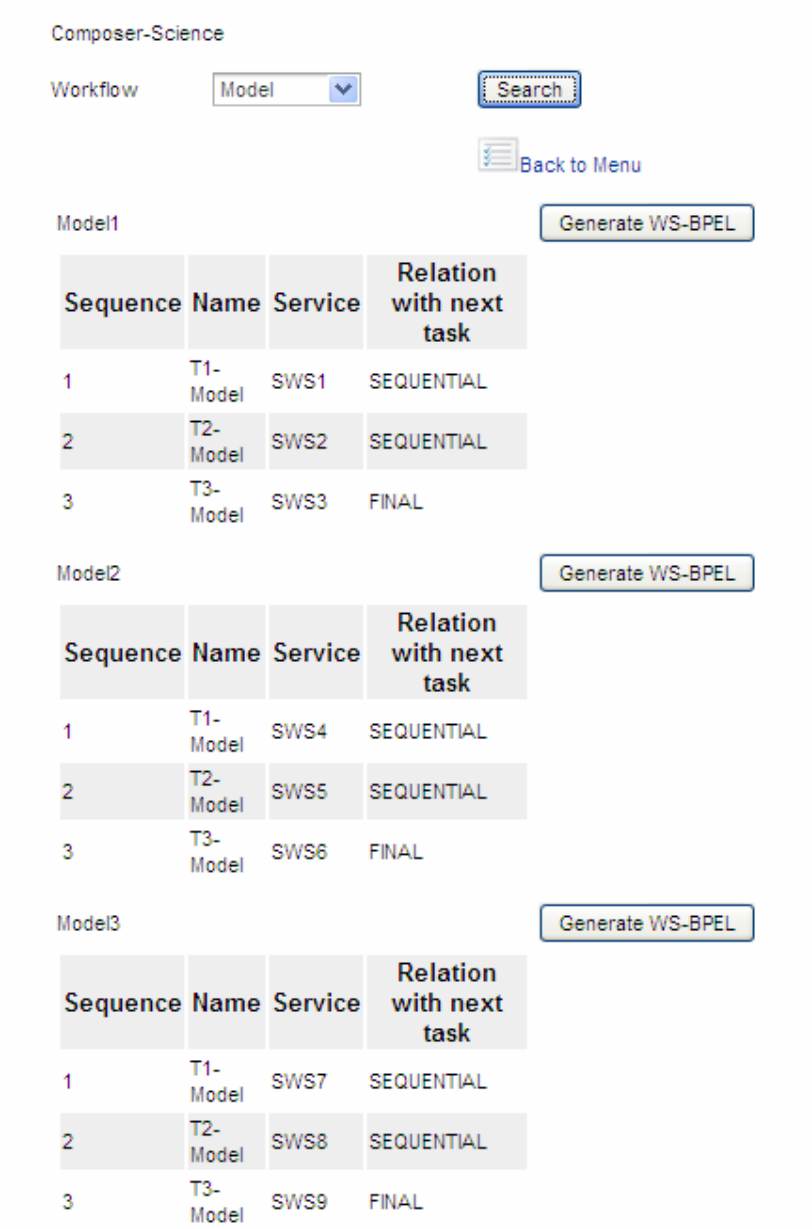


Figura 33 Interface para visualização de composições e geração de WS-BPEL.

Com o modelo WS-BPEL em mãos, o pesquisador poderá executá-lo em uma *engine* (Sistemas de Gerenciamento de Workflows Científicos) que interprete e execute modelos nesta linguagem.

Cenário de uso 2

O pesquisador desenvolveu um serviço Web que realiza uma tarefa científica relacionada a um domínio de interesse e deseja compartilhar este serviço com outros pesquisadores, de forma que eles possam reutilizá-lo em seus experimentos. O serviço criado é um serviço Web semântico e, dessa forma, possui uma descrição semântica (OWL-S) além de um WSDL que o descreve.

Este pesquisador pode registrar o serviço nos repositórios do *framework* Composer-Science, possibilitando assim, que este serviço possa ser reutilizado para compor *workflows* científicos de outros pesquisadores.

A Figura 34 apresenta a interface do *framework* por meio da qual é possível registrar um novo serviço em seus repositórios. O usuário deve definir o nome do serviço no campo *Service name*, e inserir os arquivos OWL-S e WSDL do serviço, respectivamente, nos campos *OWL-S* e *WSDL* da interface e, em seguida, pressionar o botão *Registry* para salvar os arquivos no repositório do *framework*. Ao pressionar o botão *New*, os campos são esvaziados e assim é possível inserir um novo serviço.

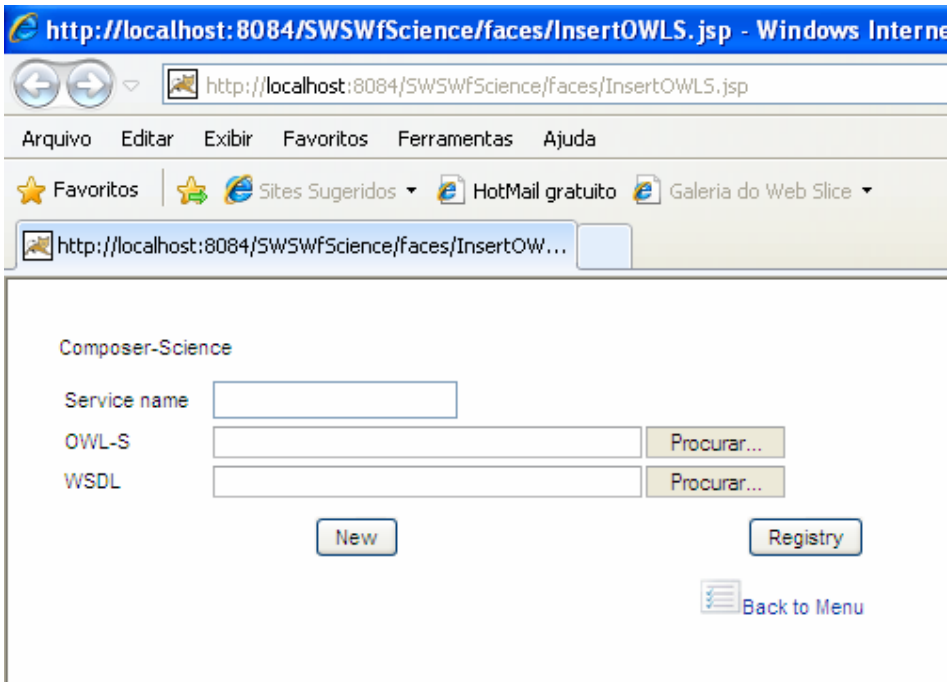


Figura 34 Interface para registro de serviço Web semântico.

Cenário de uso 3

O pesquisador necessita descrever semanticamente um *workflow* científico que deseja criar. Ao visualizar as ontologias de domínio disponíveis no *framework* Composer-Science, o pesquisador conclui que nenhuma delas apresenta a terminologia adequada para descrever semanticamente as tarefas do *workflow*.

O pesquisador necessita, então, de uma ontologia que não se encontra disponível no *framework*. Sendo assim, ele pode registrar essa ontologia no *framework* e utilizá-la para realizar a busca de serviços úteis na composição do *workflow* científico correspondente ao seu experimento.

A Figura 35 apresenta a interface do *framework* por meio da qual o usuário pode registrar novas ontologias de domínio para serem utilizadas na descrição semântica de um *workflow* científico. O usuário registra o nome da ontologia no campo *Ontology name*. O arquivo OWL da ontologia a ser registrada deve ser carregado no campo *OWL*. No campo *Image* deve ser carregado um arquivo de imagem que ilustra a estrutura da ontologia, e que será apresentado ao usuário na interface de visualização de estrutura de ontologia de domínio. O botão *Registry* registra a ontologia no *framework*. Ao pressionar o botão *New* os campos são esvaziados e é possível inserir uma nova ontologia. O *link Back to Menu* direciona para o menu principal do *framework*.

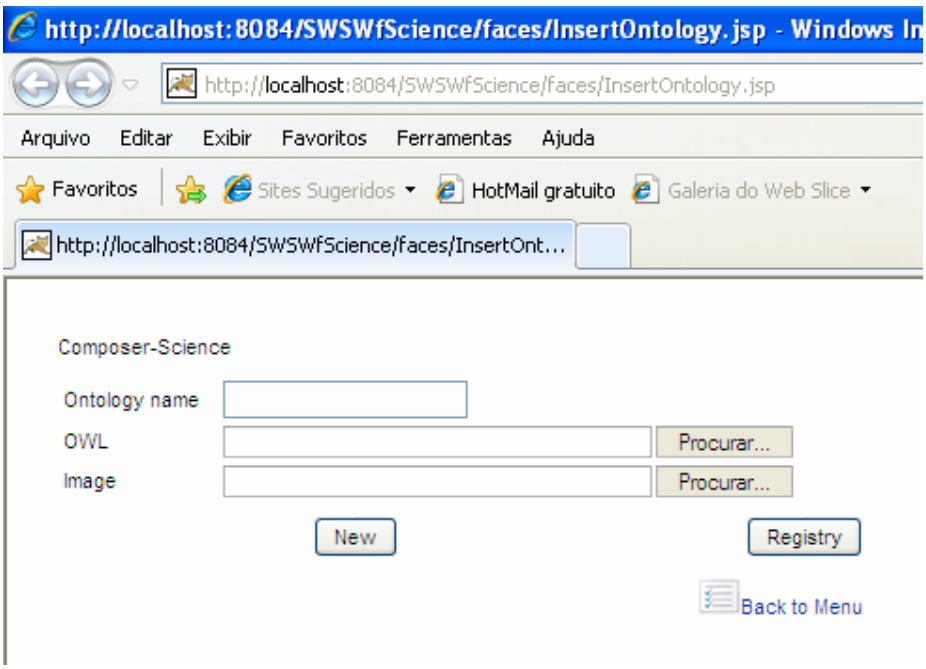


Figura 35 Interface para registro de ontologia.

5.1. EXEMPLO DO CENÁRIO DE USO 1

Considerando o caso de uso 1, que representa o principal objetivo do *framework* proposto, que é encontrar serviços Web semânticos e compor um *workflow* científico com eles a partir da definição semântica do *workflow* e de suas tarefas, vamos utilizar o exemplo de um *workflow* científico simples, relacionado a área de Bioinformática [WILLIAMS, 2009], para demonstrar o funcionamento da arquitetura proposta e do protótipo desenvolvido.

5.1.1 Contexto Biológico

As ciências relacionadas com o estudo da vida passaram por uma revolução com a transição dos métodos tradicionais de descoberta *in-vivo*[3] para os métodos de descoberta *in-silico*[4]. Esses métodos permitiram reduzir o tempo e o custo associados com a descoberta do conhecimento biológico. Consequentemente, uma nova disciplina - chamada Bioinformática - foi criada, unindo em uma única disciplina a Biologia e a Ciência da Computação. A Bioinformática objetiva o gerenciamento e análise dos dados biológicos usando técnicas avançadas de computação [MACEDO, 2005].

A pesquisa em bioinformática pode compreender desde a abstração das propriedades de um sistema biológico em um modelo matemático ou físico até a implementação de novos algoritmos para a análise de dados, o desenvolvimento de bancos de dados e das ferramentas Web para acessá-los [GIBAS e JAMBECK, 2001; LESK, 2008].

Dados biológicos encontram-se espalhados em diferentes bancos de dados, com diferentes sistemas de gerenciamento e formatos. Muitos desses dados estão disponíveis na Web por meio de arquivos ou de interfaces que permitam algum tipo de busca. Diversas instituições disponibilizam, além de dados, serviços Web que permitem o acesso e a pesquisa sobre dados armazenados:

- *DNA Data Bank of Japan (DDBJ)*. O DDBJ [DDBJ, 2010] iniciou suas atividades em banco de dados de DNA em 1986 no *National Institute of Genetics* (NIG). O DDBJ tem colaborado com o EBI (*European Bioinformatics Institute*, responsável pela base de dados EMBL na Europa) e com o NCBI (*National Center for Biotechnology Information*, responsável pela base de dados

[3] A expressão *in vivo* se refere à experimentação feita dentro ou no tecido vivo de um organismo vivo. Experimentos com animais e os ensaios clínicos são formas de investigação *in vivo* [LEMOS, 2004].
[4] A expressão *in silico* é, originalmente, usada para denotar simulações computacionais que modelam um processo natural ou de laboratório [LEMOS, 2004].

GenBank nos Estados Unidos) através do intercâmbio de dados e informações pela Internet e pela sua regular participação nas reuniões dessas instituições, a *International DNA Data Banks Advisory Meeting* e a *International DNA Data Banks Collaborative Meeting*. O DDBJ é o único banco de dados de DNA no Japão que é oficialmente certificado para recolher sequências de DNA de pesquisadores, e de emitir o número reconhecido internacionalmente para adesão dos dados fornecidos. O DDBJ coleta dados principalmente de pesquisadores japoneses, mas também aceita dados e emite o número de adesão (ID) para investigadores de diversos países. O DDBJ também fornece ferramentas para recuperação e análise dos dados disponibilizados.

- *EMBL-EBI*. O EBI (*European Bioinformatics Institute*) é parte integrante do EMBL (*European Molecular Biology Laboratory*). O EMBL-EBI [EMBL-EBI, 2010] foi o primeiro banco de dados do mundo de sequência nucleotídica, tendo surgido em 1980 em Heidelberg, na Alemanha. Seus grupos de investigação visam compreender a biologia através do desenvolvimento de novas abordagens para a interpretação dos dados biológicos.
- *GenBank.* O GenBank [GenBank, 2010] é um banco de dados de sequências genéticas e uma coleção de anotações de sequências de DNA disponíveis. Os lançamentos de novas versões são feitos a cada dois meses. O GenBank é parte do *International Nucleotide Sequence Database Collaboration*, que inclui o DDBJ, o EMBL e o GenBank. O NCBI está continuamente desenvolvendo novas ferramentas e atualizando as já existentes para melhorar a apresentação e o acesso ao GenBank. Não há restrições à utilização ou distribuição dos dados do GenBank.
- *BioMart*. O BioMart [BioMart, 2010] é um sistema robusto de integração de dados baseado em mineração de dados (*data warehousing*) distribuída. O sistema pode ser aplicado a um ou a múltiplos bancos de dados, suportando pesquisa de grande volume de dados. O sistema consiste em um esquema do banco de dados, ferramentas de administração e *softwares* de acesso a dados, que incluem interfaces Web ou *standalone*.

Um dos principais objetivos do Projeto Genoma Humano [DOE, 2010; NIH, 2010] é a compreensão mais ampla da organização do genoma e da função dos genes humanos pela comparação com outros genomas já seqüenciados. Essa comparação é importante, pois há um princípio biológico que diz que se duas sequências, sejam nucleotídicas ou protéicas, são similares, então é razoável supor que suas funções também sejam similares.

Ao isolar novas sequências moleculares em laboratório, os pesquisadores querem saber o máximo possível sobre elas. O primeiro passo para isso é verificar se outros

pesquisadores já estudaram sequências similares a essas. Os alinhamentos oferecem um ótimo meio de comparar sequências relacionadas. Nesse contexto, podemos considerar relevante o *workflow* utilizado como exemplo, que será detalhado a seguir na seção 5.1.2, pois ele permite a realização do alinhamento de sequências.

5.1.2 *Workflow* para Alinhamento de Sequências

Será utilizado como exemplo um *workflow* científico simples adaptado do *workflow* original apresentado em [WILLIAMS, 2009]. Uma visão geral do *workflow* é apresentada no Quadro 2. O *workflow* recupera de um banco de dados biológico as sequências genômicas de três espécies, alinha as sequências e exibe os resultados do alinhamento.

Quadro 2 *Workflow* para recuperação e alinhamento das sequências de três espécies.

Workflow
Acessar um banco de dados biológico e recuperar uma sequência de DNA da primeira espécie.
Acessar um banco de dados biológico e recuperar uma sequência de DNA da segunda espécie.
Acessar um banco de dados biológico e recuperar uma sequência de DNA da terceira espécie.
Ler as três sequências recuperadas e formatá-las.
Realizar o alinhamento das três sequências.
Exibir sequências alinhadas.

A Figura 36 apresenta a estrutura de execução do *workflow* que se dá da seguinte forma: 1) As sequências são recuperadas pela execução, em paralelo, de serviços que recebem como dados de entrada informações sobre qual banco de dados acessar e qual sequência (de qual espécie) recuperar. Esses serviços são capazes de acessar um banco de dados biológico e recuperar as sequências indicadas. 2) Em seguida, um serviço recebe como entrada as sequências recuperadas e realiza sua formatação. 3) Um outro serviço recebe o agrupamento de sequências, e realiza alinhamento múltiplo. 4) Os resultados do alinhamento são passados para o próximo serviço do *workflow* que, então, os exibe.

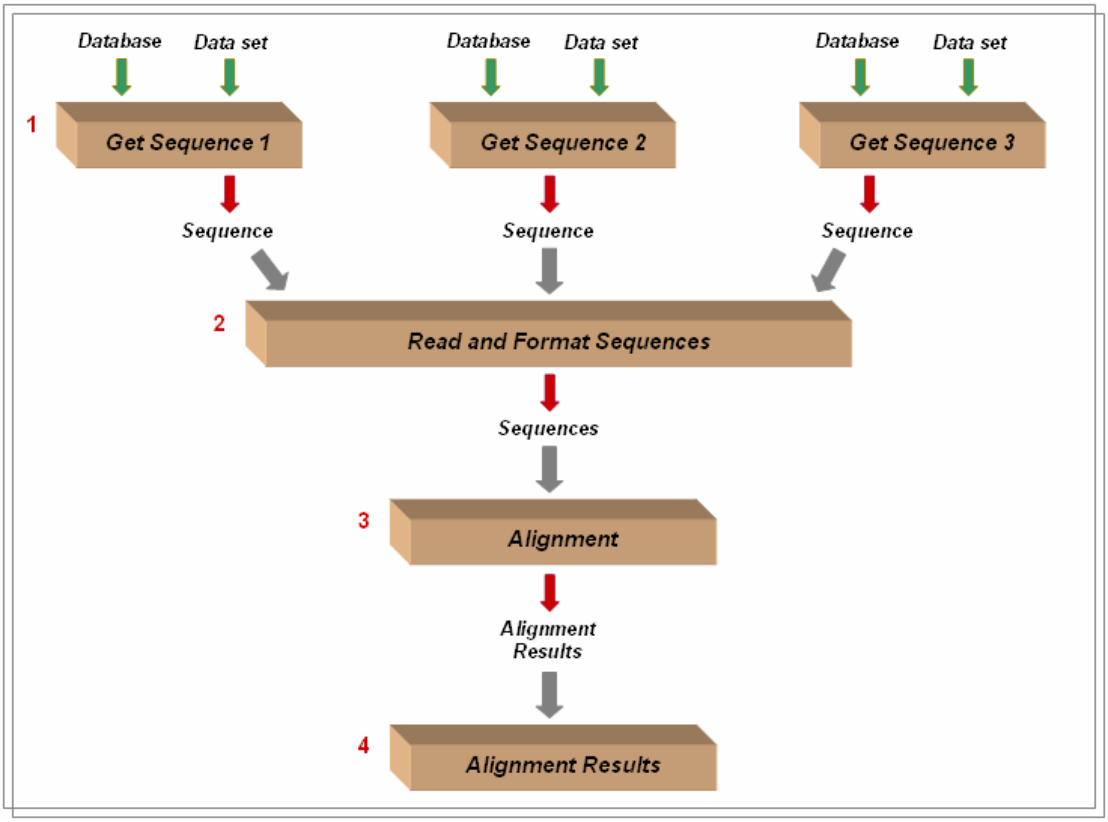


Figura 36 Estrutura de execução de *workflow* de alinhamento genético.

5.1.3 Definição do *Workflow*

Para descrever o *workflow* que realiza o alinhamento genético das sequências, será utilizada uma versão adaptada da ontologia para sistemas de gerência de análises em biossequências proposta em [LEMOS, 2004].

A ontologia proposta em [LEMOS, 2004] possui classes relevantes ao domínio de análise em biossequências, suas propriedades e como elas se relacionam. As principais classes da ontologia referem-se a processos, recursos, dados e projetos comumente envolvidos em análises de biossequências. A ontologia é flexível, ou seja, permite acrescentar, atualizar e remover classes, propriedades e instâncias.

A Figura 37 apresenta o nível topo da ontologia adaptada.

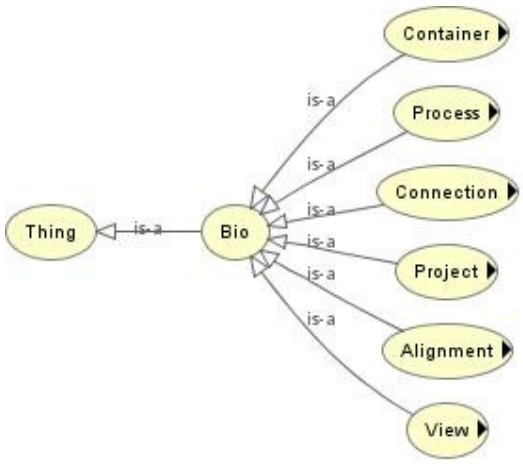


Figura 37 Nível topo da ontologia para sistemas de gerência de análises em biossequências adaptada.

Para obter uma composição de serviços Web semânticos, que represente o *workflow* utilizado como exemplo para o estudo de caso, devemos criar o modelo do *workflow* no *framework* Composer-Science.

A Figura 38 mostra a criação do modelo por meio da interface do *framework*.

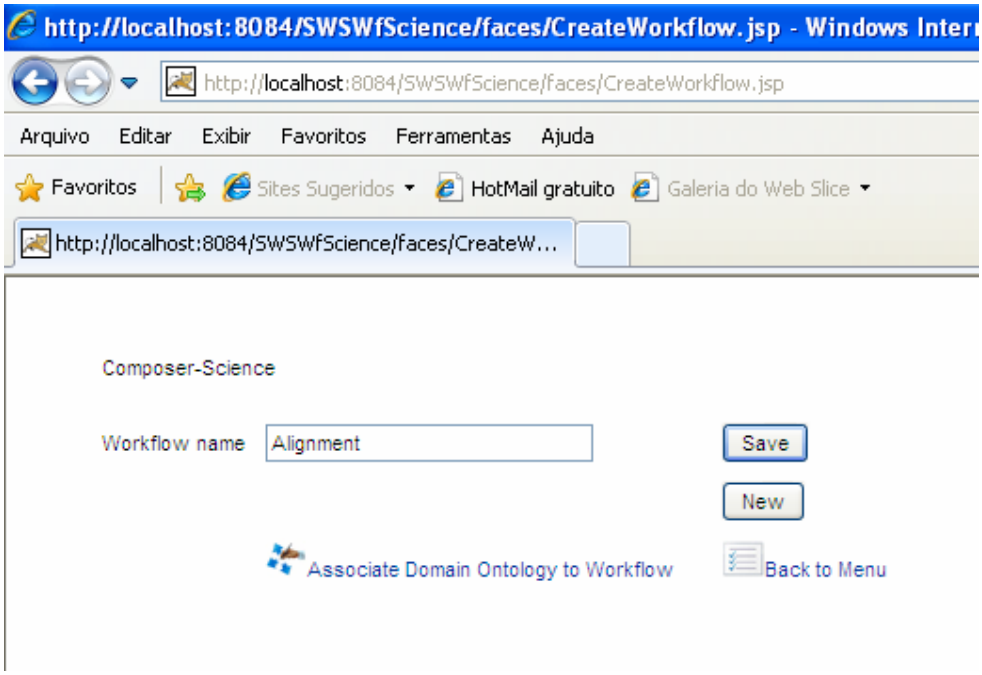


Figura 38 Criação de modelo de *workflow*.

A Figura 39 apresenta a associação de ontologias de domínio ao modelo de *workflow* criado. As ontologias associadas ao modelo são utilizadas para descrever semanticamente o *workflow* e suas tarefas.

