UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Bruno Fernandes Bastos

Uma Abordagem Baseada em Padrões para o Intercâmbio entre Especificações de Workflows Científicos

Juiz de Fora

2015

A minha família e amigos por

todo o apoio.

AGRADECIMENTOS

Aos meus pais, Edison Bastos e Maria Cristina Fernandes pelo incentivo de sempre estudar e lutar pelos meus objetivos.

A minha namorada Juliana Ermel por ter me acompanhando de perto, sempre com opiniões construtivas que me levaram a melhorar em vários sentidos.

Aos amigos que fiz durante o mestrado, Vitor Freitas, Jacimar Tavares, Eduardo Costa, Crystiam Kelle, Gustavo Henrique e Vinícius Brum, pelas conversas e pelos momentos de descontração.

Aos amigos de trabalho, Bruno Correa, Iuri Malinoski e Vinícius Macedo, por tornarem um ambiente de produção ainda mais agradável.

Aos meus orientadores Antônio Tadeu Gomes e Regina Braga por todo o apoio prestado durante esse período.

"Nem todos que sonharam
conseguiram, mas pra conseguir
é preciso sonhar."
Gabriel o Pensador

RESUMO

Workflows científicos vêm sendo utilizados para resolver problemas complexos em diferentes áreas. Sistemas Gerenciadores de Workflows Científicos (SGWfCs) são utilizados para a especificação e gerenciamento desses workflows. Porém, cada SGWfC pode possuir características diferentes e uma linguagem de especificação de workflows própria, dificultando o reuso dos workflows entre diferentes SGWfCs. A inexistência de uma padronização semântica dificulta ainda mais esse reuso, uma vez que elementos de modelagem de workflow presentes em alguns SGWfCS podem não ser mapeáveis em outros SGWfCs. O uso de uma linguagem intermediária para o intercâmbio de workflows científicos facilita o reuso de workflows desenvolvidos em diferentes SGWfCs ao permitir a definição de um arcabouço comum para esses SGWfCs. No entanto, uma linguagem desse tipo não impede que haja perda de informação semântica durante um processo de transformação de especificações entre esses SGWfCs, uma vez que essa linguagem deve ser robusta o suficiente para representar a semântica de diversos workflows desenvolvidos em diferentes SGWfCs. A existência de padrões (*patterns*) em workflows científicos pode ajudar a explicitar as informações semânticas mais importantes para a construção desses workflows. Assim a proposta deste trabalho é oferecer uma abordagem baseada em padrões para o intercâmbio entre especificações de workflows científicos, empregando uma linguagem intermediária com suporte a informações semânticas obtidas através da descrição dos padrões. Esta dissertação analisa os resultados obtidos com essa proposta a partir da aplicação da abordagem em especificações de workflows armazenadas no repositório myExperiment.

Palavras-chave: Workflows Científicos, Linguagem Intermediária, Intercâmbio de Workflows.

ABSTRACT

Scientific workflows have been used to solve complex problems in different areas. Scientific Workflow Management Systems (SWfMSs) are used for specifying and managing these workflows. Nevertheless, each SWfMS may have different characteristics and its own workflow specification language, making its reuse accross different SWfMSs a difficult process. The lack of semantic standardization makes this reuse even more difficult, since the workflow modeling elements available in some SWfMSs may not be mapped onto others SWfMSs. The use of an intermediate language for the interchange of scientific workflows may help with the reuse of workflows developed in different SWfMSs, as it allows for the definition of a common framework for these SWfMSs. Nonetheless, such a language does not prevent the loss of some semantic information during a specification transformation process between different SWfMSs, since this language must be robust enough to represent the semantics of diverse workflows developed in different SWfMSs. The identification of scientific workflow patterns may help to describe the most important semantic information for the construction of these workflows. Thus, the purpose of this study is to provide a pattern-based approach for the interchange of scientific workflow specifications, using an intermediate language that supports semantic information obtained through the description of workflow patterns. This thesis also analyses the results obtained with the proposed approach being applied to workflow specifications stored in the myExperiment repository.