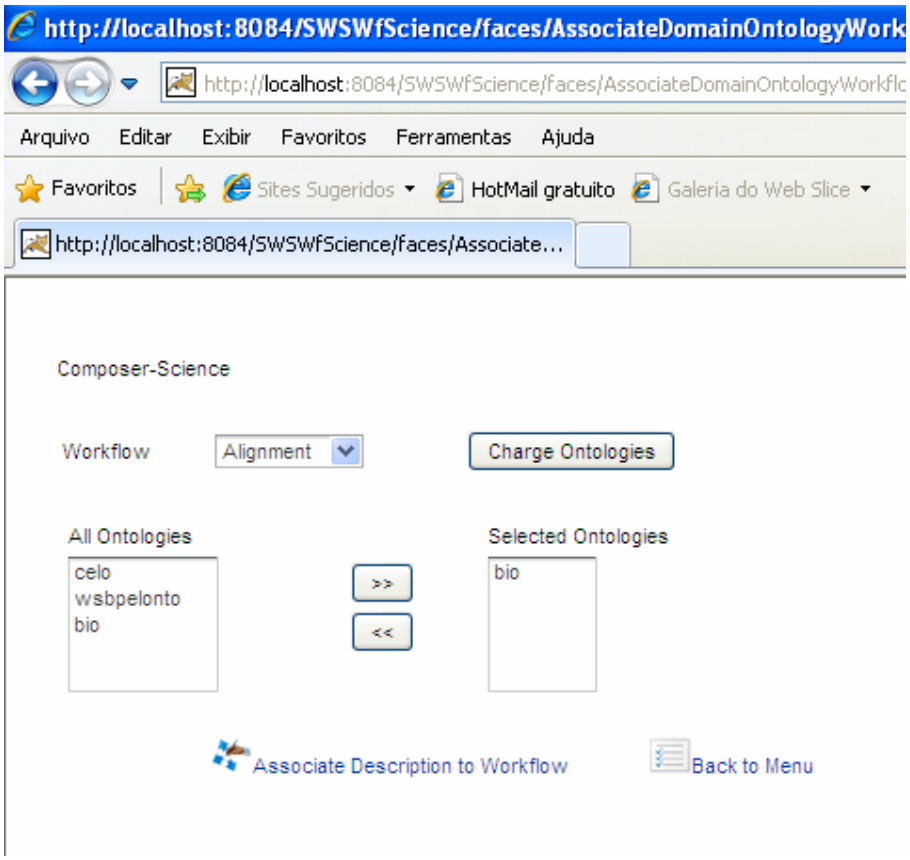


Figura 39 Associação de ontologia de domínio ao modelo do *workflow*.

A Figura 40 apresenta a descrição semântica do modelo de *workflow*. É possível criar uma ou mais descrições semânticas para o *workflow* como um todo. Essas descrições devem representar o objetivo global do *workflow*.

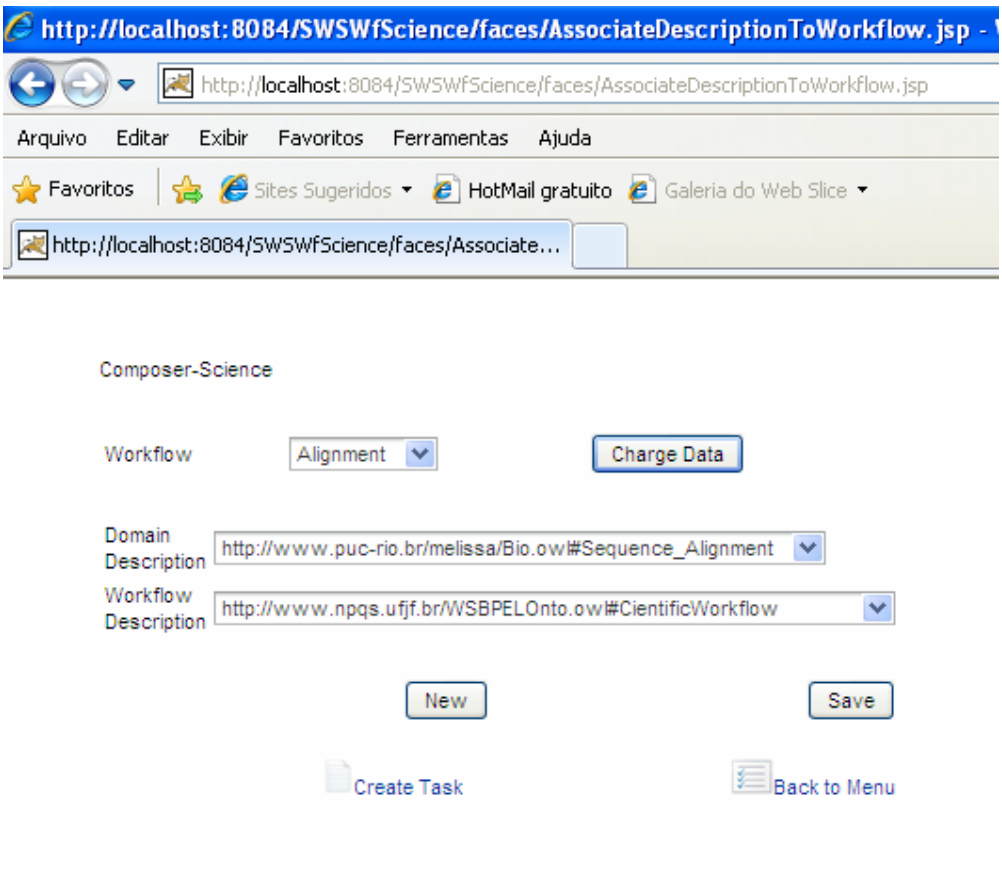


Figura 40 Descrição semântica global do *workflow*.

As figuras 41, 42, 43, 44, 45 e 46 apresentam a criação das tarefas que compõem o *workflow.*

As tarefas *GetSequence1*, *GetSequence2* e *GetSequence3* são paralelas entre si, recebem como parâmetros de entrada a indicação de qual banco de dados acessar e qual sequência recuperar, e realizam o acesso e a leitura de sequências em um banco de dados biológico. Essas tarefas retornam como saída as sequências lidas nos bancos de dados biológicos de origem.

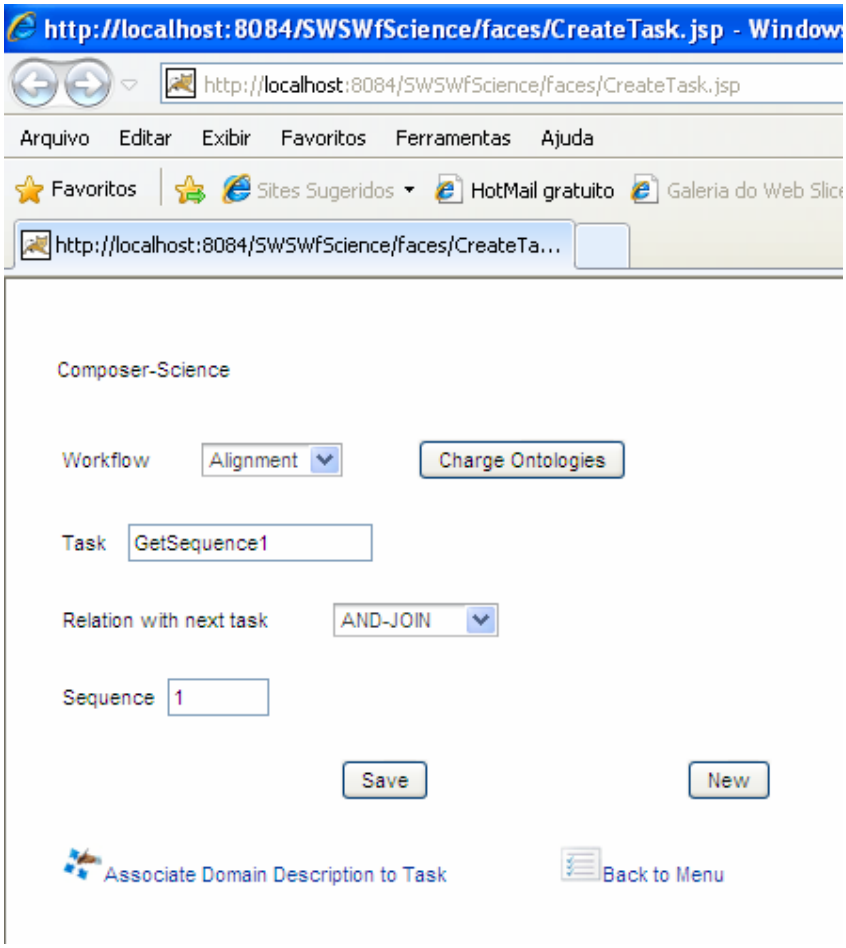


Figura 41 Criação da tarefa *GetSequence1*.

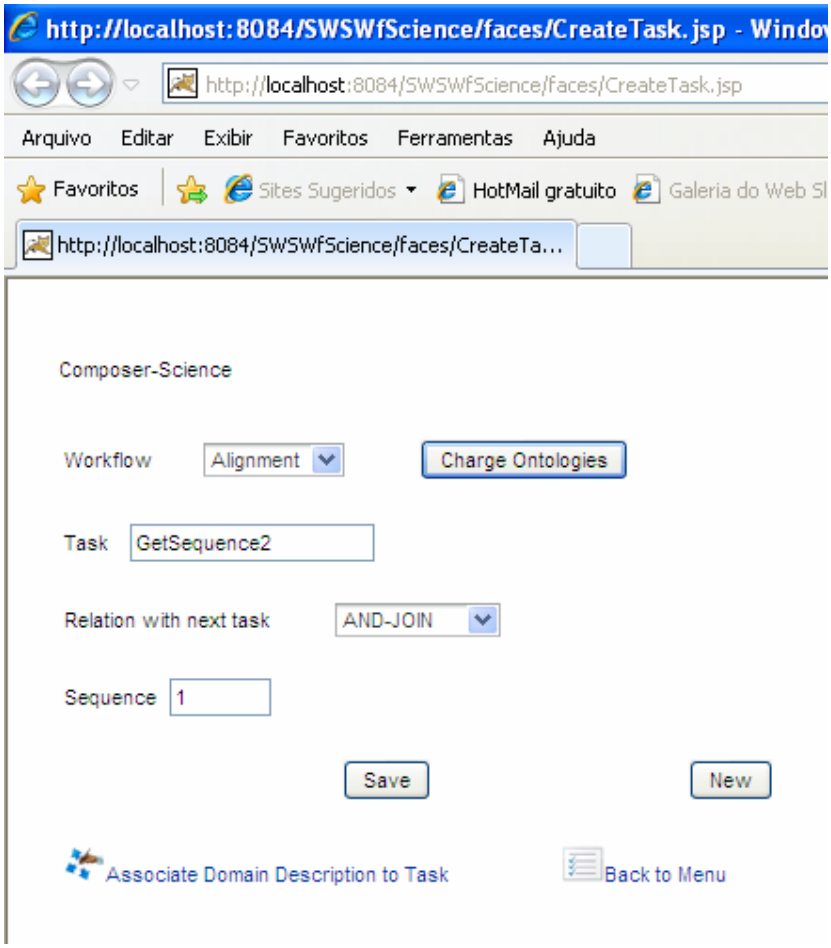


Figura 42 Criação da tarefa *GetSequence2*.

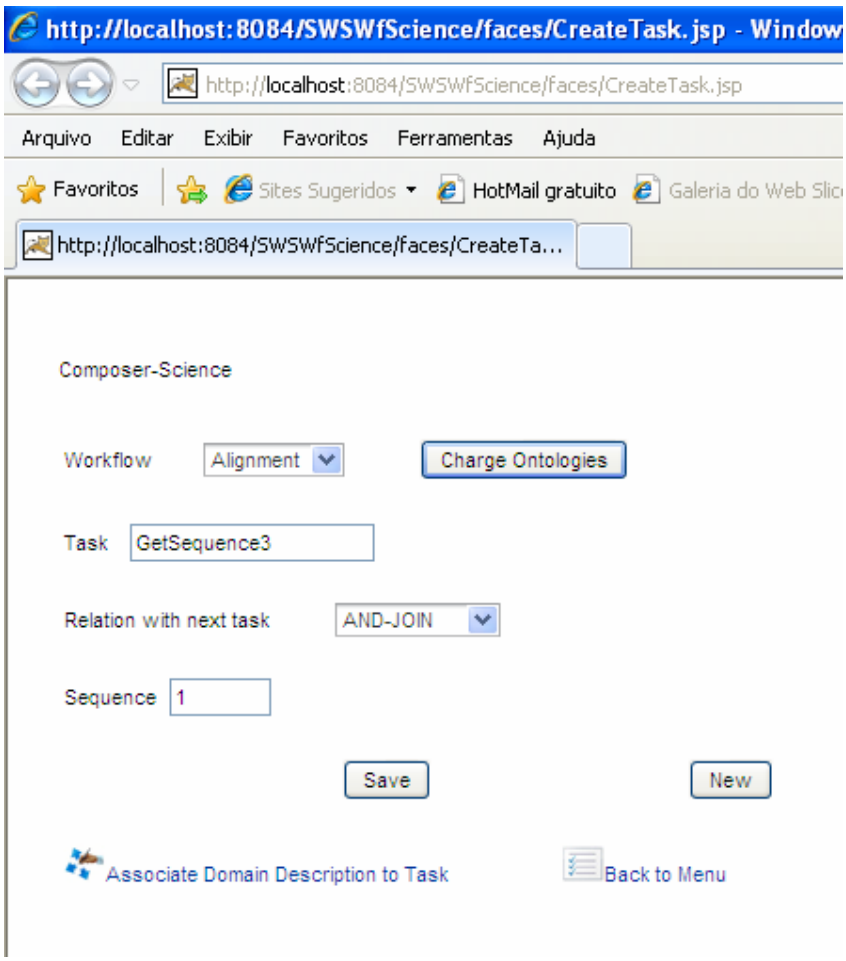


Figura 43 Criação da tarefa *GetSequence3*.

A tarefa *ReadAndFormatSequences* recebe como parâmetros de entrada as sequências recuperadas pelas tarefas paralelas anteriores, e transforma o formato das sequências em FASTA[5] [NCBI, 2010]. Essa tarefa produz como saída de sua execução as sequências formatadas.

[5] FASTA é um formato de apresentação de sequência de nucleotídeos [LEMOS, 2004].

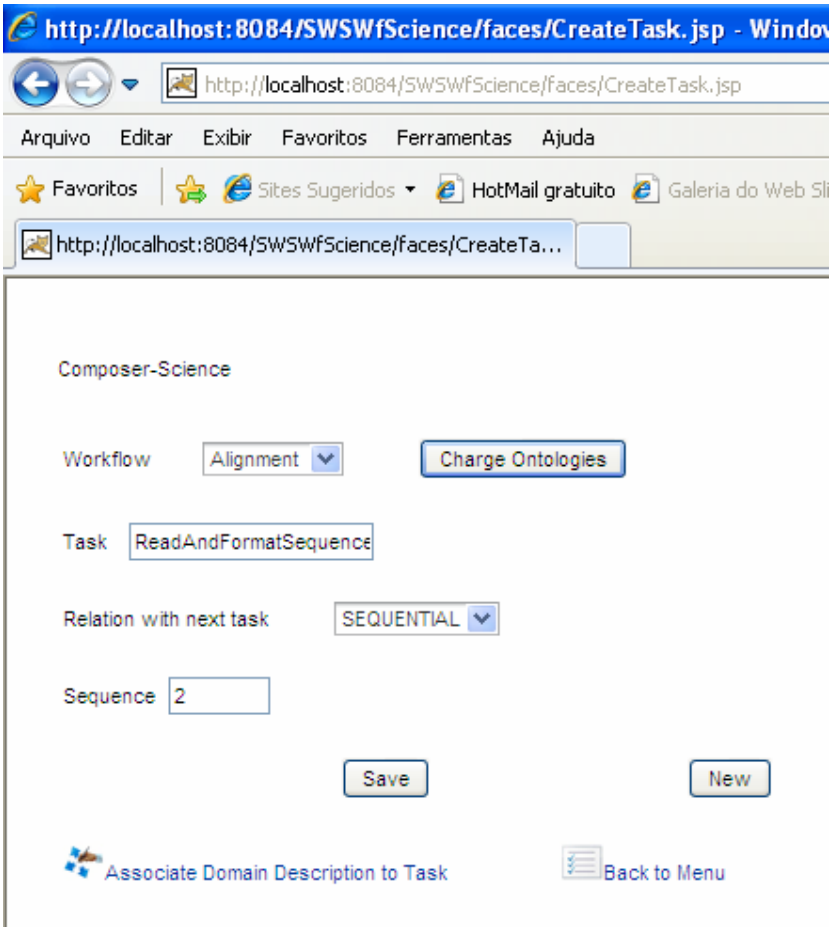


Figura 44 Criação da tarefa *ReadAndFormatSequences*.

A tarefa *Align* recebe como parâmetros de entrada as sequências formatadas, e realiza o alinhamento múltiplo dessas sequências, produzindo como resultado de sua execução os resultados do alinhamento em formato HTML ou XML.

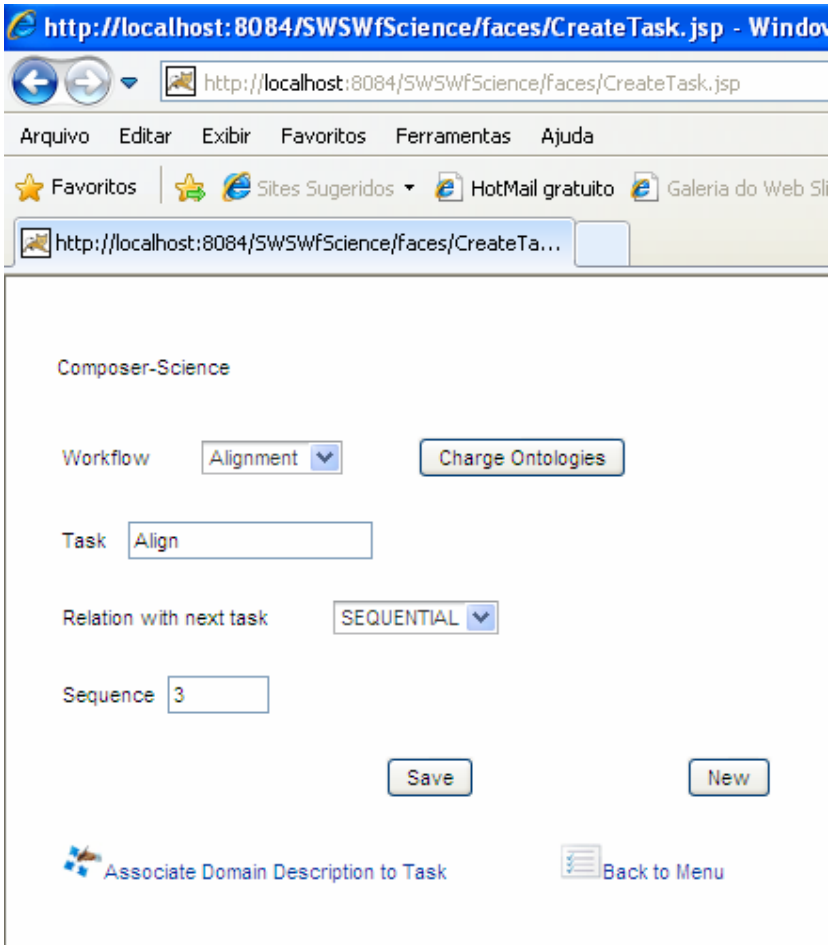


Figura 45 Criação da tarefa *Align*.

A tarefa *ResultsAndIDs* recebe como parâmetros de entrada os resultados do alinhamento, realiza sua formatação e exibe os resultados.

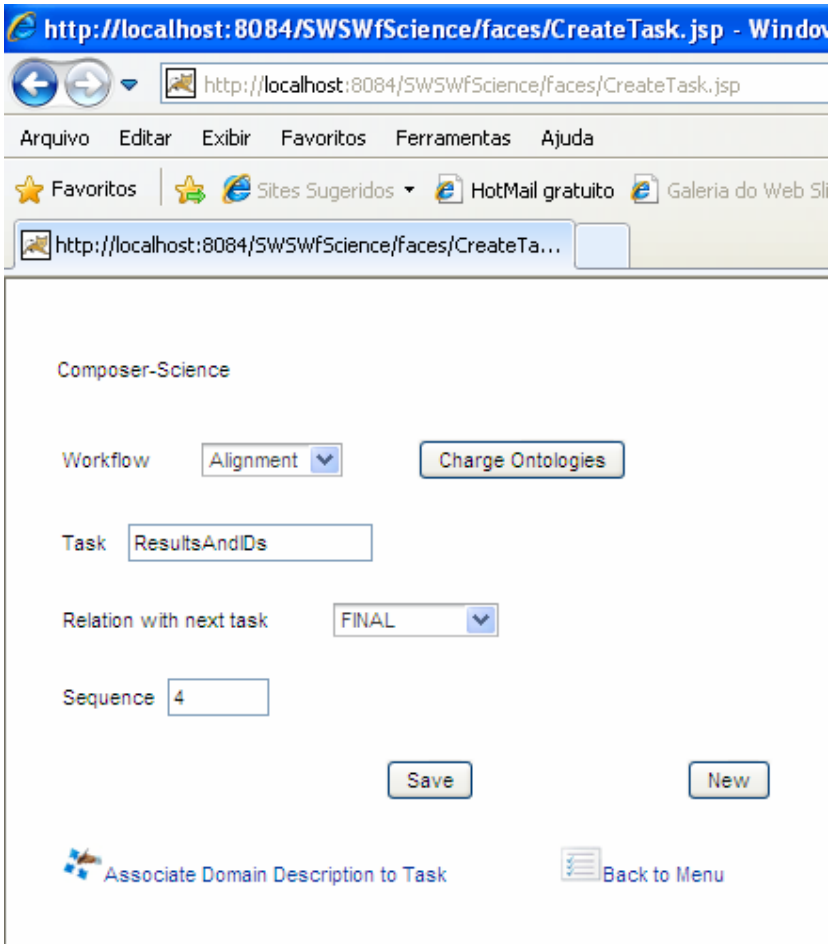


Figura 46 Criação da tarefa *ResultsAndIDs*.

Após a criação de cada tarefa do modelo do *workflow*, essas tarefas devem ser descritas semanticamente. Cada tarefa deve ter pelo menos uma descrição relacionada ao domínio de aplicação, acompanhada por uma descrição que defina a inserção dessa descrição no contexto do modelo WS-BPEL que será gerado para o *workflow*.

A Figura 47 apresenta a descrição semântica da tarefa *GetSequence1*. A tarefa *GetSequence1* é descrita, segundo o domínio de aplicação, com o termo *Read_Nucleotide_Sequence_Set*, que permite entender que a tarefa deve realizar a leitura de um conjunto de sequências de nucleotídeos. Segundo o contexto de WS-BPEL, a descrição *Read_Nucleotide_Sequence_Set* é associada à descrição *PartnerLink*, que permite entender que essa descrição semântica se relaciona à função que deve ser desempenhada pela tarefa. Em outras palavras, a partir dessa descrição semântica, podemos interpretar que o serviço Web que irá desempenhar essa tarefa deve ser capaz de ler uma sequência de nucleotídeos.

As tarefas *GetSequence2* e *GetSequence3* podem receber descrições semânticas iguais à descrição da tarefa *GetSequence1*, pois as três tarefas irão executar funções semelhantes, variando apenas os valores dos parâmetros de entrada para cada uma delas.

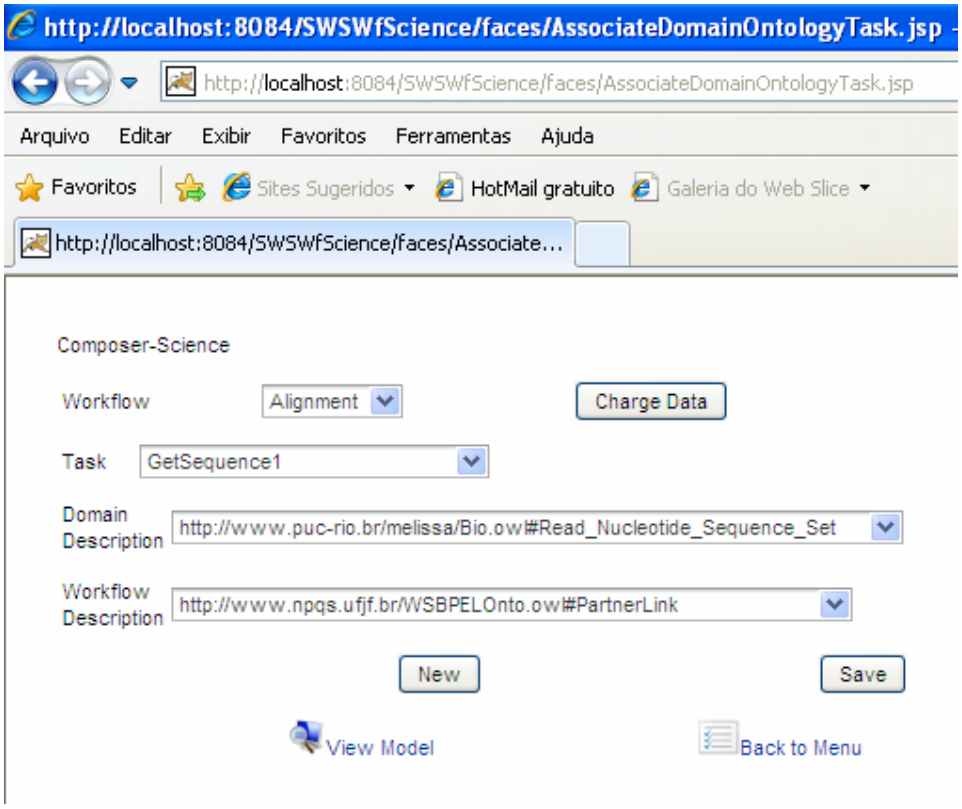


Figura 47 Descrição semântica da tarefa *GetSequence1*.

Além da descrição semântica apresentada na Figura 47, a tarefa *GetSequence1* pode receber a descrição apresentada na Figura 48, que indica que o parâmetro de saída (*output*) da tarefa deve ser um conjunto de sequências nucleotídicas. Os parâmetros de saída das tarefas *GetSequence2* e *GetSequence3* também podem ser descritos da mesma forma que o parâmetro de saída da tarefa *GetSequence1*.

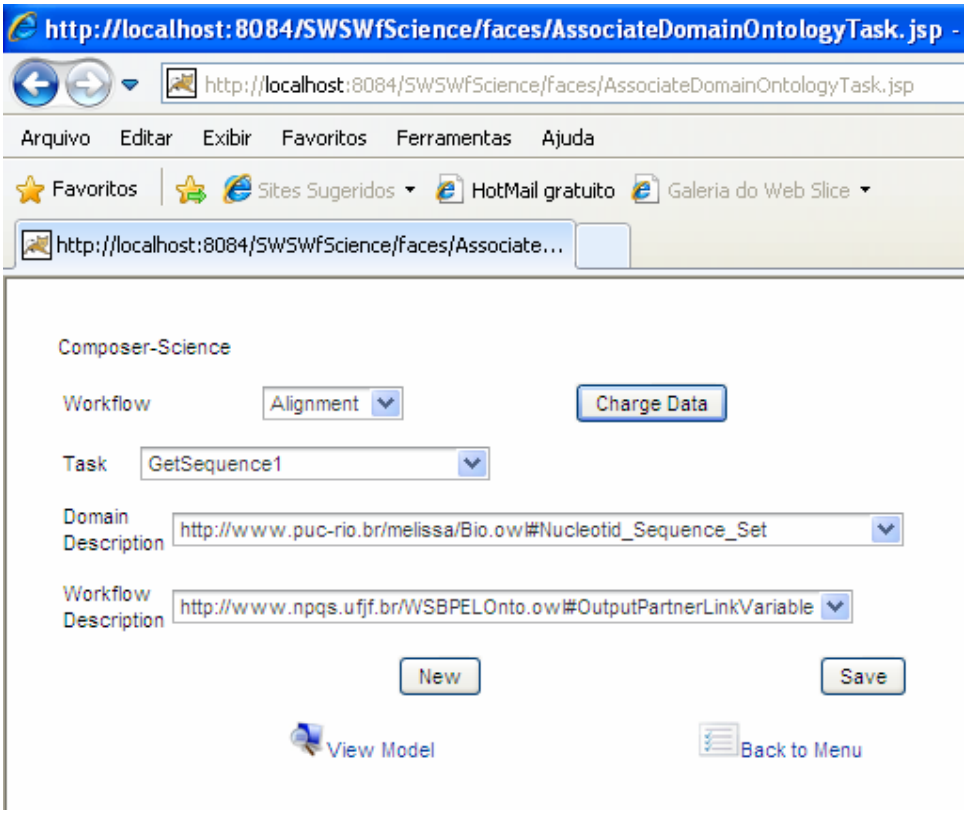


Figura 48 Descrição semântica do parâmetro de saída da tarefa *GetSequence1*.

A Figura 49 apresenta a descrição semântica do parâmetro de entrada da tarefa *ReadAndFormatSequences*. Segundo a descrição, o serviço Web que vai executar a tarefa deve ter como parâmetro de entrada um conjunto de sequências nucleotídicas.

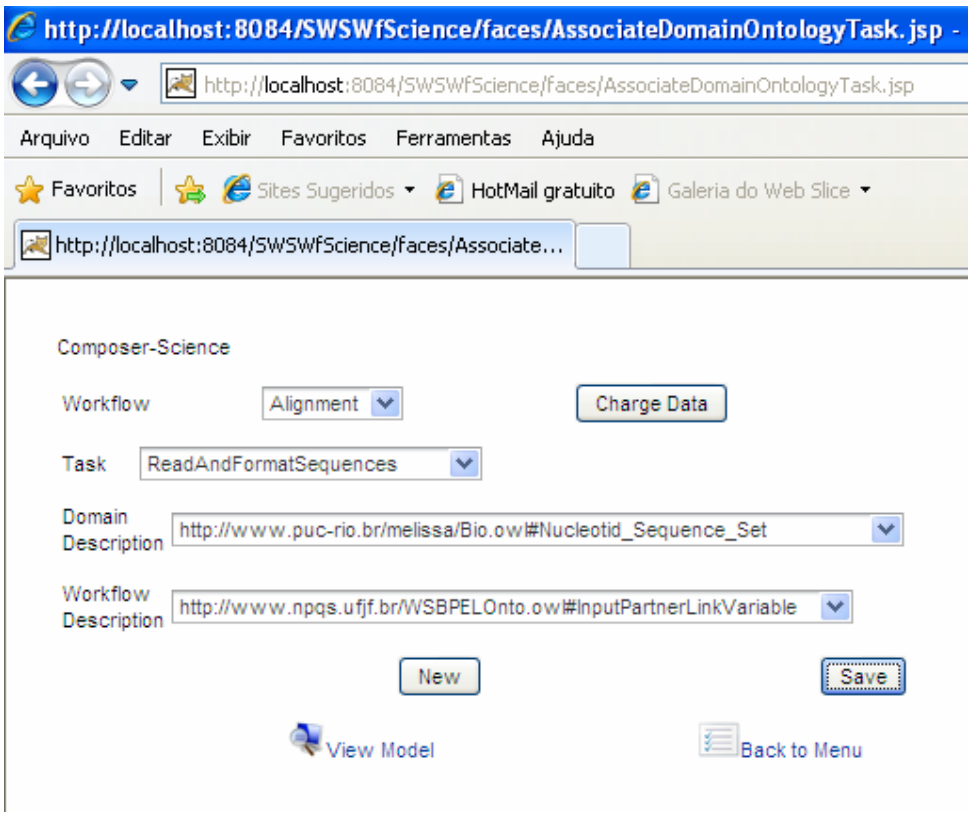


Figura 49 Descrição semântica do parâmetro de entrada da tarefa *ReadAndFormatSequences*.

A Figura 50 apresenta uma descrição semântica do parâmetro de saída da tarefa *ReadAndFormatSequences*. Essa tarefa deve retornar um conjunto de sequências nucleotídicas no formato FASTA.

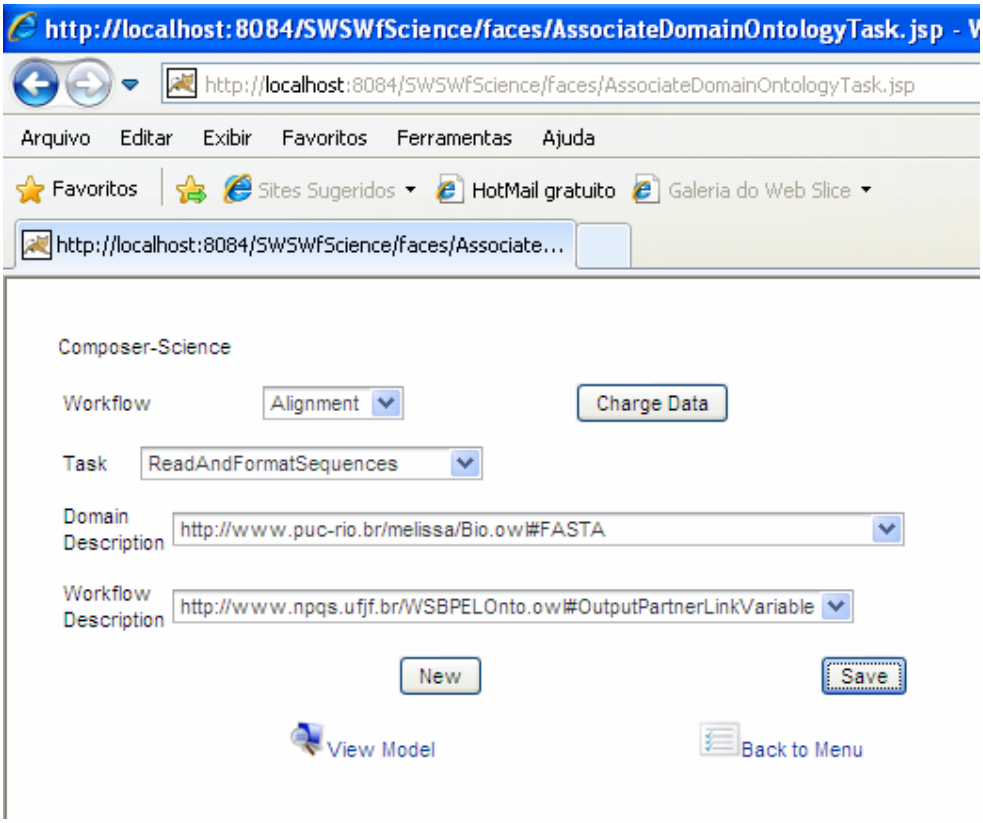


Figura 50 Descrição semântica do parâmetro de saída da tarefa *ReadAndFormatSequences*.

A Figura 51 apresenta a descrição semântica da função desempenhada pela tarefa *Align*. Segundo essa descrição, essa tarefa deve executar um alinhamento múltiplo.

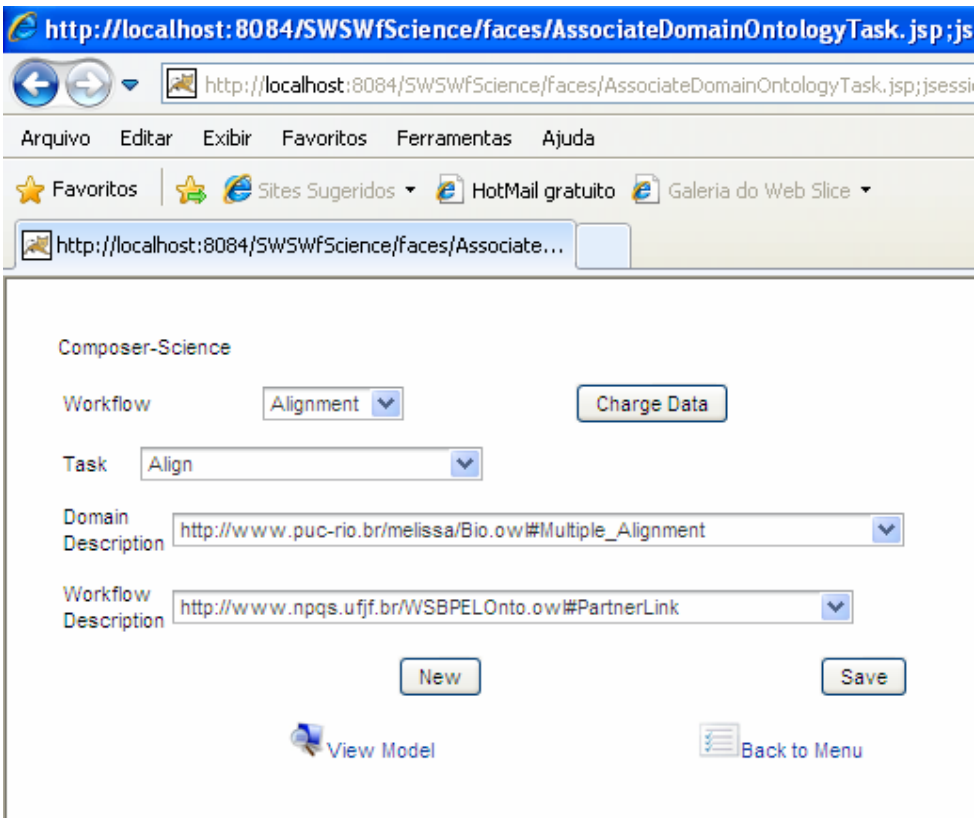


Figura 51 Descrição semântica da tarefa *Align.*

A Figura 52 apresenta mais uma característica relacionada à função desempenhada pela tarefa *Align*. A tarefa deve executar um alinhamento múltiplo utilizando o algoritmo BLASTP.

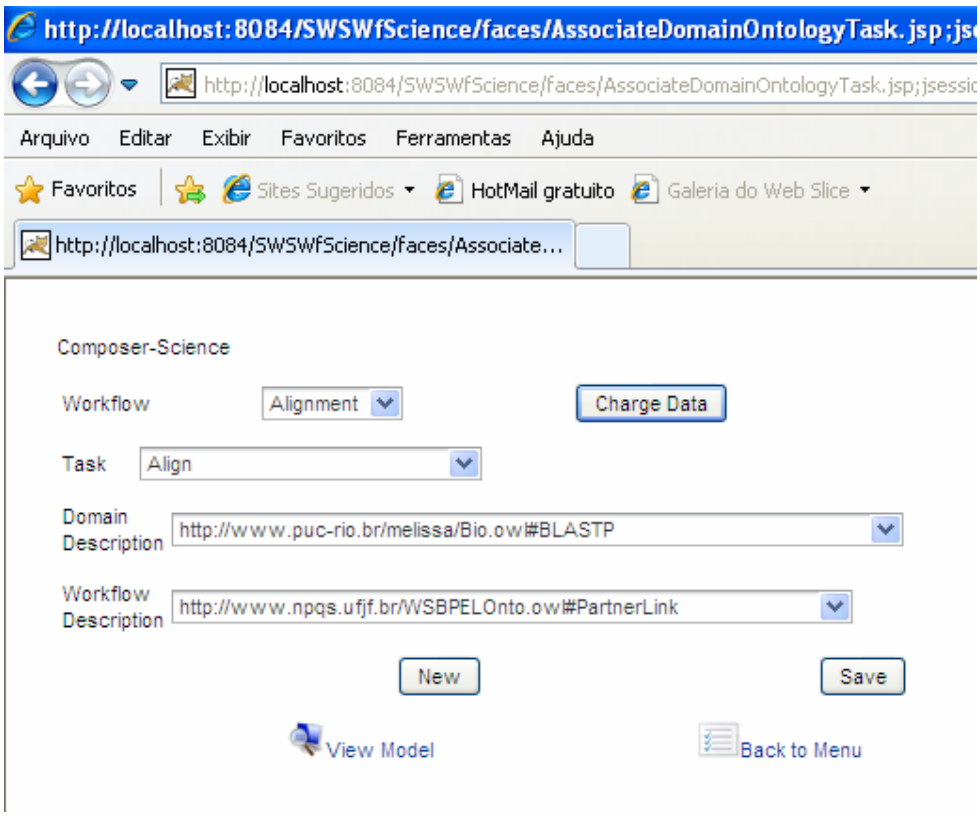


Figura 52 Outra descrição semântica da tarefa *Align*.

A Figura 53 apresenta a descrição semântica do parâmetro de entrada da tarefa *Align*. O parâmetro de entrada dessa tarefa deve estar no formato FASTA.

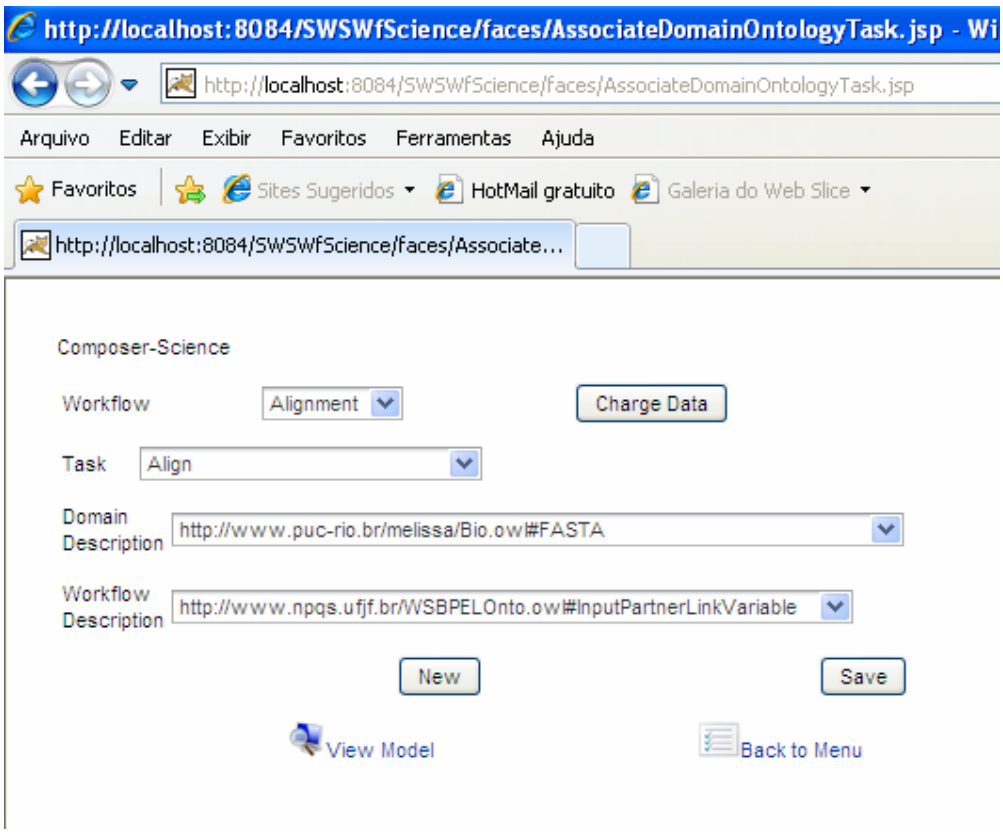


Figura 53 Descrição semântica do parâmetro de entrada da tarefa *Align*.

A Figura 54 apresenta a descrição semântica do parâmetro de saída da tarefa *Align*. O parâmetro de saída dessa tarefa deve estar no formato XML.

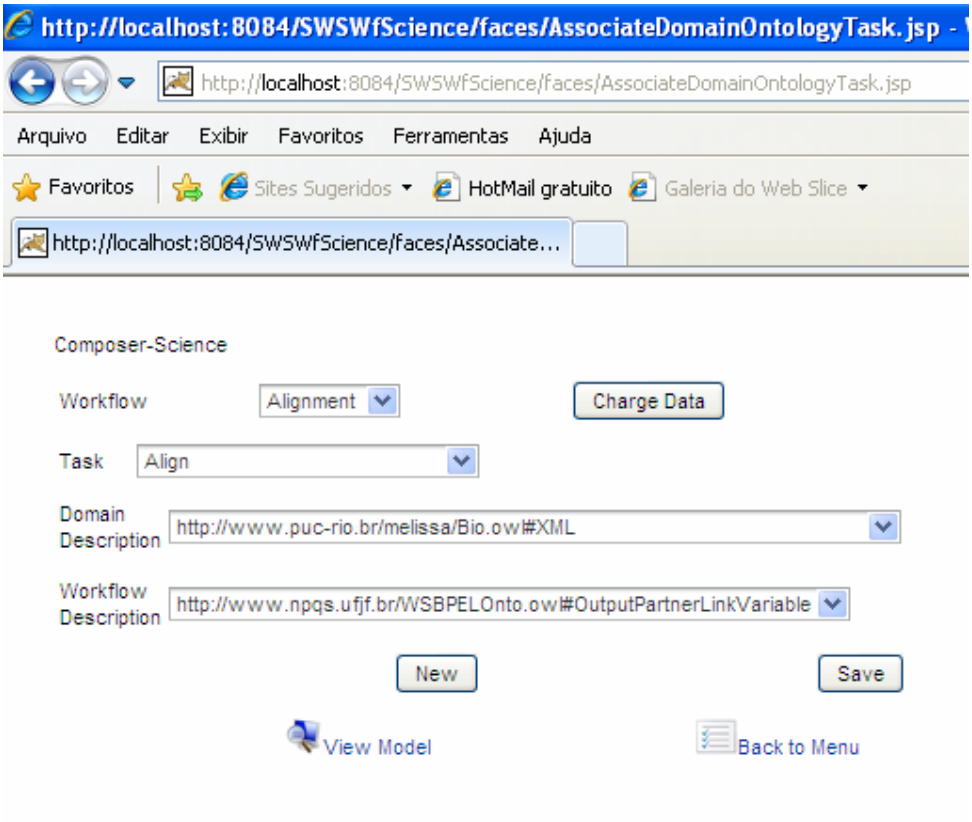


Figura 54 Descrição semântica do parâmetro de saída da tarefa *Align*.

A Figura 55 apresenta a descrição semântica do parâmetro de entrada da tarefa *AlignmentResults*. Essa tarefa deve receber dados no formato XML.

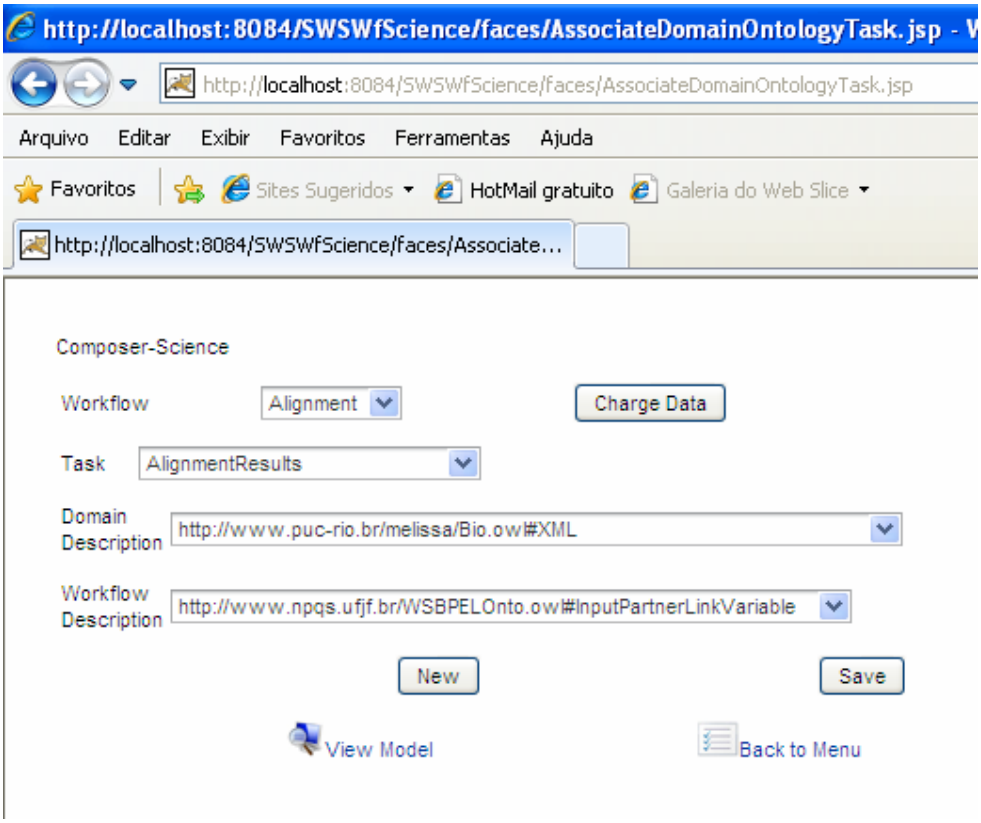


Figura 55 Descrição semântica do parâmetro de entrada da tarefa *AlignmentResults*.

A Figura 56 apresenta a descrição semântica do parâmetro de saída da tarefa *AlignmentResults*. Essa tarefa deve apresentar sua saída em formato HTML.