Keywords: Scientific Workflows, Intermediate Language, Workflow Interchange.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

H_T	Hipótese de intercâmbio **total** de um workflow científico
H_E	Hipótese de intercâmbio **estrutural** de um workflow científico

LISTA DE ABREVIATURAS E SIGLAS

ADL Architecture Description Language

API Application Programming Interface

ATL ATL Transformation Language

BPMN Business Process Modeling Notation

EMF Eclipse Modeling Framework

IWIR Interoperable Workflow Intermediate Representation

OPM Open Provenance Model

OSC Open Scientific Connectors

REST REpresentational State Transfer

SGWfC Sistema Gerenciador de Workflow Científico

SOAP Simple Object Access Protocol

UML Unified Modeling Language

WISP Workflow Interchange in Science with Patterns

XML Extensible Markup Language

XSLT Extensible Stylesheet Language Transformations

SUMÁRIO

1 INTRODUÇÃO

Workflows científicos são utilizados para resolver problemas complexos em diferentes áreas de *e-science*, como por exemplo Bioinformática (EMEAKAROHA et al., 2011), Astronomia (FREUDLING et al., 2013) e Física (ALTINTAS et al., 2006). De forma mais abstrata, um workflow (WFMC, 1999) pode ser descrito como um conjunto de tarefas interligadas através de conexões representando relações de dependência entre as tarefas. Em workflows científicos, uma tarefa representa um processo computacional com portas de entrada e de saída, e a dependência entre duas tarefas é tipicamente expressa em termos dos dados gerados por uma porta de saída de uma das tarefas e que são necessários para a execução da outra tarefa, sendo repassados a esta através de uma de suas portas de entrada.

Historicamente, workflows científicos têm sido implementados como scripts cuja função é executar diferentes tarefas segundo uma ordem estabelecida pelas relações de dependência entre essas tarefas. Contudo, o uso de scripts para implementar workflows científicos traz uma série de dificuldades, algumas de ordem puramente técnica e outras de natureza metodológica. No aspecto técnico, em primeiro lugar, os dados utilizados no script (parâmetros, arquivos de entrada) muitas vezes não acompanham o próprio. Esses dados podem, por exemplo, estar armazenados localmente na máquina de um cientista e, portanto, não disponíveis para execução do workflow em outra máquina ou em um ambiente distribuído. Outro problema técnico refere-se à portabilidade do script, pois muitas funções utilizadas em um script podem não ser nativas em um sistema operacional, e a necessidade de instalação dessas dependências pode não ser trivial para um cientista não familiarizado com computação. No âmbito metodológico, o problema fundamental dos scripts está relacionado à gestão dos experimentos conduzidos com os mesmos (SILVA, 2006). Em primeiro lugar, o resultado de um experimento sempre é relevante, independente de seu "sucesso", e essa noção não é trivial de ser implementada em um script. Da mesma forma, resultados intermediários também são relevantes e a sua organização conjuntamente aos resultados finais gerados e aos dados consumidos pelos scripts é usualmente relegada ao usuário do script, tipicamente na forma de estruturas complexas de diretórios de arquivos ou de bases de dados com esquemas desenhados de forma *ad hoc*.

Para mitigar dificuldades como as citadas acima no uso de scripts, alguns trabalhos como noWorkflow [1] e CDE [2] propõem soluções para encapsular dependências utilizadas nesses scripts de maneira que estes possam ser executados em ambientes diferentes dos quais foram desenvolvidos. Outros trabalhos, que são o foco desta dissertação, propõem o desenvolvimento de Sistemas Gerenciadores de Workflows Científicos (SGWfCs). Como exemplos desses sistemas, podemos destacar Taverna (OINN et al., 2004), Kepler (ALTINTAS et al., 2004) e VisTrails (CALLAHAN et al., 2006). Esses sistemas têm como propósito auxiliar cientistas na criação de workflows científicos, assim como facilitar a execução e a gestão dos dados consumidos e gerados a partir desses workflows. Um SGWfC geralmente possui uma interface visual, onde o cientista pode facilmente definir tarefas e conexões entre essas tarefas. Vários SGWfCs também possuem uma linguagem própria de especificação de workflows, em geral declarativa e, portanto, diferente das linguagens imperativas associadas a scripts.[3] Uma especificação de workflow científico nessas linguagens prescreve um modelo da estrutura das tarefas (suas portas e propriedades) a serem envolvidas no workflow, da orquestração da execução dessas tarefas e das relações de dependência entre essas tarefas, segundo algum formalismo particular, por exemplo, redes de Petri (utilizadas no sistema Grid-Flow (GUAN et al., 2006)) e UML (utilizada no sistema Askalon (FAHRINGER et al., 2007)). Uma especificação de workflow científico tem, portanto, papel semelhante ao de uma linguagem de programação de propósito específico e pode ser usada no intercâmbio de especificações entre usuários de diferentes implantações de um mesmo SGWfC.

1.1 MOTIVAÇÃO

O fato de cada SGWfC possuir funcionalidades específicas – e, consequentemente, uma linguagem própria para a especificação dos workflows – representa um desafio no que se refere ao reuso de workflows científicos especificados em **diferentes SGWfCs**. O reuso de workflows, no entanto, adquire importância crescente no cenário de experimentos científicos, uma vez que possibilita a integração entre diferentes grupos de pesquisa e reforça a ideia dos chamados "laboratórios colaborativos" (OLSON, 2009).

[1] `https://github.com/gems-uff/noworkflow`

[2] `http://www.pgbovine.net/cde.html`

[3] Há exceções, como no caso do SGWfC Swift (ZHAO et al., 2007), cuja linguagem SwiftScript é imperativa.

Soluções para o intercâmbio de especificações de workflows científicos desenvolvidos em diferentes SGWfCs, como as propostas por Karasavvas et al. (2012), Plankensteiner et al. (2013) e Abouelhoda et al. (2012), possibilitam um maior nível de reuso de workflows científicos desenvolvidos por diferentes grupos de pesquisa. Essas soluções são, contudo, limitadas no que se refere a: (i) adaptabilidade a diferentes SGWfCs, ou (ii) necessidade de um subsistema de execução comum às diferentes linguagens de especificação. As limitações apresentadas nesses trabalhos advêm da complexidade decorrente dos diferentes tipos de informação presentes em uma especificação de workflow. Fundamentalmente, a cada diferente tipo de informação está associada uma semântica particular e, em alguns cenários, um mapeamento imediato dessa semântica entre diferentes SGWfCs não é possível. Dois cenários simples que ilustram essa dificuldade são o de funcionalidades em um SGWfC específico que: (i) são desnecessárias em outro SGWfC (por exemplo, para formatação de dados), ou (ii) não são diretamente mapeáveis em outro SGWfC (por exemplo, suporte a proveniência). Com o número crescente de SGWfCs disponíveis e a heterogeneidade entre eles, as soluções de intercâmbio de especificações de workflows precisam ser também mais adaptáveis – no sentido de facilitarem a inclusão de novos SGWfCs.

A fim de ilustrar concretamente o desafio apontado acima, a Figura 1.1 apresenta um workflow especificado no SGWfC Taverna (OINN et al., 2004) e disponibilizado no repositório myExperiment.[4] Nessa especificação sete tarefas (runInterProScan, checkStatus, Get_text_result, Get_xml_result, Unpack_text_result, Unpack_XML_result e Format_as_-