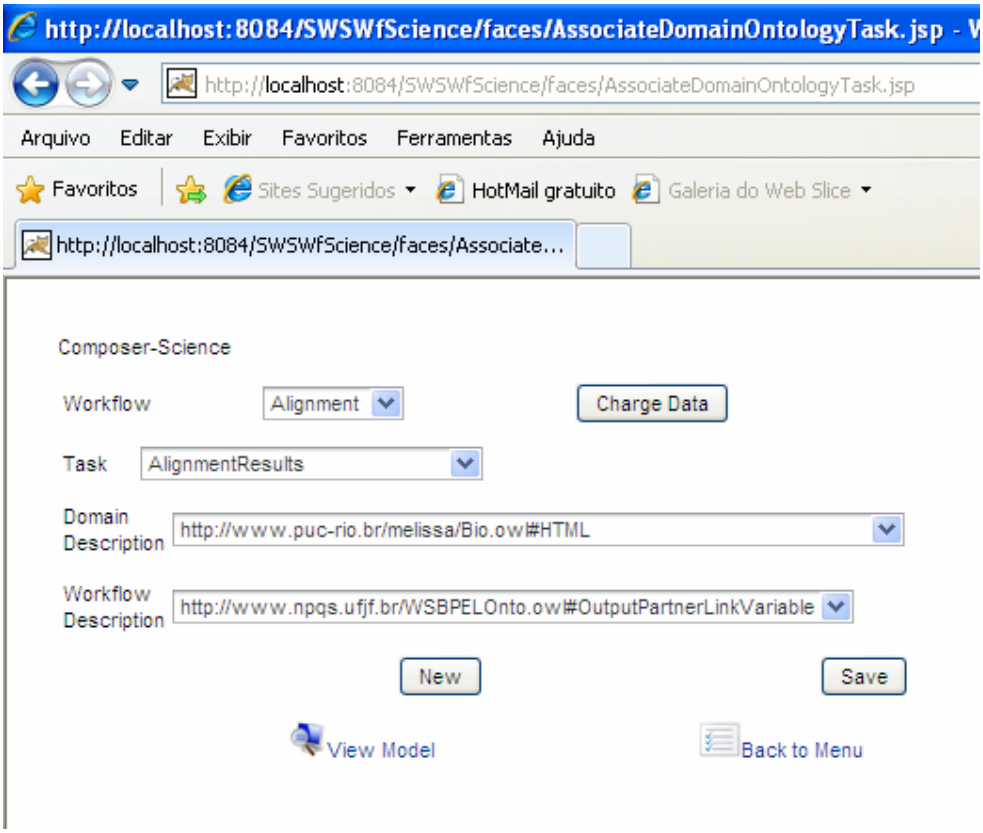


Figura 56 Descrição semântica do parâmetro de saída da tarefa *AlignmentResults*.

Para uma melhor compreensão do modelo criado, a Figura 57 apresenta a interface para visualização de um quadro resumo com o modelo completo criado pelo usuário do sistema, incluindo suas descrições semânticas.

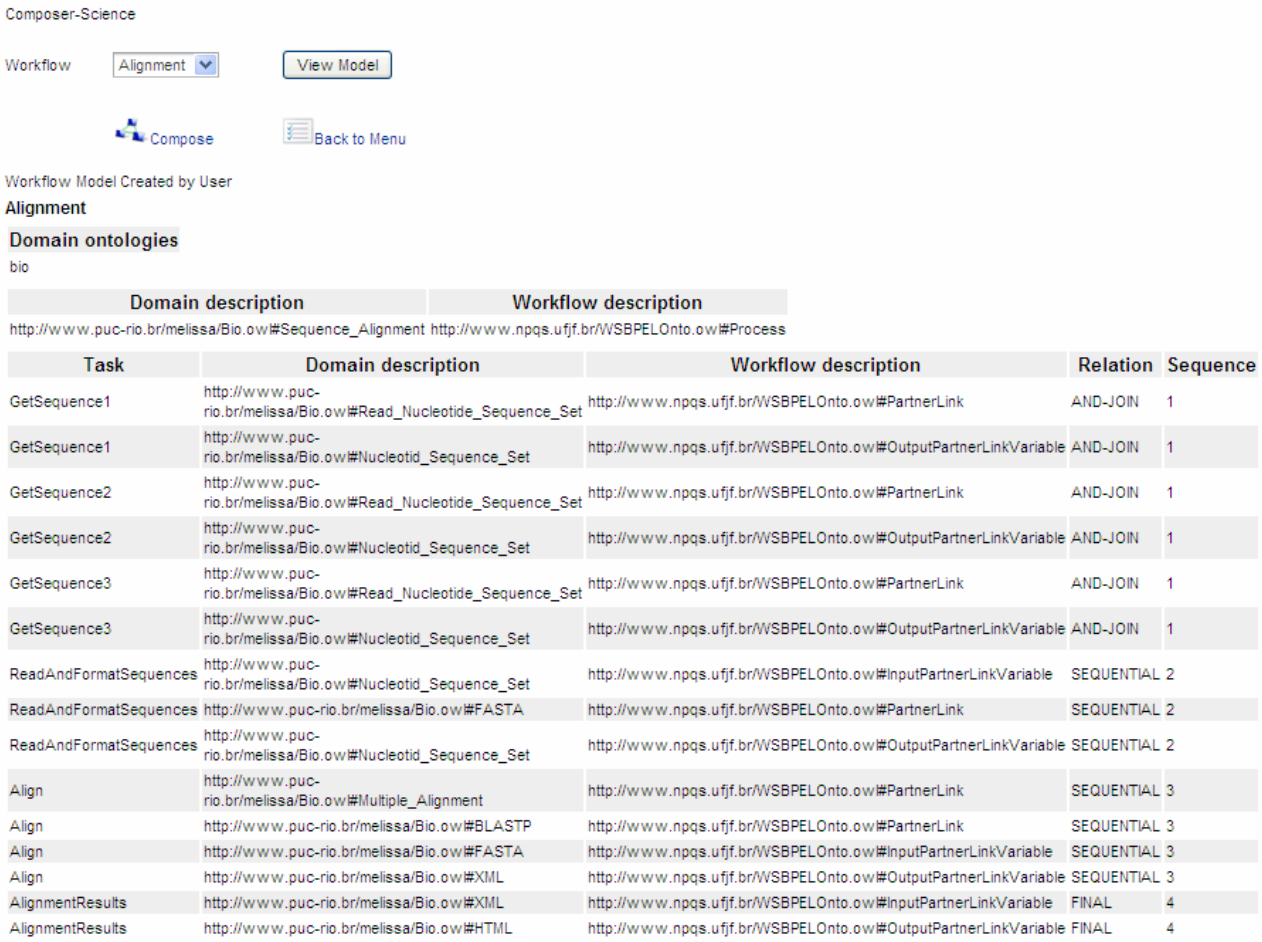
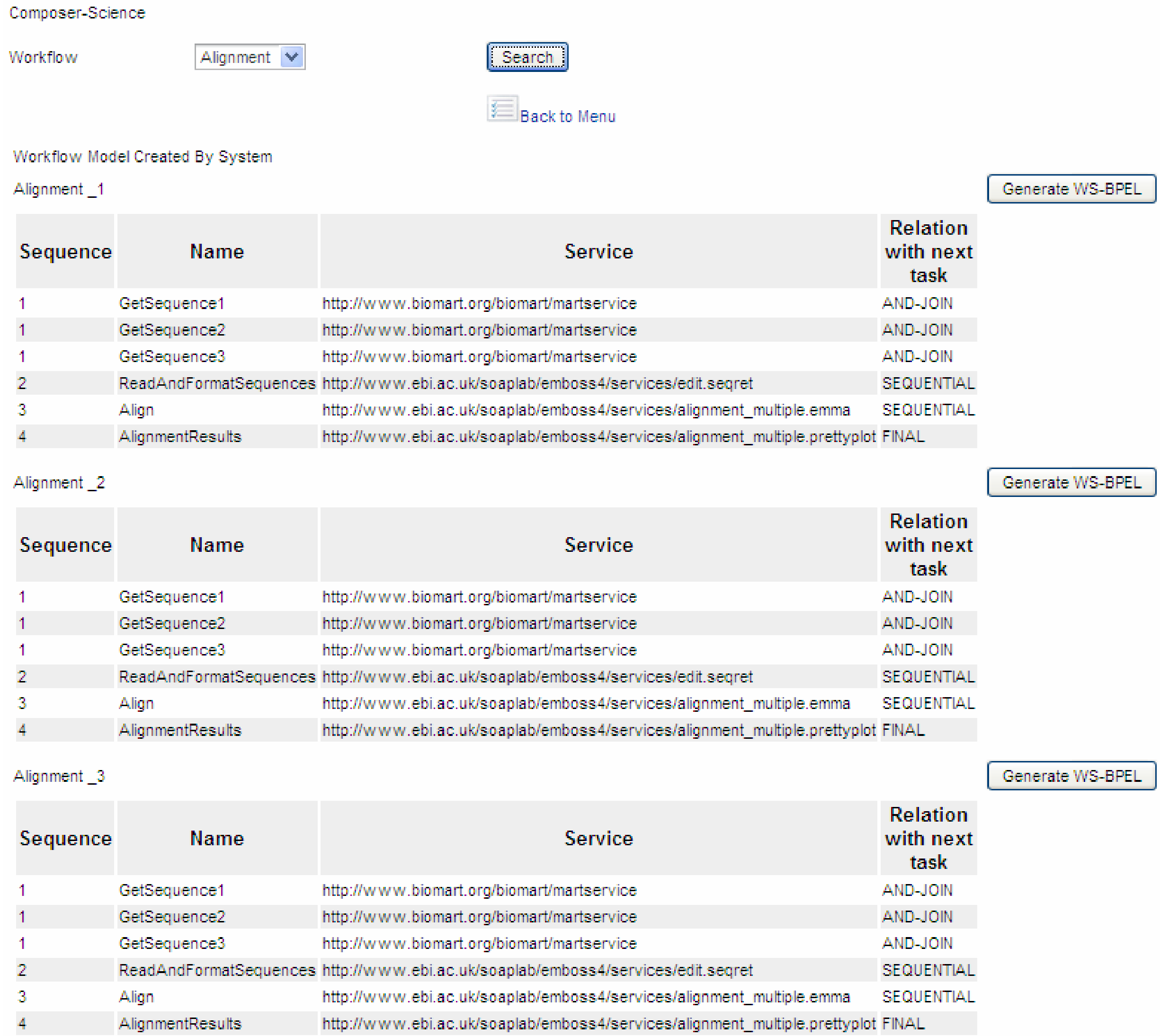


Figura 57 Interface para visualização do modelo criado pelo usuário.

Após verificar o modelo criado por ele, o usuário solicita a composição automática por parte do sistema. O usuário seleciona o modelo de *workflow* que definiu e solicita sua composição. O sistema realiza seu processamento (busca semântica (detalhada na seção 5.2), análise estrutural, composição) e apresenta para o usuário três composições semântica e estruturalmente possíveis, para que ele possa solicitar a geração do modelo WS-BPEL de uma delas. A Figura 58 apresenta a interface por meio da qual o usuário solicita o processamento e a geração do modelo WS-BPEL.



Composer-Science

Workflow Alignment Search

Back to Menu

Workflow Model Created By System

Alignment _1 Generate WS-BPEL

Sequence	Name	Service	Relation with next task
1	GetSequence1	http://www.biomart.org/biomart/martservice	AND-JOIN
1	GetSequence2	http://www.biomart.org/biomart/martservice	AND-JOIN
1	GetSequence3	http://www.biomart.org/biomart/martservice	AND-JOIN
2	ReadAndFormatSequences	http://www.ebi.ac.uk/soaplab/emboss4/services/edit.seqret	SEQUENTIAL
3	Align	http://www.ebi.ac.uk/soaplab/emboss4/services/alignment_multiple.emma	SEQUENTIAL
4	AlignmentResults	http://www.ebi.ac.uk/soaplab/emboss4/services/alignment_multiple.prettyplot	FINAL

Alignment _2 Generate WS-BPEL

Sequence	Name	Service	Relation with next task
1	GetSequence1	http://www.biomart.org/biomart/martservice	AND-JOIN
1	GetSequence2	http://www.biomart.org/biomart/martservice	AND-JOIN
1	GetSequence3	http://www.biomart.org/biomart/martservice	AND-JOIN
2	ReadAndFormatSequences	http://www.ebi.ac.uk/soaplab/emboss4/services/edit.seqret	SEQUENTIAL
3	Align	http://www.ebi.ac.uk/soaplab/emboss4/services/alignment_multiple.emma	SEQUENTIAL
4	AlignmentResults	http://www.ebi.ac.uk/soaplab/emboss4/services/alignment_multiple.prettyplot	FINAL

Alignment _3 Generate WS-BPEL

Sequence	Name	Service	Relation with next task
1	GetSequence1	http://www.biomart.org/biomart/martservice	AND-JOIN
1	GetSequence2	http://www.biomart.org/biomart/martservice	AND-JOIN
1	GetSequence3	http://www.biomart.org/biomart/martservice	AND-JOIN
2	ReadAndFormatSequences	http://www.ebi.ac.uk/soaplab/emboss4/services/edit.seqret	SEQUENTIAL
3	Align	http://www.ebi.ac.uk/soaplab/emboss4/services/alignment_multiple.emma	SEQUENTIAL
4	AlignmentResults	http://www.ebi.ac.uk/soaplab/emboss4/services/alignment_multiple.prettyplot	FINAL

Figura 58 Interface de composição e geração de WS-BPEL.

5.2 BUSCA E COMPOSIÇÃO

A partir das descrições semânticas das tarefas do *workflow* são realizadas inferências semânticas na ontologia de domínio associada ao modelo, segundo a qual as descrições das tarefas foram feitas. Essas inferências têm como objetivo encontrar termos similares aos utilizados para descrever as tarefas, os quais serão utilizados na busca por serviços Web semânticos compatíveis com as tarefas descritas. As inferências aumentam o poder de busca, ampliando as possibilidades de descoberta de serviços.

As inferências semânticas são realizadas com o uso do *framework* SOR, que permite sua realização a partir do arquivo OWL da ontologia de domínio. A Figura 59 apresenta um trecho de código utilizado para a realização de inferências a partir das descrições das tarefas do *workflow*. As descrições de domínio e relacionadas ao modelo WS-BPEL correspondentes de cada tarefa são listadas em *descs*. As ontologias de domínio associadas ao modelo do *workflow* são inseridas na lista *wfOntos*. A lista *searchDescs* é criada para conter todos os termos obtidos nas inferências. São realizadas inferências da descrição WS-BPEL relacionada a cada descrição de domínio para obter sua superclasse (*aClass*), que será utilizada nas buscas de serviços semânticos juntamente com cada descrição de domínio e os resultados das inferências realizadas a partir delas. São realizadas inferências de todas as descrições de domínio da tarefa em todas as ontologias de domínio associadas. Os termos obtidos nas inferências a partir das descrições de domínio e a superclasse da descrição WS-BPEL correspondente são inseridos na lista *searchDescs*, que posteriormente será utilizada para realizar as buscas por serviços compatíveis com as descrições da tarefa e os termos relacionados a elas, obtidos nas inferências.

```
Map<String, String> descs =
    swswfsciencecontroller.listTaskDescriptions(taskId);
List<Ontology> wfOntos =
    swswfsciencecontroller.listWfOntologies(wfId);
Map<String, String> searchDescs = new HashMap<String, String>();
Set set = descs.keySet();
for (Iterator iter = descs.iterator(); iter.hasNext();) {
    String domainDesc = (String)iter.next();
    String wfDesc = descs.get(domainDesc);
    String[] aux = domainDesc.split("#");
    domainDesc = aux[1];
    aux = wfDesc.split("#");
    wfDesc = aux[1];
    String aClass = swswfsciencecontroller.
        listAncestorClassWSBPELOnto(wfDesc);
    for(Iterator ontoIt = wfOntos.iterator(); ontoIt.hasNext();){
        Ontology o = (Ontology) ontoIt.next();
        String ontoName = o.getName().trim();
        Tupla[] idd = swswfsciencecontroller.
            inferenciaDescendantClassesController("C:/"+
            "Documents and Settings/NetBeansProjects/SWSWfScience/"+
            "src/java/files/owl/"+ontoName+".owl", domainDesc);
        for(int j=0; j<idd.length-1; j++){
            Tupla tdd = idd[j];
            String dd = tdd.getUrl();
            String[] dds = dd.split("#");
            dd = dds[1];
            searchDescs.put(dd, aClass);
        }
    }
}
```

Figura 59 Código de inferência semântica.

Com os termos utilizados inicialmente na descrição das tarefas e os termos obtidos por meio de inferências, é realizada a varredura dos repositórios em busca dos serviços Web semânticos que contenham em sua descrição semântica termos similares aos utilizados para descrever as tarefas. A Figura 60 apresenta o código que percorre a lista com as descrições de uma tarefa e suas inferências (*searchDescs*), e realiza a busca de serviços compatíveis (função *searchService*).

```
for(Iterator it = searchDescs.iterator(); it.hasNext();) {
    String domainDesc = (String)it.next();
    List<String> services =
        swswfsciencecontroller.searchService(wfId, taskId, domainDesc);
}
```

Figura 60 Busca de serviços compatíveis com cada descrição semântica de uma tarefa.

A Figura 61 apresenta o código que realiza as comparações das descrições das tarefas com as descrições dos serviços semânticos disponíveis no repositório do *framework*. O repositório é percorrido, as descrições dos serviços semânticos são extraídas dos respectivos arquivos OWL-S por meio da OWL-S API, e comparadas às descrições da tarefa. Os serviços compatíveis com a tarefa são inseridos na lista *urls*, que é retornada como resultado da execução da função de busca.

```
public List<String> searchServices(int wfId, int taskId,
        Map<String, String> descs) throws FileNotFoundException{
    List<String> urls = new ArrayList();
    String dir = "C:/xampp/htdocs/owls";
    File diretorio = new File(dir);
    String[] arquivos = diretorio.list();
    for (int i=0; i<arquivos.length; i++ ){
         String string = arquivos[i];
         String owlsFile = "";
         File diretorioRaiz =
                 new File(diretorio.getAbsolutePath() + "\\" + string);
         if ( diretorioRaiz.isFile() ){
             owlsFile = diretorioRaiz.getName();
             owlsFile = "http://localhost/owls/"+owlsFile.trim();
         }
         //extract information from owl-s using owl-s api
         org.mindswap.owl.OWLKnowledgeBase kb =
                 org.mindswap.owl.OWLFactory.createKB();
         kb.setReasoner("Pellet");
         org.mindswap.owls.service.Service s =
                 (org.mindswap.owls.service.Service)
                 kb.readService(URI.create(owlsFile));
         String param = "N";
         String in = "N";
         String ou = "N";
         //profile
         org.mindswap.owls.profile.Profile p = s.getProfile();
         //service
         String serviceName = p.getServiceName();
         //loop - task descriptions
         Set set = descs.keySet();
         for(Iterator it = set.iterator(); it.hasNext();) {
             String domainDesc = (String)it.next();
             String wfDesc = descs.get(domainDesc);
             //Service Parameters
             org.mindswap.owl.OWLIndividualList oil =
                     p.getServiceParameters();
             for (Iterator itP = oil.iterator(); itP.hasNext();) {
                org.mindswap.owl.OWLIndividual oi =
                        (org.mindswap.owl.OWLIndividual) itP.next();
                String serviceParam = oi.getLocalName();
                if(wfDesc.toLowerCase().equals("partnerlink")){
                    if(serviceParam.toLowerCase().
                            equals(domainDesc.toLowerCase())){
                        param = "Y";
                    }
                }
             }
             //Inputs
             org.mindswap.owls.process.InputList il = p.getInputs();
             for (Iterator itI = il.iterator(); itI.hasNext();) {
                org.mindswap.owls.process.Input ip =
                        (org.mindswap.owls.process.Input)itI.next();
                org.mindswap.owl.OWLType t = ip.getParamType();
                if(wfDesc.toLowerCase().
                        equals("inputpartnerlinkvariable")){
                    if(ip.getLocalName().toLowerCase().
                            equals(domainDesc.toLowerCase())){
                        in = "Y";
                    }
                }
             }
             //Outputs
             org.mindswap.owls.process.OutputList ol = p.getOutputs();
             for (Iterator itO = ol.iterator(); itO.hasNext();) {
                org.mindswap.owls.process.Output o =
                        (org.mindswap.owls.process.Output)itO.next();
                org.mindswap.owl.OWLType t = o.getParamType();
                if(wfDesc.toLowerCase().
                        equals("outputpartnerlinkvariable")){
                    if(o.getLocalName().toLowerCase().
                            equals(domainDesc.toLowerCase())){
                        ou = "Y";
                    }
                }
             }
         }
         if(param.equals("Y") && in.equals("Y") && ou.equals("Y")){
             int test = urls.indexOf(serviceName);
             if(test < 0){
                urls.add(serviceName);
             }
         }
    }
    return urls;
}
```

Figura 61 Busca de serviços semânticos compatíveis com as descrições de uma tarefa.

Os serviços compatíveis encontrados são registrados no banco de dados, criando os modelos semanticamente possíveis, ou seja, modelos executáveis do *workflow*, formados por serviços Web semânticos, cujas descrições semânticas atendam aos parâmetros

fornecidos pelo usuário. Os modelos semanticamente possíveis têm seus dados inseridos nas tabelas *workflowexe e taskexe* do banco de dados para armazenamento de modelos, que foi detalhado na seção 4.5 do capítulo 4 deste trabalho.

A partir dos modelos semanticamente possíveis, os serviços têm seus dados de entrada e saída analisados e, dessa forma, é definido se os modelos obtidos inicialmente são modelos semântica e estruturalmente possíveis.

Para a realização da análise estrutural, foi necessário criar classes para representar os modelos semanticamente possíveis. Essas classes contêm informações relevantes para a realização dessa análise relacionadas aos modelos e suas tarefas. A Figura 62 apresenta o modelo de classes que apresenta essa estrutura. Os modelos semanticamente possíveis são armazenados no banco de dados e, para a realização da análise estrutural, os dados de cada um desses modelos são selecionados e inseridos em um objeto *WorkflowExe*. O *WorkflowExe* contém uma lista de tarefas *TaskExe*. Uma *taskExe* possui um serviço da classe *Service* que a executa. O *Service* possui uma lista de parâmetros de entrada do tipo *Input* e uma lista de parâmetros de saída do tipo *Output*. Cada elemento *Input* apresenta uma descrição semântica (*inputSemanticDescription*) e seu tipo de dado (*inputType*). Cada elemento *Output* apresenta uma descrição semântica (*outputSemanticDescription*) e seu tipo de dado (*outputType*).

A função que executa a análise estrutural dos serviços de um modelo recebe como parâmetro um objeto do tipo *WorkflowExe* e, extrai desse objeto os tipos dos parâmetros de entrada e saída de cada serviço de cada tarefa, a relação da tarefa com a próxima tarefa, e realiza a análise, definindo o modelo como semântica e estruturalmente possível (*possible = "Y"*) ou não (*possible = "N"*).

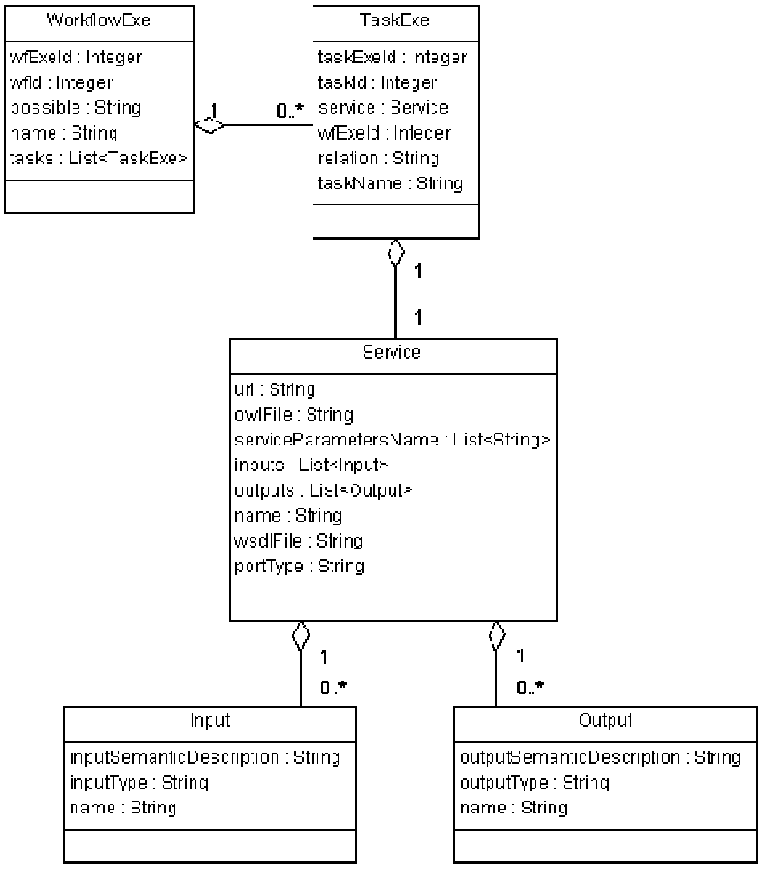


Figura 62 Classes para representação de modelos executáveis de *workflow*.

A Figura 63 apresenta o código que realiza a análise da compatibilidade estrutural entre os serviços de um modelo de *workflow* executável semanticamente possível, e define se o modelo é também estruturalmente possível. A função que realiza a análise recebe como parâmetro um objeto que representa um *workflow* semanticamente possível. Esse objeto apresenta a estrutura do modelo de classes da Figura 62.

São consideradas as seis possibilidades de relações de uma tarefa com a sua seguinte no modelo do *workfow*. Se a relação da tarefa com a próxima é *SEQUENTIAL*, a quantidade e os tipos de seus parâmetros de saída devem ser compatíveis com a quantidade e os tipos dos parâmetros de entrada da tarefa seguinte. Se a relação da tarefa com a próxima é *AND-SPLIT*, a quantidade dos seus parâmetros de saída deve ser a mesma quantidade dos parâmetros de todas as tarefas, que apresentem número seqüencial imediatamente seguinte ao seu, somados. Além disso, os tipos dos parâmetros de saída da tarefa devem ser compatíveis com os tipos dos parâmetros de entrada das tarefas que serão executadas em seguida. Se a relação da tarefa com a seguinte é *AND-JOIN*, os parâmetros de saída de todas as tarefas executadas em paralelo com ela somados devem

ser, em quantidade e tipos, compatíveis com os parâmetros de entrada da tarefa seguinte. Se a relação da tarefa com a seguinte for *OR-JOIN* ou *OR-SPLIT*, a quantidade e os tipos de seus parâmetros de saída devem ser compatíveis com a quantidade e os tipos dos parâmetros de entrada da tarefa seguinte (com número sequencial seguinte). Se a relação da tarefa é *FINAL*, seus parâmetros de entrada já foram analisados a partir da tarefa anterior a ela e, não é necessário analisar seus parâmetros de saída, pois não há tarefa seguinte para que a compatibilidade seja analisada.

```
public String structuralAnalisys(WorkflowExe wfExe){
    String possible = "Y";
    //wf tasks
    List<TaskExe> tasks = wfExe.getTasks();
    for(int i=0; i<tasks.size(); i++){
        if(possible.equals("Y")){
            TaskExe task = tasks.get(i);
            //relation
            String relation = task.getRelation();
            //service
            Service service = task.getService();
            //sequence
            int sequence = task.getSequence();
            //service inputs
            List<entity.Input> inputs = service.getInputs();
            //service outputs
            List<entity.Output> outputs = service.getOutputs();
            //relation with next task
            if(relation.toUpperCase().equals("SEQUENTIAL")){
                TaskExe nextTask = tasks.get(i+1);
                Service serviceNextTask = nextTask.getService();
                List<entity.Input> nextTaskInputs =
                        serviceNextTask.getInputs();
                if(outputs.size() == nextTaskInputs.size()){
                    for(int k=0; k<outputs.size(); k++){
                        entity.Output out = outputs.get(k);
                        String outT = out.getOutputType();
                        entity.Input in = nextTaskInputs.get(k);
                        String inT = in.getInputType();
                        if(outT.equals(inT)){
                            possible = "Y";
                        }else{
                            possible = "N";
                        }
                    }
                }else{
                    possible = "N";
                }
            }else if(relation.toUpperCase().equals("AND-SPLIT")){
                List<entity.Input> nextInputs = new ArrayList();
                int j=0;
                while(tasks.get(j).getSequence() == (sequence+1)){
                    Service s = tasks.get(j).getService();
                    List<entity.Input> in = s.getInputs();
                    for(Iterator it = in.iterator(); it.hasNext();){
                        nextInputs.add((entity.Input) it.next());
                    }
                    j++;
                }
                if(outputs.size() == nextInputs.size()){
                    for(int k=0; k<outputs.size(); k++){
                        entity.Output out = outputs.get(k);
                        String outT = out.getOutputType();
                        entity.Input in = nextInputs.get(k);
                        String inT = in.getInputType();
                        if(outT.equals(inT)){
                            possible = "Y";
                        }else{
                            possible = "N";
                        }
                    }
                }else{
                    possible = "N";
                }
            }else if(relation.toUpperCase().equals("AND-JOIN")){
                List<entity.Output> plusOutputs = new ArrayList();
                int j=0;
                while(tasks.get(j).getSequence() == sequence){
                    Service s = tasks.get(j).getService();
                    List<entity.Output> out = s.getOutputs();
                    for(Iterator it = out.iterator(); it.hasNext();){
                        plusOutputs.add((entity.Output) it.next());
                    }
                    j++;
                }
                List<entity.Input> nextInputs = new ArrayList();
                for(int k=0; k<tasks.size(); k++){
                    if(tasks.get(k).getSequence() == (sequence+1)){
                        Service s = tasks.get(k).getService();
                        nextInputs = s.getInputs();

                    }
                }
                if(plusOutputs.size() == nextInputs.size()){
                    for(int k=0; k<plusOutputs.size(); k++){
                        entity.Output out = plusOutputs.get(k);
                        String outT = out.getOutputType();
                        entity.Input in = nextInputs.get(k);
                        String inT = in.getInputType();
                        if(outT.equals(inT)){
                            possible = "Y";
                        }else{
                            possible = "N";
                        }
                    }
                }else{
                    possible = "N";
                }
            }else if(relation.toUpperCase().equals("OR-SPLIT") ||
                    relation.toUpperCase().equals("OR-JOIN")){
                for(int j=0; j<tasks.size(); j++){
                    if(tasks.get(j).getSequence() == sequence+1){
                        Service s = tasks.get(j).getService();
                        List<entity.Input> in = s.getInputs();
                        if(outputs.size() == in.size()){
                            for(int k=0; k<outputs.size(); k++){
                                if(outputs.get(k).getOutputType().
                                        equals(in.get(k).getInputType())){
                                    possible = "Y";
                                }else{
                                    possible = "N";
                                }
                            }
                        }else{
                            possible = "N";
                        }
                    }
                }
            }else if(relation.toUpperCase().equals("FINAL")){
                //no analisys
            }
        }
    }
    return possible;
}
```

Figura 63 Análise estrutural de modelo semanticamente possível.

No protótipo criado, apenas três modelos semântica e estruturalmente possíveis são apresentados ao usuário que, então, pode solicitar a geração de um documento WS-BPEL que o represente. O número reduzido de modelos apresentado ao usuário pelo protótipo se deve ao fato de que o protótipo tem como objetivo apenas testar a possibilidade de implementação da arquitetura proposta. O protótipo foi desenvolvido sem levar em consideração questões de desempenho e, a apresentação de um grande número de resultados tornaria sua execução lenta.

5.3 GERAÇÃO DE MODELO WS-BPEL

A definição de um *workflow* WS-BPEL está contida em um elemento <process>. Este elemento é composto por diversos outros elementos, como <partnerLinks> e <variables>, e uma atividade que representa um ponto de entrada para o *workflow* (normalmente isto é representado pelo elemento <sequence>) [TAYLOR *et al.*, 2006].

WS-BPEL provê um conjunto de construções simples para enviar e receber mensagens. Tipicamente, um *workflow* WS-BPEL, irá começar com uma atividade de recebimento de mensagem (<receive>) e irá terminar com uma atividade resposta de mensagem (<reply>), que irá responder à mensagem recebida inicialmente. É fácil enviar uma mensagem para outro serviço Web (que são chamados *partners* em WS-BPEL) usando a atividade de invocar (<invoke>) outros serviços. Existem duas versões de invocação, sendo uma versão em que apenas uma variável de entrada está presente, e a outra em que se está respondendo a uma requisição e uma variável de entrada e uma de saída estão presentes [TAYLOR *et al.*, 2006].

A estrutura completa de um *workflow* WS-BPEL está representada na Figura 64.

```
<process name="BpelProcessName" targetNamespace="..."
    xmlns="http://schemas.xmlsoap.org/ws/2004/03/business-process/">
  <partnerLinks>
     <partnerLink name="partnerA" partnerLinkType="wsdl:parnterALinkType"
       myRole="myRoleInRelationToPartner"/>
     ...
  </partnerLinks>
  <variables>
    <variable name="varA" messageType="wsdl:MessageA"/>
    ...
  </variables>
  <!-- this is executable part of workflow -->
  <sequence>
     <receive partnerLink="partnerA" portType="wsdl:parnterALinkType"
       operation="doSomething" variable="varA" />
     <assign>
       <copy>
         <from>$varA.someParameter</from>
         <to>$varB.anotherInfo</to>
       </copy>
     </assign>
     <invoke partnerLink="partnerB" portType="pb:anoptherPartnerPT"
       operation="doSomethingElse" inputVariable="varB"
       outputVariable="varC" />
     .... <!-- here something more happens -->
     <reply partnerLink="partnerA" portType="wsdl:parnterALinkType"
        operation="doSomething" variable="results"/>
  </sequence>
</process>
```

Figura 64 Estrutura de um *workflow* WS-BPEL [TAYLOR *et al*., 2006].

As composições semântica e estruturalmente possíveis construídas pelo *framework* como solução para a composição do *workflow* especificado são apresentadas ao usuário que, por sua vez, seleciona a que melhor atende às suas necessidades, e solicita a geração do documento WS-BPEL correspondente a ela por meio do botão *Generate WS-BPEL*. O modelo WS-BPEL é gerado a partir da composição de serviços escolhida pelo usuário. Esta composição está armazenada na base de dados e é a partir da estrutura armazenada que é gerado o documento WS-BPEL que é apresentado como resultado final do processo de composição ao usuário do *framework*.

Usamos a API JDOM [JDOM, 2010] para gerar o arquivo WS-BPEL de saída do *framework*. A JDOM é uma API simples para acessar, manipular e gerar dados em formato XML por meio de código Java. Por definição, a JDOM é configurada para utilizar o *parser* JAXP [JAXP, 2010], mas pode ser configurada para usar a maioria dos *parsers* existentes. Algumas vantagens apresentadas por essa biblioteca, que influenciaram a decisão de utilizá-la, são: facilidade de uso, o XML gerado é “limpo”, e existe possibilidade de integração com outras APIs XML.

A geração do documento WS-BPEL é realizada pela função *generateWSBPEL*, que recebe como parâmetro um objeto *WorkflowExe*, assim como a função que realiza a análise estrutural dos serviços. Cada elemento do documento WS-BPEL corresponde a um objeto *org.jdom.Element*. A Figura 65 apresenta o código da criação do primeiro e principal elemento do documento WS-BPEL, o *process*, assim como a definição de suas principais propriedades, *name* e *targetNamespace*, e os principais *namespaces* necessários para a definição dos elementos seguintes.

```
org.jdom.Element process = new org.jdom.Element("process");
process.setAttribute("name", wfExe.getName());
process.setAttribute("targetNamespace",
        "http://localhost/bpel/"+wfExe.getName()+".bpel");
process.setNamespace(org.jdom.Namespace.
        getNamespace("http://docs.oasis-open.org/wsbpel/2.0/process/" +
        "executable"));
process.addNamespaceDeclaration(org.jdom.Namespace.getNamespace("tns",
        "http://localhost/bpel/"+wfExe.getName()+".bpel"));
```

Figura 65 Código Java para criação de elemento *Process* em documento WS-BPEL.

No código da Figura 66, para cada tarefa do *workflow*, são inseridos os *namespaces* correspondentes referenciando os arquivos WSDL dos serviços que executam cada tarefa, são criados os elementos *import* que importam os arquivos WSDL de cada serviço de cada tarefa, são criados os elementos *partnerLink* que representam as tarefas, e também são inseridas a variáveis (*variables*) das tarefas. São criadas listas para cada tipo de elemento, que são inseridas no XML posteriormente.

```
int i=0;
List<TaskExe> tasks = wfExe.getTasks();
for(Iterator it = tasks.iterator(); it.hasNext();){
    TaskExe t = (TaskExe) it.next();
    Service s = t.getService();
    //namespace
    process.addNamespaceDeclaration(org.jdom.Namespace.
            getNamespace("ns"+i, s.getWsdlFile()));
    //import
    org.jdom.Element import1 = new org.jdom.Element("import");
    import1.setAttribute("namespace", s.getWsdlFile());
    import1.setAttribute("location", s.getWsdlFile());
    import1.setAttribute("importType",
            "http://schemas.xmlsoap.org/wsdl/");
    importsList.add(import1);
    //partnerlink
    org.jdom.Element partnerLink = new org.jdom.Element("partnerLink");
    partnerLink.setAttribute("name", s.getName()+"PartnerLink");
    partnerLink.addNamespaceDeclaration(org.jdom.Namespace.getNamespace(
            "ns"+i, s.getWsdlFile()));
    partnerLink.setAttribute("partnerLinkType",
            "ns"+i+":"+s.getName()+"PartnerLinkType");
    partnerLink.setAttribute("partnerRole", s.getName()+"Role");
    partnerLinksList.add(partnerLink);
    //variable
    List<entity.Input> ins = s.getInputs();
    for(Iterator ii = ins.iterator(); ii.hasNext();){
        entity.Input in = (entity.Input) ii.next();
        org.jdom.Element var = new org.jdom.Element("variable");
        var.setAttribute("name", in.getName());
        var.setAttribute("messageType",
                "ns"+i+":"+in.getName()+"Request");
        variablesList.add(var);
    }
    List<entity.Output> outs = s.getOutputs();
    for(Iterator io = outs.iterator(); io.hasNext();){
        entity.Output out = (entity.Output) io.next();
        org.jdom.Element var = new org.jdom.Element("variable");
        var.setAttribute("name", out.getName());
        var.setAttribute("messageType",
                "ns"+i+":"+out.getName()+"Response");
        variablesList.add(var);
    }
    i++;
}
```

Figura 66 Código para criação de elementos relacionados às tarefas do *workflow*.

A Figura 67 apresenta a criação dos elementos WS-BPEL a partir das listas criadas a partir das tarefas do *workflow*. Os elementos *import*, *partnerLinks* e *variables* são inseridos em *process*.

```
//imports
process.addContent(importsList);
//partnerlinks
process.addContent(partnerLinks);
partnerLinks.addContent(partnerLinksList);
//variable
process.addContent(variables);
variables.addContent(variablesList);
```

Figura 67 Inserção dos elementos relacionados às tarefas do *workflow*.

A geração da parte do documento WS-BPEL que representa a execução das tarefas do *workflow* é realizada a partir da análise das relações entre as tarefas, e também considerando a sequência de execução das mesmas. Assim como na análise estrutural dos serviços, são consideradas seis possibilidades de relação entre as tarefas do *workflow*: *SEQUENTIAL*, *AND-SPLIT*, *AND-JOIN*, *OR-SPLIT*, *OR-JOIN*, e *FINAL*.

Como citado anteriormente, um *workflow* WS-BPEL tipicamente começa com uma atividade de recebimento de mensagem (<receive>) e termina com uma atividade resposta de mensagem (<reply>). Sendo assim, é analisada a sequência de execução das tarefas a fim de definir se deve ser criado um elemento <receive>, no caso da tarefa ser a primeira a ser executada. A Figura 68 apresenta o código de criação do elemento <receive> para a primeira tarefa do *workflow*.

```
if(sequence == 1){
    org.jdom.Element receive1 = new org.jdom.Element("receive");
    receive1.setAttribute("name", task.getService().getName());
    receive1.setAttribute("partnerLink",
            task.getService().getName()+"PartnerLink");
    receive1.setAttribute("portType",
            task.getService().getPortType());
    receive1.setAttribute("operation",
            task.getService().getName());
    for(int k=0; k<inputs.size(); k++){
        receive1.setAttribute("inputVariable",
                inputs.get(k).getName());
    }
    for(int k=0; k<outputs.size(); k++){
        receive1.setAttribute("outputVariable",
                outputs.get(k).getName());
    }
    sequence1.addContent(receive1);
}
```

Figura 68 Criação de elemento <receive> para primeira tarefa a ser executada no *workflow.*

Se a relação da tarefa é *FINAL*, é possível concluir que ela é a última a ser executada no *workflow*, dessa forma, deve ser criado um elemento <reply>. A Figura 69 apresenta o código de criação do elemento <reply> para a última tarefa do *workflow*.

```
}else if(relation.toUpperCase().equals("FINAL")){
    //reply
    org.jdom.Element reply = new org.jdom.Element("reply");
    reply.setAttribute("name", task.getService().getName());
    reply.setAttribute("partnerLink",
            task.getService().getName()+"PartnerLink");
    reply.setAttribute("portType",
            task.getService().getPortType());
    reply.setAttribute("operation",
            task.getService().getName());
    for(int k=0; k<inputs.size(); k++){
        reply.setAttribute("inputVariable",
                inputs.get(k).getName());
    }
    for(int k=0; k<outputs.size(); k++){
        reply.setAttribute("outputVariable",
                outputs.get(k).getName());
    }
    sequence1.addContent(reply);
}
```

Figura 69 Criação do elemento <reply> para a última tarefa do *workflow.*

Para cada tarefa, independente da sua relação com a seguinte, é criado um elemento <invoke>, que invoca o serviço que vai executá-la. A Figura 70 apresenta o código para a criação do elemento <invoke> de uma tarefa.

```
org.jdom.Element invoke1 = new org.jdom.Element("invoke");
invoke1.setAttribute("name", task.getService().getName());
invoke1.setAttribute("partnerLink",
        task.getService().getName()+"PartnerLink");
invoke1.setAttribute("portType",
        task.getService().getPortType());
invoke1.setAttribute("operation",
        task.getService().getName());
for(int k=0; k<inputs.size(); k++){
    invoke1.setAttribute("inputVariable",
            inputs.get(k).getName());
}
for(int k=0; k<outputs.size(); k++){
    invoke1.setAttribute("outputVariable",
            outputs.get(k).getName());
}
sequence1.addContent(invoke1);
```

Figura 70 Criação do elemento <invoke> para uma tarefa do *workflow.*

6. CONSIDERAÇÕES FINAIS

No contexto de *e-Science*, os recursos computacionais estão se tornando cada vez mais importantes para a realização de pesquisas científicas, o que vem acompanhado por uma proliferação de dados e ferramentas. A proliferação de recursos gera problemas e dificuldades na gestão dos mesmos e, para as atividades de *e-Science* muitos esforços estão sendo feitos no sentido de facilitar a gestão destes recursos e melhorar o desempenho das pesquisas científicas. Assim, a quantidade de serviços relacionados a *e-Science* disponíveis na Web vem crescendo muito, portanto, a busca por ferramentas que auxiliem na descoberta, seleção, composição e invocação destes serviços também vem se ampliando.

Um componente muito importante para a pesquisa científica, no contexto de *e-Science*, é o *workflow* científico, que representa experimentos científicos e pode ser criado utilizando serviços Web como componentes. Dentro do contexto em que *e-Science*, *workflows* científicos e serviços Web são partes importantes, o uso de tecnologias da Web Semântica, como serviços Web semânticos e ontologias, facilita a descoberta de serviços, a composição dos *workflows* e todo o processo de experimentação e obtenção de resultados.

O desenvolvimento de experimentos científicos computacionais é um processo complexo. Existem diferentes componentes de *software* que podem ser utilizados em conjunto para a realização de um experimento como, por exemplo, serviços Web [GIL *et al.*, 2010]. Neste contexto, o objetivo deste trabalho é automatizar a composição de *workflows* científicos a partir de serviços Web semânticos, trazendo benefícios para os cientistas, como os seguintes:

- Redução do ciclo de geração de experimentos computacionais. A partir da descrição semântica do *workflow*, serviços Web semânticos são descobertos e compostos, gerando o modelo do *workflow* para o usuário.
- Aumento da eficiência na experimentação e descobertas científicas. Com a redução do ciclo de geração de experimentos, existe mais tempo para a execução do experimento e obtenção de resultados. Além disso, a busca automática de serviços existentes que atendam às especificações do *workflow* reduz a necessidade de desenvolvimento de novos serviços, por meio do reuso de serviços prontos.
- Facilidade para o usuário. O usuário que não possui grande conhecimento a respeito de sistemas de *workflow* e informática em geral é capaz de definir semanticamente o *workflow* científico que deseja compor.

- Compartilhamento de experimentos. A partir da obtenção do modelo formal de um *workflow* científico, é possível compartilhar esse modelo com outros cientistas que, por sua vez, podem reproduzir o experimento representado pelo *workflow* em questão.

O *framework*, denominado Composer-Science, desenvolvido no contexto deste trabalho, auxilia na descoberta e composição de serviços Web semânticos a fim de gerar um *workflow* científico. O *framework* tem o objetivo de automatizar esse processo por meio do uso de tecnologias da Web semântica como ontologias e serviços Web semânticos.

A abordagem apresentada tem como principal objetivo solucionar os problemas de busca semântica, análise estrutural, extração de informações de documentos WSDL, OWL-S e OWL, e inferências semânticas. Questões relacionadas ao desempenho do protótipo desenvolvido foram deixadas para versões futuras, bem como também outras possibilidades de utilização dos recursos semânticos fornecidos pelas tecnologias usadas.

Porém, em um domínio bem definido, com serviços que apresentem descrições semânticas completas e ontologias conhecidas, a abordagem se mostra promissora, no sentido de que é capaz de auxiliar o trabalho de descoberta e composição de serviços a fim de gerar um modelo de *workflow* científico ao final de sua execução.

Esta dissertação justificou a importância do tema e da abordagem adotada e apresentou a proposta do *framework* Composer-Science baseado em ontologias e serviços Web semânticos para a composição de *workflows* científicos. Um protótipo foi construído instanciando-se o *framework* com o uso de diversas ferramentas usadas em aplicações para a Web semântica, com o objetivo de verificar a viabilidade da arquitetura proposta e apresentar a sua real aplicabilidade. Um estudo de caso relacionado ao domínio biológico foi desenvolvido por meio da execução do protótipo desenvolvido.

Considerando os objetivos ressaltados na introdução deste trabalho, podemos destacar que:

- O *framework* proposto permite a descrição semântica do *workflow* e de suas tarefas individualmente a partir do uso de termos presentes em uma ontologia de domínio.
- A partir das descrições semânticas fornecidas pelo usuário, o *framework* é capaz de realizar inferências semânticas em ontologias de domínio e buscar serviços semanticamente compatíveis a partir das descrições e dos resultados obtidos por meio das inferências em seus repositórios.
- O *framework* é capaz de realizar a análise estrutural dos serviços encontrados para realizar as tarefas de um *workflow*, definindo se um modelo executável semanticamente possível é também estruturalmente possível.

- Os repositórios do *framework*, assim como a quantidade de ontologias de domínio disponíveis podem ser estendidos por meio do registro de novos arquivos OWL (ontologias) e OWL-S (serviços Web semânticos) pelo usuário.
- A partir de um modelo de *workflow* executável semântica e estrutural possível, o *framework* gera um documento WS-BPEL que contém a especificação do modelo.

Outro ponto de que podemos destacar como contribuição deste trabalho é a revisão bibliográfica feita, principalmente considerando o estado da arte em composição de workflows científicos, onde foram detalhados as diferentes abordagens propostas e trabalhos existentes.

Podemos destacar como trabalhos futuros o estudo de questões relacionadas à proveniência de dados e uso de semântica, além de técnicas de mapeamento entre ontologias que melhorem a busca por serviços Web semânticos relacionados a um dado domínio. A criação de um de repositório de modelos de *workflows* pode permitir a reprodução de experimentos por outros cientistas. A possibilidade de modificação dos modelos presentes nos repositórios, assim como a possibilidade de modificação dos modelos gerados pelo *framework* para um modelo de *workflow* criado pelo usuário pode permitir maior flexibilidade na execução do *workflow*.

A fim de obter um resultado mais completo para o pesquisador que solicita a geração de um modelo de *workflow* científico, a integração do *framework* proposto com um Sistema de Gerenciamento de *Workflow* (SGWf) pode permitir a execução do *workflow* obtido, assim como outras possibilidades de aproveitamento do mesmo e de seus resultados por meio do uso das funcionalidades fornecidas pelo SGWf.

O protótipo do *framework* apresentado apresenta algumas limitações relacionadas ao seu desempenho. As inferências realizadas para a realização da busca semântica também podem ser aprimoradas em versões futuras, de forma a garantir maior exatidão nos resultados das buscas semânticas. A análise estrutural dos serviços é realizada de forma que os dados de entrada e saída dos serviços tenham compatibilidade exata, sendo assim, questões relacionadas à transformação de grandezas para tipos iguais também podem ser tratadas em trabalhos futuros.

REFERÊNCIAS

ActiveBPEL. The open source BPEL engine. 2006. Disponível em: http://activebpel.org. Acesso em 13 set. 2009.

AKKIRAJU, R. *et al*. Web Service Semantics: WSDL-S. 2005. Disponível em: http://www.w3.org/Submission/WSDL-S/. Acesso em 17 jun. 2009.

ALBRECHT, F. Taverna – Workflows para bioinformática. 2007. Disponível em: pihisall.wordpress.com/2007/01/19/taverna-workflows-para-bioinformatica/. Acesso em: 26 abr. 2010.

ALMEIDA, M. B., BAX, M. P. Taxonomia para projetos de integração de fontes de dados baseados em ontologias. *V Encontro Nacional de Pesquisa em Ciência da Informação*. Belo Horizonte, 2003.

ASKALON. Askalon – Grid Application Development and Computing Environment. 2010. Disponível em: http://www.dps.uibk.ac.at/projects/askalon/. Acesso em 14 abr. 2010.

ATKINS, D. E., DROEGEMEIER, K. K., FELDMAN, S. I., GARCIA-MOLINA, H., Klein M. L., MESSERCHMITT, D. G., MESSINA, P., OSTRIKER, J. P., WRIGHT, M. H. Revolutionizing Science and Engineering Through Cyberinfrastructure. 2003. Disponível em: http://www.nsf.gov/publications/pub_summ.jsp?ods_key=cise051203. Acesso em 10 abr. 2010.