[4] `http://www.myexperiment.org/workflows/820.html`

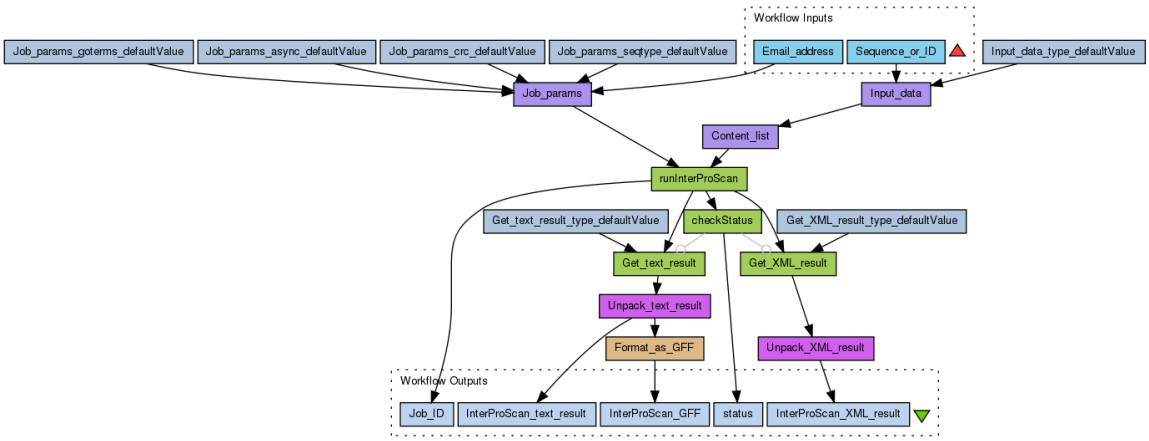


Figura 1.1: Workflow Taverna obtido do repositório myExperiment

GFF) carregam as informações semânticas mais importantes do workflow, enquanto as outras tarefas são valores constantes, representações de portas de entrada ou saída do workflow, ou tarefas auxiliares de serviços Web [5], utilizadas para formatar suas entradas e/ou e saídas . Capturar automaticamente a informação semântica mais importante desse workflow e transformá-la na forma de representação de um outro SGWfC é um exemplo desse desafio.

Nas seções que se seguem o desafio apresentado acima e a proposta de solução deste trabalho são mais precisamente delineados.

1.2 PROBLEMA

O problema de pesquisa original levantado nesta dissertação pode ser formulado da seguinte forma:

> Não há soluções no estado-da-arte suficientemente abrangentes para permitir a um cientista executar ou mesmo manipular workflows científicos desenvolvidos em SGWfCs diferentes daquele(s) que lhe é(são) mais familiar(es), bem como suficientemente adaptáveis para facilitar a inclusão de novos SGWfCs.

Para contribuir na obtenção de uma solução para o problema posto, é necessário identificar uma forma de representar workflows científicos desenvolvidos em diferentes SGWfCs de maneira que sua informação semântica seja preservada.

Devido a grande heterogeneidade de SGWfCs, alguns complicadores para o problema descrito podem ser apontados:

1. A ausência de um arcabouço semântico comum entre SGWfCs;
2. A existência de funcionalidades em um SGWfC que podem não ser nativamente mapeáveis para outros SGWfCs; e
3. O desconhecimento do cientista acerca de todas as funcionalidades existentes em um SGWfC (o que pode levar o cientista a implementar a semântica desejada em seu workflow de maneira mais complexa do que a necessária).

[5] O termo "tarefas auxiliares", no contexto dessa dissertação, é utilizado para referenciar tarefas que são criadas especificamente para tratar as entradas e saídas de tarefas do tipo "serviços Web".

1.3 HIPÓTESES

As hipóteses exploradas nesta pesquisa estão relacionadas ao potencial benefício que uma solução para o problema descrito traria em termos de uma maior integração de diferentes grupos de pesquisa cujos objetivos possuam alguma interseção.

Neste contexto, uma estratégia baseada no uso de padrões (*patterns*) em workflows científicos (YILDIZ et al., 2009) combinados a conceitos de arquitetura de software (SHAW, 1990) pode ajudar a explicitar as informações semânticas mais importantes para a construção desses workflows. Segundo essa estratégia, uma vez que fosse identificada a existência de um determinado padrão em uma especificação de workflow científico, a informação semântica associada poderia ser integral e automaticamente capturada por meio de componentes e conectores, elementos de modelagem típicos de arquitetura de software.