BALBI, R., PIRES, P., MATTOSO, M. BioProvenance: Um framework para a proveniência de dados aplicado à bioinformática. *In*: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 20, 2005, Uberlândia. Anais... Uberlândia: [s.n.], [2005].

BARROS, A., DUMAS, M., OAKS, P. A Critical Overview of the Web Services Choreography Description Language (WS-CDL). 2005. Disponível em: http://www.bptrends.com/publicationfiles/03%2D05%20WP%20WS%2DCDL%20Barros%20et%20al%2Epdf. Acesso em 25 out. 2009.

BARRY, D. K. Service-oriented architecture (SOA) definition. 2010. Disponível em: http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html. Acesso em 14 abr. 2010.

BARTHELMESS, P. Sistemas de Workflow: Análise da Área e Proposta de Modelo. Campinas, 1996. 151f. Tese (Mestrado em Ciência da Computação) – Instituto de Computação, Universidade Estadual de Campinas.

BENATALLAH, B., DUMAS, M., FAUVET, M., RABHI, F. Towards Patterns of Web Services Composition. *Patterns and Skeletons for Parallel and Distributed Computing,* Springer Verlag, UK. 2002.

BERNERS-LEE, T., HENDLER, J., LASSILA, O. The Semantic Web. A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, 01/05/2001. Disponível em: http://www.librarything.com/work/2389150. Acesso em 08 abr. 2010.

BioMart. 2010. Disponível em: http://www.biomart.org/. Acesso em 20 mai. 2010.

BIRN. Biomedical Informatics Research Network. 2010. Disponível em: http://www.birncommunity.org/. Acesso em 14 abr. 2010.

BizTalk. Microsoft BizTalk Server. 2009. Disponível em: http://www.microsoft.com/biztalk. Acesso em 24 out. 2009.

BRADSHAW, J. M., JEFFERS, R., JOHNSON, M., TATE, A., DALTON, J., USZOK, A., AITKEN, S. KAoS Policy Management for Semantic Web Services. *IEEE Intelligent Systems*, 19, 2004.

BORST, W.N., 1997. Construction of Engineering Ontologies. Phd Thesis.Disponível em: http://www.ub.utwente.nl/webdocs/inf/1/t0000004.pdf. Acesso em 08 abr. 2010.

caBIG. Cancer Biomedical Informatics Grid. 2010. Disponível em: https://cabig.nci.nih.gov/. Acesso em 14 abr. 2010.

CARDOSO, J. Semantic Web Services: Theory, Tools, and Applications. *New York: Information Science Reference.* 2007.

CARVALHO A. *et al*. Grandes Desafios da Pesquisa em Computação no Brasil: 2006-2016. Seminário Grandes Desafios da Pesquisa em Computação no Brasil. São Paulo, 2006.

CHARIF, Y., SABOURET, N. Dinamic Web Service Selection and Composition: An Approach based on Agent Dialogues. 2006. Disponível em: http://www.springerlink.com/content/a8817156q1382115/. Acesso em 14 abr. 2010.

CHINNICI, R. *et al.* Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. 2007. Disponível em: http://www.w3.org/TR/wsdl20/. Acesso em 17 jun. 2009.

COOLAHAN, J.E., ROUSSOPOULOS, N. Timing Requirements for Time- Driven Systems Using Augmented Petri Nets, *IEEE Transaction on Software Engineering*. 1983.

CORCHO, O., FERNANDEZ-LOPEZ, M., GOMEZ-PEREZ, A. Ontological Engineering: What are ontologies and how can we build them?. In: CARDOSO, J. *Semantic Web Services: Theory, Tools and Applications*. New York: Information Science Reference. 2007. Cap. 3, p. 44-70.

CURBERA, F., NAGY, W. A., WEERAWARANA, S. Web Services: Why and How?. 2001. Disponível em: http://www.research.ibm.com/people/b/bth/OOWS2001/nagy.pdf. Acesso em: 17 jun. 2009.

DA PENHA, D. O., DE FREITAS, H. C., MARTINS, C. A. P. S. Modelagem de Sistemas Computacionais usando Redes de Petri: aplicação em projeto, análise e avaliação. 2004. Disponível em: www.sbc.org.br/bibliotecadigital/download.php?paper=33. Acesso em 25 out. 2009.

DDBJ. 2010. Disponível em: http://www.ddbj.nig.ac.jp. Acesso em 20 mai. 2010.

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

DOE. 2010. U. S. Department of Energy. Disponível em: http://www.doe.gov. Acesso em 20 mai. 2010.

EMBL-EBI. 2010. Disponível em: http://www.ebi.ac.uk. Acesso em 20 mai. 2010.

EMMERICH, W. *et al.* Grid Service Orchestration using the Business Process Execution Language (BPEL). 2006. Disponível em: http://www.cs.ucl.ac.uk/staff/w.emmerich/publications/JoGC/Workflow/bpel.pdf. Acesso em 24 out. 2009.

ENTREZ Web Service for NCBI. 2008. Disponível em: http://fiehnlab.ucdavis.edu/staff/scholz/. Acesso em 14 abr. 2010.

ESG. Earth System Grid. 2010. Disponível em: http://www.earthsystemgrid.org/. Acesso em 14 abr. 2010.

GenBank. GenBank at NCBI. 2010. Disponível em: http://www.ncbi.nlm.nih.gov/Genbank. Acesso em 20 mai. 2010.

GIBAS, C., JAMBECK, P. Desenvolvendo bioinformática: ferramentas de software para aplicação em biologia. Rio de Janeiro. Ed. Campus. 2001.

GIL, Y., GONZÁLEZ-CALERO, P. A., KIM, J., MOODY, J., RATNAKAR, V. A Semantic Framework for Automatic Generation of Computational Workflows Using Distributed Data and Component Catalogs. In *Journal of Experimental and Theoretical Artificial Intelligence*. 2010. Disponível em: http://www.isi.edu/~gil/papers/gil-etal-jetai10.pdf. Acesso em 14 abr. 2010.

GIL, Y., DEELMAN, E., ELLISMAN, M., FAHRINGER, T., FOX, G., GANNON, D., GOBLE C., LIVNY, M., MOREAU, L., MYERS, J. Examining the Challenges of Scientific Workflows. *IEEE Computer*, vol. 40, no. 12, pp. 24-32. 2007.

GOBLE, C., KESSELMAN, C., SURE, Y., Semantic Grid: The Convergence of Technologies, *Dagstuhl Seminar Proceedings*, 2005.

GOBLE, C., ROURE, D. The Impact of Workflow Tools on Data-centric Research. In HEY, T., TANSLEY, S., TOLLE, K. The Fourth Paradigm: Data-Intensive Scientific Discovery. p. 137-145. 2009.

GRANDA, M., DRAKE, J. M., GREGORIO, J. Performance evaluation of parallel systems by using unbounded generalized stochastic petri nets. *IEEE Trans. on Software Engineering*, Vol. 18, n. 1, p. 55-71. 1992.

GRUBER, T. A Translation Approach to Portable Ontology Specification, *Proceedings of Japanese Knowledge Acquisition Workshop (JKAW92)*, 1992.

GRUBER, T. R. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. In: International Journal Human-Computer Studies Vol. 43, Issues 5-6, November 1995, pp.907–928.

HAASE, P., AGARWAL, S., SURE, Y. Service-Oriented Semantic Peer-to-Peer Systems. *In* BUSSLER, C. *et al.*, Workshop Web Information Systems Engineering, volume 3307 of LNCS, p. 46–57. Springer-Verlag, 2004.

HAMADI, R., BENATALLAH, B. A Petri Net-based Model for Web Service Composition. *Fourteenth Australasian Database Conference (ADC2003)*, Adelaide, Australia. 2003.

HEY, T., TANSLEY, S., TOLLE, K. The Fourth Paradigm: Data-Intensive Scientific Discovery. Microsoft Research. Redmond, Washington. 2009.

HINE, C. M. New infrastructures for knowledge production: understanding E-science. 1.ed. United States of America: Information Science Publishing, 2006. 306p.

IBM. 2010. Disponível em: http: //www.ibm.com/us/em. Acesso em 05 mai. 2010.

JAXP. 2010. Disponível em: https://jaxp.dev.java.net/. Acesso em 02 jun. 2010.

JBoss jBPM. 2009. Disponível em: http://jboss.org/. Acesso em 24 out. 2009.

JDOM. 2010. Disponível em: http://www.jdom.org/index.html. Acesso em 02 jun. 2010.

JENA. JENA - A Semantic Web Framework for Java. 2010. Disponível em: http://jena.sourceforge.net/. Acesso em 03 mai. 2010.

JENSEN, K., HUBER, P., SHAPIRO, R. M. Hierarchies in Coloured Petri Nets. Lectures Notes in Computer Science, Vol.483, p. 313-341, Springer-Verlag. 1990.

JORDAN, D. *et al.* Oasis web services business process execution language (wsbpel) v2.0. 2007. Disponível em: http://docs.oasis-open.org/wsbpel/2.0/. Acesso em 11 set. 2009.

KARAKOC, E., SENKUL, P. Composing semantic Web services under constraints. 2009. Disponível em: http://www.elsevier.com/locate/eswa. Acesso em 29 abr. 2010.

KEPLER. The Kepler Project. 2010. Disponível em: https://kepler-project.org/. Acesso em 14 abr. 2010.

KRUCHTEN, P. The Rational Unified Process: an Introduction. Pearson Education, Inc. Boston. 2004.

LAMBRIX, P., TAN, H., JAKONIENE, V., STRÖMBÄCK, L. Biological Ontologies. In: *Semantic Web: Revolutionizing Knowledge Discovery in the Life Sciences*. New York: Information Science Reference. 2007. Cap. 4, p. 85-99.

LAUSCHNER, T. Ambientes de e-Science e a Evolução dos Padrões de Arquitetura de Computação em Grade. Rio de Janeiro, 2005. Disponível em: http://www-di.inf.puc-rio.br/~endler/semGSD/monografias/. Acesso em: 10 abr. 2010.

LEMOS, M. Workflow para bioinformática. Tese (Doutorado em Informática) – Programa de Pós-Graduação em Informática, Pontíficia Universidade Católica do Rio de Janeiro. 239f. Rio de Janeiro. 2004.

LESK, A. M. Introdução à Bioinformática. 2 ed. Porto Alegre. Ed. Artmed. 2008.

LEYMANN, F., ROLLER, D., THATTE, S. Goals of the BPEL4WS Specification. 2007. Disponível em: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel. Acesso em 11 set. 2009.

LUDÄSCHER, B., ALTINTAS, I., BERKLEY, C., HIGGINS, D., JAEGER, E., JONES, M., LEE, E. A., TAO, J., ZHAO, Y., Scientific Workflow Management and the KEPLER System. 2004. Disponível em: www.sdsc.edu/~ludaesch/Paper/kepler-swf.pdf. Acesso em: 10 abr. 2010.

MACEDO, J. A. F., HAEUSLER, E. H. Um modelo conceitual para biologia molecular. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro. Rio de Janeiro, RJ, Brasil. 2005.

MATTOS, A., MATTOSO, M. Uma Estratégia para Gerência de Dados de Workflows Científicos no Contexto da Bioinformática. In: WORKSHOP DE TESES E DISSERTAÇÕES EM BANCOS DE DADOS, 6, 2007, João Pessoa. Anais... João Pessoa: [s.n.], [2007].

MATTOS, A., SILVA, F. C., RUBERG, N., CRUZ, S. M. S. MATTOSO, M. L. Q. Gerência de Workflows Científicos: Uma Análise Crítica no Contexto da Bioninformática. 2008. Disponível em: http://www.cos.ufrj.br/uploadfiles/1204740464.pdf. Acesso em 26 abr. 2010.

MATOS, E. E., NOCELLI, C., BRAGA, R., CAMPOS F. MathWS: Broker de Serviços Web para e-Science. 2007. Disponível em: http://www.lbd.dcc.ufmg.br/bdbcomp/servlet/Trabalho?id=7301. Acesso em 26 abr. 2010.

MATOS, E. E. CelOWS: Um Framework Baseado Em Ontologias Com Serviços Web Para Modelagem Conceitual Em Biologia Sistêmica. Dissertação de Mestrado. Mestrado em Modelagem Computacional – Universidade Federal de Juiz de Fora. 2008.

MATOS, E. E., MENDES, L. F., CAMPOS F., BRAGA, R. ASOW-Science: a service oriented framework to support e-Science Applications. In: IEEE IRI 2009, Las Vegas. *IEEE Procedings of Internacional on Information Reuse and Integration*, 2009.

MathML. Mathematical Markup Language. 2008. Disponível em: http://www.w3.org/TR/MathML3/. Acesso em 26 abr. 2010.

MURATA, T. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, Vol. 77, No. 4, p. 541-579. 1989.

MILLER, J., ARNOLD, J., CARDOSO, J., SHETH, A., KOCHUT, K. Quality of Service for Workflows and Web Service Processes. *Journal of Web Semantics*, 2004.

NARDI, A. R. Uma arquitetura de baixo acoplamento para execução de padrões de controle de fluxo em grades. Tese de Doutorado. Doutorado em Ciência da Computação. Instituto de Matemática e Estatística da Universidade de São Paulo. 2009.

NCBI. National Center of Biotechnology Information, FASTA Format. 2010. Disponível em: http://www.ncbi.nlm.nih.gov/BLAST/fasta.shtml. Acesso em 21 mai. 2010.

NIH. 2010. National Institutes of Health. Disponível em: http://www.nih.gov. Acesso em 20 mai. 2010.

NVO. National Virtual Observatory. 2010. Disponível em: http://www.virtualobservatory.org/. Acesso em 14 abr. 2010.

OASIS. Web Services Business Process Execution Language Version 2.0. 2007. Disponível em: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel. Acesso em 11 abr. 2010.

ORACLE. Oracle BPEL Process Manager. 2009. Disponível em: http://www.oracle.com/technology/products/ias/bpel/index.html. Acesso em 24 out. 2009.

OWL. OWL Web Ontology Language Reference. Disponível em: <http://www.w3.org/TR/owl-ref/>. Acesso em 17 jun. 2009. 2004.

OWL. 2010. OWL Web Ontology Language Overview. Disponível em http://www.w3.org/TR/owl-features/. Acesso em 03 abr. 2010.

OWL-S API. 2010. Disponível em: http://www.mindswap.org/2004/owl-s/services.shtml. Acesso em 10 jun. 2010.

OWL-S. OWL-based Web Service Ontology. 2004. Disponível em: http://www.daml.org/services/owl-s/. Acesso em: 17 jun. 2009.

PALAZZI, D., SILVA, L., MENDES, L. F., GASPAR, W., MATOS, E., CAMPOS, F. C. A., BRAGA, R. M. M. Uso de Ontologias em Projetos de e-Science. In: II Seminário de Pesquisa em Ontologia, 2009, IME- Instituto Militar de Engenharia, Rio de Janeiro, RJ.

PARASTATIDIS, S. A Platform for All That We Know: Creating a Knowledge-Driven Research Infrastructure. In HEY, T., TANSLEY, S., TOLLE, K. The Fourth Paradigm: Data-Intensive Scientific Discovery. p. 165-172. 2009.

PELLET. Pellet: The Open Source OWL 2 Reasoner. 2010. Disponível em: http://clarkparsia.com/pellet/. Acesso em 03 mai. 2010.

PENNINGTON, D. D. Supporting Large-Scale Science With Workflows. 2007. Disponível em : http://delivery.acm.org/10.1145/1280000/1273369/p45-pennington.pdf?key1=1273369&key2=0019480911&coll=GUIDE&dl=ACM&CFID=15151515&CFTOKEN=6184618. Acesso em: 10 abr. 2010.

PEGASUS. 2010. Disponível em: http://pegasus.isi.edu/. Acesso em 14 abr. 2010.

PELTZ, C. Web Services Orchestration and Choreography. 2003. Disponível em: http://www.computer.org/portal/web/csdl/doi/10.1109/MC.2003.1236471. Acesso em 25 out. 2009.

PETRI, C. Kommunikation mit Automaten. Ph.D. Dissertation, Darmstadt Universitat, West Deutschland, 1962.

PIGNOTTI, E., EDWARDS, P., PREECE, A., GOTTS, N., POLHILL, G. Enhancing workflow with a semantic description of scientific intent. *5th European Semantic Web Conference.* Tenerife, Espanha. 2008.

POLLERES, A., TOMA, I., FENSEL, D. Modeling Services for the Semantic Grid. *In*

PROTÉGÉ. 2010. The Protégé Ontology Editor and Knowledge Acquisition System. Disponível em http://protege.stanford.edu. Acesso em 08 abr. 2010.

RACER. 2010. Disponível em: http://www.racer-systems.com/products/download/. Acesso em 03 mai. 2010.

RDF. 2004. Resource description framework (RDF). Disponível em http://www.w3.org/RDF/. Acesso em 03 abr. 2010.

RDFS. 2004. RDF vocabulary description language 1.0: RDF schema. Disponível em http://www.w3.org/TR/rdf-schema/. Acesso em 03 abr. 2010.

RIBEIRO, J. L. V. Orquestração e Composição de Serviços Web Usando BPEL. Dissertação de Mestrado. Mestrado em Engenharia de Computadores e Telemática. Universidade de Aveiro. Portugal. 2008.

ROSEN, M., LUBLINSKY, B., SMITH, K. T., BALCER, M. J. Applied SOA: Service-Oriented Architecture and Design Strategies. Wiley Publishing, Inc. Indianapolis. 2008.

RUSSEL, N. *et al.* Workflow Control-flow Patterns: A revised view. 2006. Disponível em: http://www.workflowpatterns.com/documentation/documents/BPM-06-22.pdf. Acesso em 25 out. 2009.

SALIMIFARD, S., WRIGHT M. Petri net based modelling of workflow systems: an overview. *European Journal of Operational Research*, v. 134, p. 664-676. 2001.

SÁNCHEZ, F. G., GARCÍA, R. V., BÉJAR, R. M., BREIS, J. T. F. An ontology, intelligent agent-based framework for the provision of semantic web services. 2009. Disponível em: http://www.sciencedirect.com. Acesso em 26 abr. 2010.

SBC. Grandes Desafios da Pesquisa em Computação no Brasil – 2006 – 2016. Relatório sobre o seminário realizado em 8 e 9 de maio de 2006. 2006.

SHIN, D. H., LEE, K. H., SUDA, T. Automated Generation of Composite Web Services Based on Functional Semantics. 2009. Disponível em: http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B758F-4WGK4K2-1&_user=10&_coverDate=12%2F31%2F2009&_rdoc=1&_fmt=high&_orig=search&_sort=d&_docanchor=&view=c&_searchStrId=1223974485&_rerunOrigin=google&_acct=C000050221&_version=1&_urlVersion=0&_userid=10&md5=bff7413d89a52201fdaf77313217b057. Acesso em 26 jan. 2010.

SINGH, M. P., HUHNS, M. N., Service-Oriented Computing: Semantics, Process, Agents. John Willey & Sons Ltd. 2005.

SIRIN, E. OWL-S API. 2004. Disponível em: http://www.mindswap.org/2004/owl-s/api/index.shtml. Acesso em 01 fev. 2010.

STUDER, R., GRIMM, S., ABECKER, A. Semantic Web Services: Concepts, Technologies and Applications. Springer. Germany, 2007.

TAVERNA. Taverna Workflow System. 2010. Disponível em: http://www.taverna.org.uk/. Acesso em 26 abr. 2010.

TAYLOR, I. J., DEELMAN, E., GANNON, D. B., SHIELDS, M. Workflows for e-Science: Scientific Workflows for Grids. 2006. Springer. 1 edição. 530 p.

VAN DER AALST, W. M. P. The Application of Petri Nets to Workflow Management. 1998. Disponível em: http://wwwis.win.tue.nl/~wvdaalst/publications/p53.pdf. Acesso em 14 abr. 2010.

VIEIRA, T. A. S. C., CASANOVA, M. A. Execução Flexível de Workflows. Rio de Janeiro. 2005. 429p. Tese de Doutorado.Departamento de Informática - Pontifícia Universidade Católica do Rio de Janeiro.

VISTRAILS. 2010. Disponível em: http://www.vistrails.org/index.php/Main_Page. Acesso em 26 abr. 2010.

WASHINGTON, W. M. *et al*. National Science Board 2020 Vision for the NSF. 2005. Disponível em: www.nsf.gov/publications/pub_summ.jsp?ods_key=nsb05142. Acesso em 10 abr. 2010.

W3C. 2004. Web Services Architecture. Disponível em: http://www.w3.org/TR/ws-arch/. Acesso em 08 abr. 2010.

WfMC. 1995. Workflow Management Coalition – The Workflow Reference Model. Disponível em: http://www.wfmc.org/reference-model.html. Acesso em 10 abr. 2010.

WfMC. Workflow Management Coalition – Terminology and Glossary. 1999. Disponível em: http://www.wfmc.org/Download-document/WFMC-TC-1011-Ver-3-Terminology-and-Glossary-English.html. Acesso em 03 mai. 2010.

WebSphere. IBM WebSphere Process Server. 2009. Disponível em: http://www-01.ibm.com/software/websphere/. Acesso em 24 out. 2009.

WILLIAMS, A. BioMart and EMBOSS Analysis. 2009. Disponível em: http://www.myexperiment.org/workflows/997. Acesso em 20 mai. 2010.

WS-BPEL. Web Services Business Process Execution Language Version 2.0. 2007. Disponível em: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. Acesso em 20 Dez. 2009.

WSDL4J. Web Service Description Language for Java Toolkit. Disponível em: <http://wsdl4j.sourceforge.net/>. Acesso em: 02 Dez. 2009. 2005.

WSFL. Web Services Flow Language. 2007. Disponível em: http://www.ibm.com/developerworks/webservices/library/ws-ref7/. Acesso em 11 set. 2009.

WSML. Web Service Modeling Language (WSML). 2004. Disponível em: http://www.wsmo.org/wsml/. Acesso em: 17 jun. 2009.

WSMO. Web Services Modeling Ontology (WSMO). 2004. Disponível em: http://www.wsmo.org. Acesso em: 17 jun. 2009.

WSMX. Web Service Execution Environment (WSMX). 2004. Disponível em: http://www.wsmx.org. Acesso em 17 jun. 2009.

XLANG. XLANG/s Language. 2007. Disponível em: http://msdn2.microsoft.com/en-us/library/aa577463.aspx. Acesso em 11 set. 2009.

APÊNDICE

Apêndice 1

OWL-S

A1.1 Detalhamento da estrutura da ontologia OWL-S

A OWL-S é uma ontologia de serviços por meio da qual é possível representar as descrições semânticas de serviços Web. A estrutura dessa ontologia está baseada em três partes principais: service Profile, servicemodel e servicegrounding.

ServiceProfile

A classe *ServiceProfile* descreve o serviço. O relacionamento entre as classes *ServiceProfile* e *Service* é bidirecional, por meio das propriedades *presents* e *presentedBy*, respectivamente. A propriedade *presents* relaciona uma instância de *Service* com uma instância de *ServiceProfile*, e sua inversa, *presentedBy*, relaciona uma instância de *ServiceProfile* com uma instância de *Service*.

A classe *Profile* é uma subclasse de *ServiceProfile*, podendo ser uma representação de um perfil de um serviço. Esta representação contém três propriedades dirigidas ao ser humano e não processáveis por máquina: (1) a propriedade *contactInformation*, que descreve o fornecedor do serviço; (2) a propriedade *serviceName*, de cardinalidade igual a 1, usada como um identificador do serviço; e (3) a propriedade *textDescription*, cujo valor é uma descrição textual da funcionalidade oferecida pelo serviço.

Além dessas propriedades, a classe *Profile* oferece uma descrição das características funcionais de um serviço, sob dois aspectos: a transformação da informação (representada pelas entradas e saídas) e a mudança de estado produzida pela execução do serviço (representada pelas pré-condições e pelos efeitos do serviço).

Em OWL-S existe o conceito de *Parameter*, subclasse da classe *Variable*. Esta classe *Parameter* é a superclasse tanto da classe que representa os parâmetros de entrada

de um processo (*Input*) quanto os de saída (*Output*). *Parameter*, *Input* e *Output* são definidas no contexto da ontologia de processos de OWL-S. As pré- condições e os efeitos de um processo, por sua vez, são expressões lógicas.