Considerando o cenário exposto acima, formulou-se originalmente a seguinte hipótese:

> H_T : O uso de padrões (*patterns*) combinados a conceitos de arquitetura de software para representar as informações semânticas mais importantes contidas nos workflows científicos permite auxiliar no estabelecimento de processos automáticos de transformação de especificações desses workflows entre diferentes SGWfCs, o que possibilita reduzir, em comparação com um intercâmbio manual, o esforço de um cientista para reutilizar workflows científicos desenvolvidos por outros grupos de pesquisa em SGWfCs que não fazem parte de seu ambiente de trabalho.

Entretanto, como poderá ser observado ao longo deste trabalho, esse cenário ideal é contraproducente, haja vista o enorme esforço de desenvolvimento demandado na implementação dos processos automáticos de transformação de especificações toda vez que um novo SGWfC precisa ser contemplado no processo. Uma parte importante deste esforço, como será comprovado nesta dissertação, está relacionado à grande heterogeneidade na representação e implementação interna das tarefas em cada SGWfC.

Como apontado anteriormente, há de se considerar o interesse neste trabalho de se oferecer soluções adaptáveis à inclusão de novos SGWfCs no processo. Sendo assim, o estudo e experimentação conduzidos ao longo da elaboração desta dissertação levaram à formulação de uma hipótese mais fraca, mas que se comprovada verdadeira é de resultado prático potencialmente útil:

H_E : O uso de padrões (*patterns*) combinados a conceitos de arquitetura de software para representar as informações semânticas mais importantes contidas **na estrutura** dos workflows científicos permite auxiliar no estabelecimento de processos **semi-automáticos** de transformação de especificações desses workflows entre diferentes SGWfCs, o que possibilita reduzir, em comparação com um intercâmbio manual, o esforço de um cientista para reutilizar workflows científicos **complexos** (no sentido de terem um grande número de tarefas e relacionamentos entre elas) desenvolvidos por outros grupos de pesquisa em SGWfCs que não fazem parte de seu ambiente de trabalho.

1.4 OBJETIVOS

Para validar a hipótese H_E pontuada acima, este trabalho tem por objetivo *especificar uma linguagem para o intercâmbio de estruturas de workflows científicos baseada em padrões, bem como desenvolver o ferramental computacional necessário para transformar estruturas de workflows especificados em diferentes SGWfCs de e para essa linguagem.*

De acordo com a hipótese H_E apresentada, uma linguagem de intercâmbio baseada em padrões ofereceria mais segurança em preservar a semântica associada à estrutura de um workflow (o que chamaremos neste trabalho de **semântica estrutural**) cuja especificação foi gerada a partir de um processo de transformação de uma especificação feita em qualquer outro SGWfC. Mais ainda, acredita-se que a utilidade da linguagem aumente de forma proporcional ao tamanho das especificações de workflow reutilizadas (em termos de número de tarefas e relacionamentos entre elas). Mesmo para os casos em que determinados aspectos comportamentais de uma especificação não possam ser completamente cobertos pela linguagem e pelo ferramental associado (p.ex. em função de estarem encapsulados na codificação interna de uma tarefa), a reprodução integral da estrutura de um workflow complexo possibilita um primeiro passo importante para reduzir consideravelmente o esforço de reuso dessa especificação.

1.5 ESTRUTURA DO TRABALHO

Este trabalho está organizado em cinco capítulos, conforme ilustrado na Figura 1.2. Além deste capítulo introdutório que apresenta a motivação, problema, hipóteses e objetivos do

trabalho, os capítulos restantes são apresentados como se segue:

- O Capítulo 2 apresenta a fundamentação teórica do trabalho, como o conceito de workflows científicos e suas áreas de uso, assim como exemplos de SGWfCs, padrões em workflows científicos, transformações em especificações de workflows científicos e trabalhos relacionados.