Para descrever os parâmetros de entrada e saída, as pré-condições e os efeitos de um serviço, a classe *Profile* conta com as seguintes propriedades, definidas na ontologia de *profiles*:

- *hasParameter*: propriedade cujo contra-domínio é a classe *Parameter* da ontologia de processos de OWL-S. Esta propriedade não é utilizada diretamente e serve apenas como uma superpropriedade das demais;
- *hasInput*: propriedade cujo contra-domínio é a classe *Input*, subclasse de *Parameter* da ontologia de processos de OWL-S. A classe *Input* representa a informação de que o processo necessita para sua execução;
- *hasOutput*: propriedade cujo contra-domínio é a classe *Output* da ontologia de processos de OWL-S. Tanto *hasOutput* como *hasInput* são subpropriedades da propriedade *hasParameter*;
- *hasPrecondition*: propriedade cujo contra-domínio é a classe *Precondition* da ontologia de processos de OWL-S;
- *hasResult*: propriedade cujo contra-domínio é a classe *Result* da ontologia de processos de OWL-S. O resultado de um processo especifica as condições sob as quais as saídas são geradas e, sob estas condições, quais são as alterações causadas pela execução do serviço.

Idealmente todas as instâncias de *Inputs*, *Outputs*, *Preconditions* e *Results* são criadas na ontologia de processos e referenciadas na classe *Profile*. Dessa forma, cada uma dessas propriedades possuem como valor, não novas instâncias criadas da classe *Profile*, mas sim uma referência para instâncias já criadas na ontologia de processos do serviço correspondente.

Por fim, a classe *Profile* apresenta um conjunto de características específicas de um serviço, por meio das propriedades:

- *serviceParameter*: uma lista extensível de propriedades, sendo que cada valor da propriedade é uma instância da classe *ServiceParameter*. Esta classe *ServiceParameter* possui as propriedades *serviceParameterName*, que indica o nome do parâmetro, e *sParameter*, que aponta para o valor do parâmetro em alguma ontologia OWL. Esta propriedade permite que diversos outros atributos sejam associados a um serviço, de acordo com a funcionalidade que ele deseja assumir.
- *serviceCategory*: se refere a uma entrada em alguma ontologia ou taxonomia de serviços existente, sendo o seu valor uma instância da classe *ServiceCategory*.

Esta classe descreve categorias de serviços e possui um nome (propriedade *categoryName*), aponta para uma taxonomia (propriedade *taxonomy*), possui um ou mais valores na taxonomia especificada (propriedade *value*) e, para cada tipo de serviço, possui um código associado na taxonomia (propriedade *code*).

Um *Profile* pode tornar públicas apenas as funcionalidades do serviço que o seu provedor desejar, independentemente de quantas outras funcionalidades o serviço for capaz de atender. No entanto, uma vez não declaradas, essas funcionalidades nunca serão encontradas e, por isso, não poderão ser utilizadas.

ServiceModel

Em OWL-S, um serviço é considerado um processo, quando visto de forma detalhada, e este processo é descrito por meio da classe *Process*, subclasse de *ServiceModel*.

Processos possuem condições vinculadas a eles que determinam situações sob as quais eles podem ser executados. Processos também podem ter um conjunto de dados de entrada e sua execução pode levar à transformação destes dados em um conjunto de dados de saída. Uma execução também pode provocar “transformações no mundo”, de um estado para outro, descritas através dos efeitos do processo. Por exemplo, em um processo de compra de um produto, o efeito pode ser o débito do valor do produto da conta corrente do comprador e o seu crédito na conta corrente do vendedor. A saída do processo, por sua vez, pode ser uma notificação de que o processo de compra foi efetuado com sucesso.

Dessa forma, para se descrever um processo em OWL-S, é necessário especificar seu conjunto de parâmetros de entrada, e de saída, suas pré-condições e seus efeitos. Além disso, é preciso definir o seu fluxo de dados, ou seja, quais entradas de subprocessos são provenientes de saídas de subprocessos anteriormente executados.

Assim como o *ServiceProfile*, o *ServiceModel* é uma representação do serviço. No entanto, estas classes são diferentes, mas devem garantir que, para um mesmo serviço, as IOPEs (*Input*, *Output*, *Precondition* e *Effects*) definidas em uma representação devem ser refletidas na outra representação.

Em OWL-S, *Inputs* e *Outputs* são subclasses de *Parameter*. Cada parâmetro possui um tipo, identificado pela propriedade *parameterType*, que indica a classe à qual seus valores pertencem. O seu escopo é o processo dentro do qual está definido. Além da definição do tipo do parâmetro, se o seu é uma constante, então ele pode ser definido em OWL-S através da propriedade *parameterValue*. Essa é a forma mais direta de se associar

um valor constante a um parâmetro de um processo, embora também seja possível através *valueData*.

A OWL-S define um outro tipo de parâmetro, chamado de *Local*, também subclasse de *Parameter*, e que, uma vez definido, pode ser usado em qualquer lugar do processo. Variáveis locais apenas podem ser usadas em processos atômicos. Para descrever o relacionamento entre um processo e cada um de seus parâmetros, tornando possível a especificação da transformação de dados, cada processo OWL-S possui as propriedades definidas na ontologia de processos *hasInput*, *hasOutput* e *hasLocal*, subpropriedades da propriedade *hasParameter*, cujos contra-domínios são, respectivamente, as classes *Input*, *Output* e *Local*. Não existe razão para que a propriedade *hasParameter* da ontologia de processos seja diretamente utilizada; ela pode ser comparada a uma classe abstrata em Java e serve apenas como superpropriedade das propriedades já citadas.

Além de parâmetros de entrada e de saída, processos em OWL-S podem estar associados a pré-condições, de modo que apenas são executados quando essas pré-condições se tornam verdadeiras. A propriedade de *Process hasPrecondition* indica a sua pré-condição de execução. Quando um processo é executado, além dos dados de saída, os *Outputs*, ele pode gerar alterações no estado do mundo, chamadas de *Effect*. Conforme já mencionado, efeitos são diferentes de dados de saída porque não são dados gerados pelo processo, mas são alterações produzidas pela execução do processo sobre algum objeto no mundo como, por exemplo, um conta corrente quando do recebimento de um dinheiro.

Para expressar as pré-condições e os efeitos de um processo, OWL-S utiliza expressões em uma linguagem específica. Assim, *Condition* e *Effect* são subclasses de *Expression*. Esta superclasse, por sua vez, possui duas propriedades de cardinalidade igual a 1: *expressionLanguage*, que identifica em qual linguagem a expressão está escrita, e *expressionBody*, que contém a expressão em si. Uma das linguagens mais citadas para representação das condições em OWL-S é a SWRL.

Em OWL-S, *Outputs* e *Effects* não são diretamente relacionados aos processos aos quais dizem respeito, uma vez que esta ligação pode ser dependente do contexto. Neste sentido, foi criada a classe *Result*, que mantém uma relação direta com a classe *Process* através da propriedade *hasResult*. Assim, o resultado de um processo em OWL-S pode conter tanto os dados de saída quanto os efeitos gerados pela execução deste processo.

A classe *Result* possui cinco propriedades. A propriedade *hasResultVar* permite a declaração de uma variável, da classe *ResultVar*, subclasse de *Parameter*, que tem seu escopo limitado ao resultado dentro do qual está sendo declarada. Uma vez declarada, esta variável pode ser usada para descrever as saídas e os efeitos daquele processo, sob a

condição definida para o resultado no qual está contida. Esta condição é definida pela propriedade *inCondition*, cujo contra-domínio é a classe *Condition*.

A propriedade opcional *withOutput*, por sua vez, com contra-domínio em *OutputBinding*, especifica as saídas do processo quando da ocorrência daquele resultado. A propriedade *hasEffect*, analogamente, especifica os efeitos da execução deste processo sob este resultado. Finalmente, existe uma propriedade associada a *Result*, chamada *resultForm*, cujo objetivo é fornecer um *template* XML abstrato para as saídas enviadas de volta ao cliente do serviço.

Vale ressaltar que mais do que um resultado pode ser definido para um mesmo processo OWL-S. No entanto, as condições sob as quais eles são válidos devem ser mutuamente exclusivas, de modo que apenas um resultado ocorra a cada execução do processo.

Uma outra característica de processo em OWL-S é que ele envolve no mínimo dois agentes: o cliente, representado pela classe *TheClient*, aquele que faz uso do serviço, e o servidor, representado pela classe *TheServer*, aquele que implementa e oferece o serviço.

Em processos compostos, a entrada de um processo componente (ou subprocesso) pode ser obtida da saída de processos componentes anteriores. Outro tipo de fluxo de dados é a especificação das saídas de um processo composto como derivações das saídas de alguns de seus processos componentes.

O conceito de *binding* foi criado através da classe *Binding*. Este conceito é então utilizado para: (1) determinar valores para os parâmetros de entrada de um processo, dentro da sua definição; e (2) determinar como valores de parâmetros de saída são especificados dentro do resultado do processo. *InputBinding* e *OutputBinding* são subclasses de *Binding*, onde *InputBinding* tem como valor da propriedade *toParam* uma instância de *Input* e *OutputBinding* uma instância de *Output*.

Associadas a classe *Binding* existem duas propriedades: *toParam*, que identifica o nome do parâmetro, e *valueSpecifier*, que corresponde à descrição de seu valor. Esta especificação de valor pode ocorrer de quatro formas distintas (subpropriedades de *valueSpecifier*): *valueSource*, *valueType*, *valueData* e *valueFunction*, dentre as quais as mais utilizadas são *valueSource* e *valueType*.

A propriedade *valueSource* possui como contra-domínio a classe *ValueOf*, que especifica as propriedades *theVar*, cujo contra-domínio é a classe *Parameter*, e *fromProcess*, com contra-domínio *Perform*. Esta propriedade permite indicar que o valor deste parâmetro é proveniente do valor de um parâmetro em outro processo, no mesmo processo composto. Desta forma, se um *Binding* com *toParam* = *p* tem *valueSource* = *s* com propriedades *theVar* = *v* e *fromProcess* = *R*, isto significa que o parâmetro *p* do processo é igual ao parâmetro *v* de *R*.

A propriedade *valueType* tem como valor uma URI (*Uniform Resource Identifier*) que aponta para uma classe OWL, indicando que o valor do parâmetro pertence a esta classe. A classe especificada deve ser subclasse da classe especificada em *parameterType* do referido parâmetro.

A propriedade *valueData* serve para a especificação de constantes em XML. A propriedade *valueFunction* representa funções parametrizadas e foi projetada com o objetivo de permitir futuras extensões da linguagem.

Assim, a propriedade *hasDataFrom* de *Process* é utilizada para relacionar *Inputs* de *Performs* a *InputBindings*, indicando de onde vêm os valores dos parâmetros de entrada do processo. A propriedade *withOutput* de *Result*, por sua vez, que tem como contra-domínio a classe *OutputBinding*, permite que o valor de um parâmetro de saída seja especificado dentro de um resultado, de acordo com a condição a ele associada.

A variável *TheParentPerform* foi introduzida em OWL-S para se referir, em tempo de execução, a uma particular instância em execução de um processo definido via *Perform*, como componente de um processo composto. Assim, torna-se possível referenciar o valor de um determinado parâmetro durante uma execução.

A classe *Process* de OWL-S corresponde a união de três classes: (1) *AtomicProcess*, que representa processos que podem ser diretamente invocados, que não têm subprocessos e que executam em um único passo sob a perspectiva do cliente; (2) *SimpleProcess*, que representa processos que não podem ser diretamente invocados, mas que, assim como um processo atômico, são vistos como executados em apenas um único passo, podendo ser utilizados como elementos de abstração de um processo atômico ou uma representação simplificada de um processo composto; (3) *CompositeProcess,* que representa processos que podem ser decompostos em outros processos, sendo o mais complexo dos processos, e podendo ser considerado uma arvore cujos nós não-terminais são rotulados com construtores de controle e as folhas representam a invocação de outros processos.

A OWL-S inclui os seguintes construtores de controle: *Sequence*, *Split*, *Split-Join*, *Any-Order*, *Choice*, *If-Then-Else*, *Iterate*, *Repeat-While* e *Repeat-Until*. A semântica dos construtores de controle em OWL-S é similar à semântica dos respectivos construtores de controle em BPEL [LEYMANN, ROLLER, 2002].

ServiceGrounding

A informação de como acessar um serviço em OWL-S é descrita na classe *ServiceGrounding*. Esta informação diz respeito basicamente ao formato das mensagens que devem ser trocadas e aos protocolos de serialização, transporte e endereçamento.

Um *grounding* em OWL-S pode ser visto como um mapeamento de uma especificação abstrata para uma especificação concreta dos elementos que descrevem o serviço e que são necessários para interagir com o serviço. Assim, dentre os documentos que juntos formam a linguagem OWL-S, o *ServiceGrounding* é o único que lida com um nó concreto de especificação, enquanto o *ServiceProfile* e o *ServiceModel* são representações em um nó abstrato.

No entanto, OWL-S não inclui uma construção abstrata para explicitamente descrever mensagens. O que ocorre, de fato, é que o conteúdo abstrato de uma mensagem é implicitamente especificado pelas propriedades de entrada e de saída dos processos atômicos. As mensagens concretas, ao contrário, são explicitamente especificadas em um *grounding*. Para isso, é utilizada a linguagem WSDL (*Web Services Description Language*). O objetivo do *grounding* então é determinar entradas e saídas de um processo atômico que podem ser descritas em termos de mensagens trocadas entre os processos.

O conceito de *grounding* em OWL-S é geralmente consistente com o conceito de *binding* de WSDL. Porém, as duas linguagens são necessárias para a especificação concreta de um processo, porque elas não cobrem o mesmo espaço conceitual.

A1.2 Exemplo simplificado de documento OWL-S

O serviço (*Book Finder*) descrito por este documento OWL-S retorna informações de um livro com o título semelhante a uma dado parâmetro que ele recebe (*String*). O serviço realiza uma busca baseada em palavra chave para retornar como resultado o número ISBN do livro, o autor e informações de publicação.

```
<rdf:RDF xml:base="http://www.mindswap.org/2004/owl-s/1.1/BookFinder.owl">
<owl:Ontology rdf:about="">
<owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.1/Service.owl"/>
<owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.1/Profile.owl"/>
<owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.1/Process.owl"/>
<owl:imports rdf:resource="http://www.daml.org/services/owl-
s/1.1/Grounding.owl"/>
<!-- use the cached version for bibtex ontology -->
<owl:imports rdf:resource="http://www.mindswap.org/ontologies/bibtex.owl"/>
</owl:Ontology>
<!-- Service description -->
<service:Service rdf:ID="BookFinderService">
<service:presents rdf:resource="#BookFinderProfile"/>
<service:describedBy rdf:resource="#BookFinderProcess"/>
<service:supports rdf:resource="#BookFinderGrounding"/>
</service:Service>
<!-- Profile description -->
<mind:BookInformationService rdf:ID="BookFinderProfile">
<service:presentedBy rdf:resource="#BookFinderService"/>
<profile:serviceName xml:lang="en">Book Finder</profile:serviceName>
<profile:textDescription xml:lang="en">
This service returns the information of a book whose title best matches the
given string.
</profile:textDescription>
<profile:hasInput rdf:resource="#BookName"/>
<profile:hasOutput rdf:resource="#BookInfo"/>
</mind:BookInformationService>
<!-- Process description -->
<process:AtomicProcess rdf:ID="BookFinderProcess">
<service:describes rdf:resource="#BookFinderService"/>
<process:hasInput rdf:resource="#BookName"/>
<process:hasOutput rdf:resource="#BookInfo"/>
</process:AtomicProcess>
<process:Input rdf:ID="BookName">
<process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://www.w3.org/20
01/XMLSchema#string</process:parameterType>
<rdfs:label>Book Name</rdfs:label>
</process:Input>
<process:Output rdf:ID="BookInfo">
<process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://purl.org/net/
nknouf/ns/bibtex#Book</process:parameterType>
<rdfs:label>Book Info</rdfs:label>
</process:Output>
<!-- Grounding description -->
<grounding:WsdlGrounding rdf:ID="BookFinderGrounding">
<service:supportedBy rdf:resource="#BookFinderService"/>
<grounding:hasAtomicProcessGrounding
rdf:resource="#BookFinderProcessGrounding"/>
</grounding:WsdlGrounding>
<grounding:WsdlAtomicProcessGrounding rdf:ID="BookFinderProcessGrounding">
<grounding:owlsProcess rdf:resource="#BookFinderProcess"/>
<grounding:wsdlDocument
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://cheeso.members.winisp.net/books/books.asmx?WSDL
</grounding:wsdlDocument>
<grounding:wsdlOperation>
```

```
<grounding:WsdlOperationRef>
<grounding:portType rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://cheeso.members.winisp.net/books/LookyBookServiceSoap
</grounding:portType>
<grounding:operation
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://cheeso.members.winisp.net/books/DoKeywordSearch
</grounding:operation>
</grounding:WsdlOperationRef>
</grounding:wsdlOperation>
<grounding:wsdlInputMessage
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://cheeso.members.winisp.net/books/DoKeywordSearchSoapIn
</grounding:wsdlInputMessage>
<grounding:wsdlInput>
<grounding:WsdlInputMessageMap>
<grounding:owlsParameter rdf:resource="#BookName"/>
<grounding:wsdlMessagePart
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">http://cheeso.member
s.winisp.net/books/keyword</grounding:wsdlMessagePart>
</grounding:WsdlInputMessageMap>
</grounding:wsdlInput>
<grounding:wsdlOutputMessage
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://cheeso.members.winisp.net/books/DoKeywordSearchSoapOut
</grounding:wsdlOutputMessage>
<grounding:wsdlOutput>
<grounding:WsdlOutputMessageMap>
<grounding:owlsParameter rdf:resource="#BookInfo"/>
<grounding:wsdlMessagePart
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI">
http://cheeso.members.winisp.net/books/DoKeywordSearchResult
</grounding:wsdlMessagePart>
<grounding:xsltTransformationString>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:ns1="http://dinoch.dyndns.org/webservices/books">
      <xsl:output method="xml" version="1.0" encoding="UTF-8"
indent="yes"/>
      <xsl:template match="/ ">
            <xsl:variable name="pubdate"
select="ns1:DoKeywordSearchResult/ns1:bookInfo/ns1:pubdate"/>
            <xsl:variable name="month_day" select="substring-
before($pubdate,', ')"/>
            <xsl:variable name="year" select="substring-after($pubdate,',
')"/>
            <rdf:RDF
                  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
                  xmlns:bibtex="http://purl.org/net/nknouf/ns/bibtex#">

                  <bibtex:Book>
                        <bibtex:hasISBN>
                              <xsl:value-of
select="ns1:DoKeywordSearchResult/ns1:bookInfo/ns1:isbn"/>
                        </bibtex:hasISBN>
                        <bibtex:hasTitle>
                              <xsl:value-of
select="ns1:DoKeywordSearchResult/ns1:bookInfo/ns1:title"/>
                        </bibtex:hasTitle>
                        <bibtex:hasAuthor>
```

```
                              <xsl:value-of
select="ns1:DoKeywordSearchResult/ns1:bookInfo/ns1:author"/>
                        </bibtex:hasAuthor>
                        <bibtex:hasPublisher>
                              <xsl:value-of
select="ns1:DoKeywordSearchResult/ns1:bookInfo/ns1:publisher"/>
                        </bibtex:hasPublisher>
                        <bibtex:hasMonth>
                              <xsl:choose>
                                    <xsl:when test="contains($month_day,'
')">
                                          <xsl:value-of select="substring-
after($month_day,' ')"/>
                                    </xsl:when>
                                    <xsl:otherwise>
                                          <xsl:value-of
select="$month_day"/>
                                    </xsl:otherwise>
                              </xsl:choose>
                        </bibtex:hasMonth>
                        <bibtex:hasYear
rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">
                              <xsl:value-of select="$year"/>
                        </bibtex:hasYear>
                  </bibtex:Book>
            </rdf:RDF>
      </xsl:template>
</xsl:stylesheet>
</grounding:xsltTransformationString>
</grounding:WsdlOutputMessageMap>
</grounding:wsdlOutput>
</grounding:WsdlAtomicProcessGrounding>
</rdf:RDF>
```

Apêndice 2

OWL-S dos serviços utilizados para compor o *workflow* para alinhamento de sequências

A2.1 Serviço Mart

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE uridef [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY owl "http://www.w3.org/2002/07/owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY service "http://www.daml.org/services/owl-s/1.1/Service.owl">
  <!ENTITY profile "http://www.daml.org/services/owl-s/1.1/Profile.owl">
  <!ENTITY process "http://www.daml.org/services/owl-s/1.1/Process.owl">
  <!ENTITY           grounding           "http://www.daml.org/services/owl-
s/1.1/Grounding.owl">
  <!ENTITY groundingWSDL "http://www.biomart.org/biomart/martwsdl">
  <!ENTITY concepts "http://www.mindswap.org/2004/owl-s/concepts.owl">
  <!ENTITY             mind             "http://www.mindswap.org/2004/owl-
s/1.1/MindswapProfileHierarchy.owl">
  <!ENTITY bibtex "http://purl.org/net/nknouf/ns/bibtex">
]>
<rdf:RDF
  xmlns:rdf="&rdf;#"
  xmlns:rdfs="&rdfs;#"
  xmlns:owl="&owl;#"
  xmlns:xsd="&xsd;#"
  xmlns:service="&service;#"
  xmlns:profile="&profile;#"
  xmlns:process="&process;#"
  xmlns:grounding="&grounding;#"
  xmlns:bibtex="&bibtex;#"
  xmlns:mind="&mind;#"
  xml:base="http://localhost/owls/Mart.owl"
>

<owl:Ontology rdf:about="">
      <owl:imports rdf:resource="&service;"/>
      <owl:imports rdf:resource="&profile;"/>
      <owl:imports rdf:resource="&process;"/>
      <owl:imports rdf:resource="&grounding;"/>
      <owl:imports rdf:resource="&concepts;"/>
    <!-- use the cached version for bibtex ontology -->
    <owl:imports
rdf:resource="http://www.mindswap.org/ontologies/bibtex.owl"/>
 </owl:Ontology>

<!-- Service description -->
<service:Service rdf:ID="MartService">
```

```
      <service:presents rdf:resource="#MartProfile"/>
      <service:describedBy rdf:resource="#MartProcess"/>
      <service:supports rdf:resource="#MartGrounding"/>
</service:Service>

<!-- Profile description -->
<profile:Profile rdf:ID="MartProfile">
      <service:presentedBy rdf:resource="#MartService"/>
      <profile:serviceName xml:lang="en">Mart</profile:serviceName>
      <profile:textDescription xml:lang="en">Get a nucleotid sequence set
from a database.</profile:textDescription>
      <profile:hasOutput>
            <process:Output rdf:ID="Nucleotide_Sequence_Set">
                  <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

      >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
                  <process:parameterValue
rdf:parseType="Literal">Nucleotide_Sequence_Set</process:parameterValue>
            </process:Output>
        </profile:hasOutput>
      <profile:hasInput>
            <process:Input rdf:ID="Database">
                  <process:parameterValue
rdf:parseType="Literal">Database</process:parameterValue>
                  <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

      >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
            </process:Input>
        </profile:hasInput>
      <profile:hasInput>
            <process:Input rdf:ID="Dataset">
                  <process:parameterValue
rdf:parseType="Literal">Dataset</process:parameterValue>
                  <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

      >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
            </process:Input>
        </profile:hasInput>
      <profile:serviceParameter>
            <profile:ServiceParameter
rdf:ID="Read_Nucleotide_Sequence_Set">
                  <profile:serviceParameterName
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"

      >Read_Nucleotide_Sequence_Set</profile:serviceParameterName>
            </profile:ServiceParameter>
      </profile:serviceParameter>
      <profile:serviceParameter>
            <profile:ServiceParameter rdf:ID="Database_Connection">
                  <profile:serviceParameterName
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                  >Database_Connection</profile:serviceParameterName>
            </profile:ServiceParameter>
      </profile:serviceParameter>
</profile:Profile>

<!-- Process Model description -->
<process:AtomicProcess rdf:ID="MartProcess">
```

```
      <process:hasInput>
            <process:Input rdf:ID="Database_Process_Input">
                  <process:parameterValue
rdf:parseType="Literal">Database</process:parameterValue>
                  <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

      >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
            </process:Input>
      </process:hasInput>
      <process:hasInput>
            <process:Input rdf:ID="Dataset_Process_Input">
                  <process:parameterValue
rdf:parseType="Literal">Dataset</process:parameterValue>
                  <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

      >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
            </process:Input>
      </process:hasInput>
      <process:hasOutput>
            <process:Output rdf:ID="Nucleotid_Sequence_Set_Process_Output">
                  <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

      >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
                  <process:parameterValue
rdf:parseType="Literal">Nucleotid_Sequence_Set</process:parameterValue>
            </process:Output>
      </process:hasOutput>
</process:AtomicProcess>

<!-- Grounding description -->
<grounding:WsdlGrounding rdf:ID="MartGrounding">
        <service:supportedBy rdf:resource="#MartService"/>
</grounding:WsdlGrounding>