- O Capítulo 3 apresenta a proposta deste trabalho, descrevendo a linguagem de intercâmbio e o mecanismo de transformações proposto com detalhes de implementação, além de alguns casos de uso baseados em workflows disponíveis em repositórios públicos na web.

- O Capítulo 4 apresenta métricas utilizadas para validar a linguagem e o mecanismo de transformações proposto, bem como experimentos realizados para a avaliação da proposta segundo essas métricas.

- O Capítulo 5 apresenta as considerações finais bem como sugestões de trabalhos futuros.

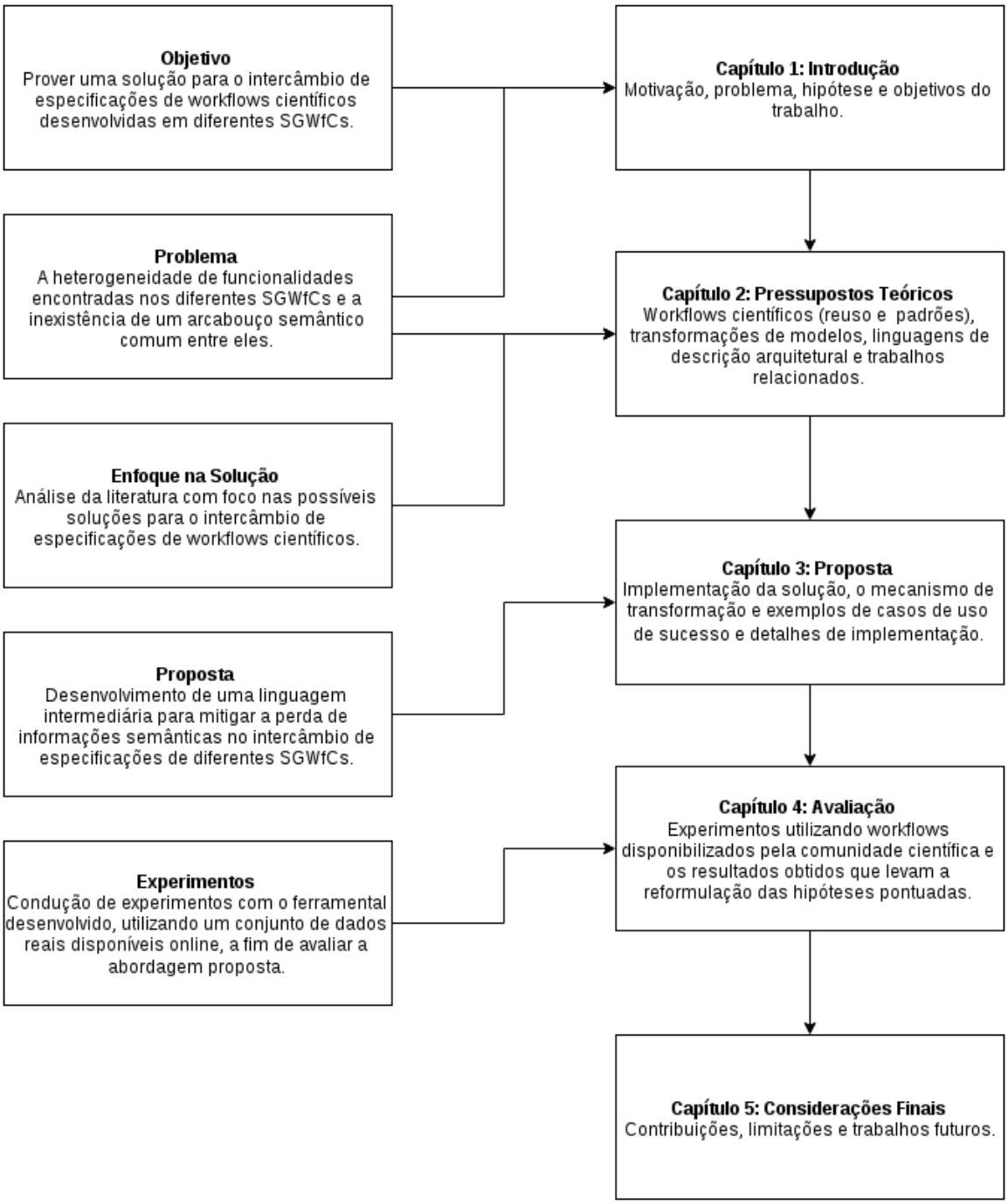


Figura 1.2: Estrutura da Dissertação

2 FUNDAMENTAÇÃO TEÓRICA

2.1 WORKFLOWS CIENTÍFICOS

Um workflow científico pode ser descrito como um conjunto de tarefas (processos computacionais) com portas de entrada e saída interligadas através de conexões. Essas conexões podem representar relações de dependência de controle (em que uma tarefa só pode iniciar sua execução ao término de uma outra tarefa) ou relações de dependência de geração (em portas de saída) e consumo (em portas de entrada) de dados entre as tarefas. Para os objetivos desta dissertação, a não ser quando dito em contrário, as conexões sempre representarão relações de dependência de dados.

Diversos SGWfCs têm sido desenvolvidos com o propósito de auxiliar cientistas na criação de workflows científicos, assim como facilitar a execução e a gestão dos dados consumidos e gerados a partir desses workflows. A Tabela 2.1 enumera alguns SGWfCs disponíveis para a comunidade científica.[1]

Anduril	ASKALON	Apache Airavata	BioBIKE
Bioclipse	DiscoveryNet	Ergatis	Galaxy
Kepler	KNIME	Mobyle	OnlineHPC
OpenMOLE	Orange	Pegasus	P-GRADE
Pipeline Pilot	Ptolemy II	Swift	Taverna
Triana	VisTrails	Yabi	YAWL

Tabela 2.1: Alguns Sistemas Gerenciadores de Workflow Científicos (Jan/15)