</rdf:RDF>
```

A2.2 Serviço Seqret

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE uridef [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY owl "http://www.w3.org/2002/07/owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY service "http://www.daml.org/services/owl-s/1.1/Service.owl">
  <!ENTITY profile "http://www.daml.org/services/owl-s/1.1/Profile.owl">
  <!ENTITY process "http://www.daml.org/services/owl-s/1.1/Process.owl">
  <!ENTITY           grounding           "http://www.daml.org/services/owl-
s/1.1/Grounding.owl">
  <!ENTITY                                                    groundingWSDL
"http://www.ebi.ac.uk/soaplab/emboss4/services/edit.seqret?wsdl">
  <!ENTITY concepts "http://www.mindswap.org/2004/owl-s/concepts.owl">
```

```
  <!ENTITY              mind              "http://www.mindswap.org/2004/owl-
s/1.1/MindswapProfileHierarchy.owl">
  <!ENTITY bibtex "http://purl.org/net/nknouf/ns/bibtex">
]>
<rdf:RDF
  xmlns:rdf="&rdf;#"
  xmlns:rdfs="&rdfs;#"
  xmlns:owl="&owl;#"
  xmlns:xsd="&xsd;#"
  xmlns:service="&service;#"
  xmlns:profile="&profile;#"
  xmlns:process="&process;#"
  xmlns:grounding="&grounding;#"
  xmlns:bibtex="&bibtex;#"
  xmlns:mind="&mind;#"
  xml:base="http://localhost/owls/Seqret.owl"
>

<owl:Ontology rdf:about="">
      <owl:imports rdf:resource="&service;"/>
      <owl:imports rdf:resource="&profile;"/>
      <owl:imports rdf:resource="&process;"/>
      <owl:imports rdf:resource="&grounding;"/>
      <owl:imports rdf:resource="&concepts;"/>
    <!-- use the cached version for bibtex ontology -->
    <owl:imports
rdf:resource="http://www.mindswap.org/ontologies/bibtex.owl"/>
 </owl:Ontology>

<!-- Service description -->
<service:Service rdf:ID="SeqretService">
      <service:presents rdf:resource="#SeqretProfile"/>
      <service:describedBy rdf:resource="#SeqretProcess"/>
      <service:supports rdf:resource="#SeqretGrounding"/>
</service:Service>

<!-- Profile description -->
<profile:Profile rdf:ID="SeqretProfile">
      <service:presentedBy rdf:resource="#SeqretService"/>
      <profile:serviceName xml:lang="en">Seqret</profile:serviceName>
      <profile:textDescription  xml:lang="en">Read  and  format  nucleotid
sequence set.</profile:textDescription>
      <profile:hasOutput>
            <process:Output rdf:ID="Nucleotid_Sequence_Set_Output">
                  <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

      >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
                  <process:parameterValue
rdf:parseType="Literal">Nucleotide_Sequence_Set</process:parameterValue>
            </process:Output>
        </profile:hasOutput>
      <profile:hasInput>
            <process:Input rdf:ID="Nucleotid_Sequece_Set_Input">
                  <process:parameterValue
rdf:parseType="Literal"></process:parameterValue>
                  <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

      >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
            </process:Input>
```

```
        </profile:hasInput>
      <profile:serviceParameter>
            <profile:ServiceParameter rdf:ID="FASTA">
                  <profile:serviceParameterName
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                  >FASTA</profile:serviceParameterName>
            </profile:ServiceParameter>
      </profile:serviceParameter>
</profile:Profile>

<!-- Process Model description -->
<process:AtomicProcess rdf:ID="SeqretProcess">
      <process:hasInput>
            <process:Input rdf:ID="Nucleotid_Sequence_Set_Process_Input">
                  <process:parameterValue
rdf:parseType="Literal">Nucleotid_Sequence_Set</process:parameterValue>
                  <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

      >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
            </process:Input>
      </process:hasInput>
      <process:hasOutput>
            <process:Output rdf:ID="Nucleotid_Sequence_Set_Process_Output">
                  <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

      >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
                  <process:parameterValue
rdf:parseType="Literal">FASTA</process:parameterValue>
            </process:Output>
      </process:hasOutput>
</process:AtomicProcess>

<!-- Grounding description -->
<grounding:WsdlGrounding rdf:ID="SeqretGrounding">
        <service:supportedBy rdf:resource="#SeqretService"/>
</grounding:WsdlGrounding>

</rdf:RDF>
```

A2.3 Serviço Emma

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE uridef [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY owl "http://www.w3.org/2002/07/owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY service "http://www.daml.org/services/owl-s/1.1/Service.owl">
  <!ENTITY profile "http://www.daml.org/services/owl-s/1.1/Profile.owl">
  <!ENTITY process "http://www.daml.org/services/owl-s/1.1/Process.owl">
  <!ENTITY           grounding           "http://www.daml.org/services/owl-
s/1.1/Grounding.owl">
```

```
  <!ENTITY                                                    groundingWSDL
"http://www.ebi.ac.uk/soaplab/emboss4/services/alignment_multiple.emma?wsdl
">
  <!ENTITY concepts "http://www.mindswap.org/2004/owl-s/concepts.owl">
  <!ENTITY              mind              "http://www.mindswap.org/2004/owl-
s/1.1/MindswapProfileHierarchy.owl">
  <!ENTITY bibtex "http://purl.org/net/nknouf/ns/bibtex">
]>
<rdf:RDF
  xmlns:rdf="&rdf;#"
  xmlns:rdfs="&rdfs;#"
  xmlns:owl="&owl;#"
  xmlns:xsd="&xsd;#"
  xmlns:service="&service;#"
  xmlns:profile="&profile;#"
  xmlns:process="&process;#"
  xmlns:grounding="&grounding;#"
  xmlns:bibtex="&bibtex;#"
  xmlns:mind="&mind;#"
  xml:base="http://localhost/owls/Emma.owl"
>

<owl:Ontology rdf:about="">
      <owl:imports rdf:resource="&service;"/>
      <owl:imports rdf:resource="&profile;"/>
      <owl:imports rdf:resource="&process;"/>
      <owl:imports rdf:resource="&grounding;"/>
      <owl:imports rdf:resource="&concepts;"/>
    <!-- use the cached version for bibtex ontology -->
    <owl:imports
rdf:resource="http://www.mindswap.org/ontologies/bibtex.owl"/>
 </owl:Ontology>

<!-- Service description -->
<service:Service rdf:ID="EmmaService">
      <service:presents rdf:resource="#EmmaProfile"/>
      <service:describedBy rdf:resource="#EmmaProcess"/>
      <service:supports rdf:resource="#EmmaGrounding"/>
</service:Service>

<!-- Profile description -->
<profile:Profile rdf:ID="EmmaProfile">
      <service:presentedBy rdf:resource="#EmmaService"/>
      <profile:serviceName xml:lang="en">Emma</profile:serviceName>
      <profile:textDescription  xml:lang="en">Multiple  nucleotide  sequence
alignment.</profile:textDescription>
      <profile:hasOutput>
            <process:Output rdf:ID="XML">
                  <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

      >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
                  <process:parameterValue
rdf:parseType="Literal">XML</process:parameterValue>
            </process:Output>
        </profile:hasOutput>
      <profile:hasInput>
            <process:Input rdf:ID="FASTA">
                  <process:parameterValue
rdf:parseType="Literal">FASTA</process:parameterValue>
```

```
                  <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

      >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
            </process:Input>
        </profile:hasInput>
      <profile:serviceParameter>
            <profile:ServiceParameter rdf:ID="Multiple_Alignment">
                  <profile:serviceParameterName
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                  >Multiple_Alignment</profile:serviceParameterName>
            </profile:ServiceParameter>
      </profile:serviceParameter>
      <profile:serviceParameter>
            <profile:ServiceParameter rdf:ID="BLASTP">
                  <profile:serviceParameterName
rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
                  >BLASTP</profile:serviceParameterName>
            </profile:ServiceParameter>
      </profile:serviceParameter>
</profile:Profile>

<!-- Process Model description -->
<process:AtomicProcess rdf:ID="EmmaProcess">
      <process:hasInput>
            <process:Input rdf:ID="FASTA_Process_Input">
                  <process:parameterValue
rdf:parseType="Literal">FASTA</process:parameterValue>
                  <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

      >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
            </process:Input>
      </process:hasInput>
      <process:hasOutput>
            <process:Output rdf:ID="XML_Process_Output">
                  <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

      >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
                  <process:parameterValue
rdf:parseType="Literal">XML</process:parameterValue>
            </process:Output>
      </process:hasOutput>
</process:AtomicProcess>

<!-- Grounding description -->
<grounding:WsdlGrounding rdf:ID="EmmaGrounding">
        <service:supportedBy rdf:resource="#EmmaService"/>
</grounding:WsdlGrounding>

</rdf:RDF>
```

A2.4 Serviço Plot

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE uridef [
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns">
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema">
  <!ENTITY owl "http://www.w3.org/2002/07/owl">
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema">
  <!ENTITY service "http://www.daml.org/services/owl-s/1.1/Service.owl">
  <!ENTITY profile "http://www.daml.org/services/owl-s/1.1/Profile.owl">
  <!ENTITY process "http://www.daml.org/services/owl-s/1.1/Process.owl">
  <!ENTITY           grounding           "http://www.daml.org/services/owl-
s/1.1/Grounding.owl">
  <!ENTITY                                                    groundingWSDL
"http://www.ebi.ac.uk/soaplab/emboss4/services/alignment_multiple.prettyplo
t?wsdl">
  <!ENTITY concepts "http://www.mindswap.org/2004/owl-s/concepts.owl">
  <!ENTITY             mind             "http://www.mindswap.org/2004/owl-
s/1.1/MindswapProfileHierarchy.owl">
  <!ENTITY bibtex "http://purl.org/net/nknouf/ns/bibtex">
]>
<rdf:RDF
  xmlns:rdf="&rdf;#"
  xmlns:rdfs="&rdfs;#"
  xmlns:owl="&owl;#"
  xmlns:xsd="&xsd;#"
  xmlns:service="&service;#"
  xmlns:profile="&profile;#"
  xmlns:process="&process;#"
  xmlns:grounding="&grounding;#"
  xmlns:bibtex="&bibtex;#"
  xmlns:mind="&mind;#"
  xml:base="http://localhost/owls/Plot.owl"
>

<owl:Ontology rdf:about="">
      <owl:imports rdf:resource="&service;"/>
      <owl:imports rdf:resource="&profile;"/>
      <owl:imports rdf:resource="&process;"/>
      <owl:imports rdf:resource="&grounding;"/>
      <owl:imports rdf:resource="&concepts;"/>
    <!-- use the cached version for bibtex ontology -->
    <owl:imports
rdf:resource="http://www.mindswap.org/ontologies/bibtex.owl"/>
 </owl:Ontology>

<!-- Service description -->
<service:Service rdf:ID="PlotService">
      <service:presents rdf:resource="#PlotProfile"/>
      <service:describedBy rdf:resource="#PlotProcess"/>
      <service:supports rdf:resource="#PlotGrounding"/>
</service:Service>

<!-- Profile description -->
<profile:Profile rdf:ID="PlotProfile">
      <service:presentedBy rdf:resource="#PlotService"/>
      <profile:serviceName xml:lang="en">Plot</profile:serviceName>
```

```
      <profile:textDescription   xml:lang="en">Plot   multiple   alignment
results.</profile:textDescription>
      <profile:hasOutput>
            <process:Output rdf:ID="HTML">
                  <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

      >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
                  <process:parameterValue
rdf:parseType="Literal">HTML</process:parameterValue>
            </process:Output>
        </profile:hasOutput>
      <profile:hasInput>
            <process:Input rdf:ID="XML">
                  <process:parameterValue
rdf:parseType="Literal">XML</process:parameterValue>
                  <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

      >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
            </process:Input>
        </profile:hasInput>
</profile:Profile>

<!-- Process Model description -->
<process:AtomicProcess rdf:ID="PlotProcess">
      <process:hasInput>
            <process:Input rdf:ID="XML_Process_Input">
                  <process:parameterValue
rdf:parseType="Literal">XML</process:parameterValue>
                  <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

      >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
            </process:Input>
      </process:hasInput>
      <process:hasOutput>
            <process:Output rdf:ID="HTML_Process_Output">
                  <process:parameterType
rdf:datatype="http://www.w3.org/2001/XMLSchema#anyURI"

      >http://www.w3.org/2001/XMLSchema#string</process:parameterType>
                  <process:parameterValue
rdf:parseType="Literal">HTML</process:parameterValue>
            </process:Output>
      </process:hasOutput>
</process:AtomicProcess>

<!-- Grounding description -->
<grounding:WsdlGrounding rdf:ID="PlotGrounding">
        <service:supportedBy rdf:resource="#PlotService"/>
</grounding:WsdlGrounding>

</rdf:RDF>
```

Apêndice 3

OWL da ontologia de [LEMOS, 2004] e da ontologia WS-BPELOnto

A3.1 OWL da ontologia de [Lemos, 2004]

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.puc-rio.br/melissa/Bio.owl#"
  xml:base="http://www.puc-rio.br/melissa/Bio.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="FASTX3">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="FAST"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Pattern_Discovery">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Constructive"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="MultiIAlign">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Multiple_Alignment"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Smith_Waterman">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Alignment"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Database_Connection">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Connection"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Process">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Bio"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Gene_Prediction">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Constructive"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Genome_Nucleotide_Sequence_Set">
    <rdfs:subClassOf>
```

```
    <owl:Class rdf:ID="Nucleotid_Sequence_Set"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Molecular_Prediction">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Constructive"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Aminoacid_Sequence_Set">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Sequence_Set"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Container_Connection">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Connection"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Container">
  <rdfs:subClassOf rdf:resource="#Bio"/>
</owl:Class>
<owl:Class rdf:about="#Constructive">
  <rdfs:subClassOf rdf:resource="#Process"/>
</owl:Class>
<owl:Class rdf:ID="Stop_Point">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="External_Control"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="XML">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="View"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Verification_Point">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#External_Control"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Format_Transformation_Process">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Internal_Control"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Contig_Nucleotide_Sequence_Set">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Nucleotid_Sequence_Set"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Sequence_Assembly">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Sequence_Alignment"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Base_Identification">
  <rdfs:subClassOf rdf:resource="#Constructive"/>
</owl:Class>
<owl:Class rdf:about="#Sequence_Set">
  <rdfs:subClassOf rdf:resource="#Container"/>
</owl:Class>
<owl:Class rdf:ID="CLUSTAL_W">
```

```
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Multiple_Alignment"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="ORF_Nucleotide_Sequence_Set">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Nucleotid_Sequence_Set"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="EST">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Project"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Project">
  <rdfs:subClassOf rdf:resource="#Bio"/>
</owl:Class>
<owl:Class rdf:about="#Nucleotid_Sequence_Set">
  <rdfs:subClassOf rdf:resource="#Sequence_Set"/>
</owl:Class>
<owl:Class rdf:ID="Read_Nucleotide_Sequence_Set">
  <rdfs:subClassOf rdf:resource="#Nucleotid_Sequence_Set"/>
</owl:Class>
<owl:Class rdf:ID="BLASTX">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="BLAST"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="BLASTN">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#BLAST"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Alignment">
  <rdfs:subClassOf rdf:resource="#Bio"/>
</owl:Class>
<owl:Class rdf:ID="Complete">
  <rdfs:subClassOf rdf:resource="#Project"/>
</owl:Class>
<owl:Class rdf:about="#Connection">
  <rdfs:subClassOf rdf:resource="#Bio"/>
</owl:Class>
<owl:Class rdf:ID="HTML">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#View"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="FASTA">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#FAST"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Chromatogram_Set">
  <rdfs:subClassOf rdf:resource="#Container"/>
</owl:Class>
<owl:Class rdf:ID="ssearch">
  <rdfs:subClassOf rdf:resource="#Smith_Waterman"/>
</owl:Class>
<owl:Class rdf:ID="TBLASTN">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#BLAST"/>
```

```
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="Inspection_Point">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#Internal_Control"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#View">
  <rdfs:subClassOf rdf:resource="#Bio"/>
</owl:Class>
<owl:Class rdf:about="#Sequence_Alignment">
  <rdfs:subClassOf rdf:resource="#Constructive"/>
</owl:Class>
<owl:Class rdf:ID="Pattern_Set">
  <rdfs:subClassOf rdf:resource="#Container"/>
</owl:Class>
<owl:Class rdf:ID="Pattern_Recognition">
  <rdfs:subClassOf rdf:resource="#Constructive"/>
</owl:Class>
<owl:Class rdf:about="#FAST">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Local_Alignment"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="BLASTP">
  <rdfs:subClassOf>
    <owl:Class rdf:about="#BLAST"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:about="#Internal_Control">
  <rdfs:subClassOf rdf:resource="#Process"/>
</owl:Class>
<owl:Class rdf:ID="Filter">
  <rdfs:subClassOf rdf:resource="#Process"/>
</owl:Class>
<owl:Class rdf:about="#BLAST">
  <rdfs:subClassOf rdf:resource="#Alignment"/>
</owl:Class>
<owl:Class rdf:about="#Multiple_Alignment">
  <rdfs:subClassOf rdf:resource="#Sequence_Alignment"/>
</owl:Class>
<owl:Class rdf:ID="Pairwise_Alignment">
  <rdfs:subClassOf rdf:resource="#Sequence_Alignment"/>
</owl:Class>
<owl:Class rdf:about="#Local_Alignment">
  <rdfs:subClassOf rdf:resource="#Pairwise_Alignment"/>
</owl:Class>
<owl:Class rdf:ID="Genome_Comparison">
  <rdfs:subClassOf rdf:resource="#Constructive"/>
</owl:Class>
<owl:Class rdf:ID="Phylogenetic_Analysis">
  <rdfs:subClassOf rdf:resource="#Constructive"/>
</owl:Class>
<owl:Class rdf:about="#External_Control">
  <rdfs:subClassOf rdf:resource="#Process"/>
</owl:Class>
<owl:Class rdf:ID="TBLASTX">
  <rdfs:subClassOf rdf:resource="#BLAST"/>
</owl:Class>
<owl:Class rdf:ID="Global_Alignment">
  <rdfs:subClassOf rdf:resource="#Pairwise_Alignment"/>
```

```
  </owl:Class>
</rdf:RDF>
```

A3.2 OWL da ontologia WS-BPELOnto

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns="http://www.npqs.ufjf.br/WSBPELOnto.owl#"
  xml:base="http://www.npqs.ufjf.br/WSBPELOnto.owl">
  <owl:Ontology rdf:about=""/>
  <owl:Class rdf:ID="Function">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="AutomatedActivity"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="InputVariable">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Variable"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="InputProcessVariable">
    <rdfs:subClassOf rdf:resource="#InputVariable"/>
  </owl:Class>
  <owl:Class rdf:about="#AutomatedActivity">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Activity"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="OutputProcessVariable">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="OutputVariable"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="CientificWorkflow">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Workflow"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="BusinessProcess">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Workflow"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Step">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="PartnerLink"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#Workflow">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Process"/>
```

```
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="BusinessWorkflow">
    <rdfs:subClassOf rdf:resource="#Workflow"/>
  </owl:Class>
  <owl:Class rdf:ID="WebService">
    <rdfs:subClassOf rdf:resource="#AutomatedActivity"/>
  </owl:Class>
  <owl:Class rdf:ID="OutputPartnerLinkVariable">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#OutputVariable"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Service">
    <rdfs:subClassOf rdf:resource="#AutomatedActivity"/>
  </owl:Class>
  <owl:Class rdf:ID="ManualActivity">
    <rdfs:subClassOf>
      <owl:Class rdf:about="#Activity"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="#Process">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="WSBPEL"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="InputPartnerLinkVariable">
    <rdfs:subClassOf rdf:resource="#InputVariable"/>
  </owl:Class>
  <owl:Class rdf:about="#PartnerLink">
    <rdfs:subClassOf rdf:resource="#WSBPEL"/>
  </owl:Class>
  <owl:Class rdf:ID="Task">
    <rdfs:subClassOf rdf:resource="#PartnerLink"/>
  </owl:Class>
  <owl:Class rdf:ID="CientificProcess">
    <rdfs:subClassOf rdf:resource="#Workflow"/>
  </owl:Class>
  <owl:Class rdf:about="#Variable">
    <rdfs:subClassOf rdf:resource="#WSBPEL"/>
  </owl:Class>
  <owl:Class rdf:about="#OutputVariable">
    <rdfs:subClassOf rdf:resource="#Variable"/>
  </owl:Class>
  <owl:Class rdf:ID="InvokedApplication">
    <rdfs:subClassOf rdf:resource="#AutomatedActivity"/>
  </owl:Class>
  <owl:Class rdf:about="#Activity">
    <rdfs:subClassOf rdf:resource="#PartnerLink"/>
  </owl:Class>
</rdf:RDF>
```

Apêndice 4

WS-BPEL

A4.1 WS-BPEL do *workflow* para alinhamento de sequências

```
<?xml version="1.0" encoding="UTF-8"?>
<process xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
 xmlns:tns="http://localhost/bpel/Alignment_1.bpel"
 xmlns:ns0="http://localhost/martservice.wsdl"
 xmlns:ns1="http://localhost/seqret.wsdl"
 xmlns:ns2="http://localhost/emma.wsdl"
 xmlns:ns3="http://localhost/prettyplot.wsdl"
 name="Alignment_1"
 targetNamespace="http://localhost/bpel/Alignment_1.bpel">
  <import xmlns="" namespace="http://localhost/martservice.wsdl"
location="http://localhost/martservice.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/" />
  <import xmlns="" namespace="http://localhost/seqret.wsdl"
location="http://localhost/seqret.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/" />
  <import xmlns="" namespace="http://localhost/emma.wsdl"
location="http://localhost/emma.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/" />
  <import xmlns="" namespace="http://localhost/prettyplot.wsdl"
location="http://localhost/prettyplot.wsdl"
importType="http://schemas.xmlsoap.org/wsdl/" />
  <partnerLinks xmlns="">
    <partnerLink name="martservicePartnerLink"
partnerLinkType="ns0:martservicePartnerLinkType"
partnerRole="martserviceRole"/>
    <partnerLink name="seqretPartnerLink"
partnerLinkType="ns0:seqretPartnerLinkType" partnerRole="seqretRole"/>
    <partnerLink name="emmaPartnerLink"
partnerLinkType="ns0:emmaPartnerLinkType" partnerRole="emmaRole"/>
    <partnerLink name="prettyplotPartnerLink"
partnerLinkType="ns0:prettyplotPartnerLinkType"
partnerRole="prettyplotRole"/>
  </partnerLinks>
  <variables xmlns="">
    <variable name="database" messageType="ns0:databaseRequest" />
    <variable name="dataset" messageType="ns0:datasetRequest" />
    <variable name="sequence" messageType="ns1:sequenceRequest" />
    <variable name="fasta" messageType="ns2:fastaRequest" />
    <variable name="xml" messageType="ns3:xmlRequest" />
    <variable name="sequence" messageType="ns0:sequenceResponse" />
    <variable name="fasta" messageType="ns1:fastaResponse" />
    <variable name="xml" messageType="ns2:xmlResponse" />
    <variable name="html" messageType="ns3:htmlResponse" />
  </variables>
  <sequence>
    <flow>
```

```
      <receive name="martservice" partnerLink="martservicePartnerLink"
operation="martservice" portType="ns0:martservicePortType"
variable="database" variable="dataset"/>
      <invoke name="martservice" partnerLink="martservicePartnerLink"
operation="martservice"
portType="ns0:martservicePortType"
inputVariable="database" inputVariable="dataset"
outputVariable="sequence"/>
      <receive name="martservice" partnerLink="martservicePartnerLink"
operation="martservice" portType="ns0:martservicePortType"
inputVariable="database" inputVariable="dataset"
outputVariable="sequence"/>
      <invoke name="martservice" partnerLink="martservicePartnerLink"
operation="martservice"
portType="ns0:martservicePortType"
inputVariable="database" inputVariable="dataset"
outputVariable="sequence"/>
      <receive name="martservice" partnerLink="martservicePartnerLink"
operation="martservice" portType="ns0:martservicePortType"
inputVariable="database" inputVariable="dataset"
outputVariable="sequence"/>
      <invoke name="martservice" partnerLink="martservicePartnerLink"
operation="martservice"
portType="ns0:martservicePortType"
inputVariable="database" inputVariable="dataset"
outputVariable="sequence"/>
    </flow>
    <invoke name="seqret" partnerLink="seqretPartnerLink"
operation="seqret"
portType="ns1:seqretPortType"
inputVariable="sequence" outputVariable="fasta"/>
    <invoke name="emma" partnerLink="emmaPartnerLink" operation="emma"
portType="ns2:emmaPortType"
inputVariable="fasta" outputVariable="xml"/>
    <invoke name="prettyplot" partnerLink="prettyplotPartnerLink"
operation="prettyplot"
portType="ns3:prettyplotPortType"
inputVariable="xml" outputVariable="html"/>
    <reply name="prettyplot" partnerLink="prettyplotPartnerLink"
operation="prettyplot"
portType="ns3:prettyplotPortType"
variable="html"/>
  </sequence>
</process>
```