Uma decisão importante no projeto de um SGWfC é a dos tipos de tarefa aos quais o SGWfC dá suporte. Tarefas do tipo serviços Web (*Web Services*) frequentemente recebem suporte de SGWfCs, sendo bastante utilizados, uma vez que não dependem de linguagem e/ou sistema operacional, trabalham sob um mesmo protocolo (SOAP) e estão disponíveis para consumo imediato via Internet. Contudo, Wassink et al. (2009) fazem uma análise dos tipos de tarefas utilizadas em um total de 415 workflows do SGWfC Taverna e identificam que o uso de tipos de tarefas proprietários do SGWfC são muito comuns, constituindo 57% das tarefas, enquanto serviços Web constituem 22% e scripts desenvolvidos por cientistas[2]

[1] No Apêndice B.1 são apresentados os *websites* onde esses SGWfCs podem ser obtidos.

[2] Nos SGWfCs o suporte a scripts geralmente é restrito a formatos específicos. Por exemplo, no Taverna é oferecido o tipo *BeanShell*, que encapsula scripts desenvolvidos na linguagem Java.

constituem 14% das tarefas. Dito isso, é possível identificar que os tipos proprietários fornecidos pelo SGWfC são de grande importância para os workflows e podem determinar a escolha de um determinado SGWfC por um usuário ou grupo de pesquisa.

2.2 REUSO DE WORKFLOWS CIENTÍFICOS

Ainda que o intercâmbio de especificações de workflows científicos entre usuários de um mesmo SGWfC seja possível e facilitado por esses sistemas, o reuso dessas especificações **entre diferentes SGWfCs** ainda é um tópico a ser devidamente explorado na literatura (GIL et al., 2007; OGASAWARA et al., 2009). Os cientistas, mesmo que de uma mesma área do conhecimento, acabam empregando SGWfCs distintos, por exemplo, em função de atributos de qualidade de maior ou menor interesse providos por cada SGWfC (por exemplo, usabilidade, tolerância a falhas, suporte a proveniência, etc). Por sua vez, as linguagens de especificação, quando fornecidas por esses SGWfCs, não são nativamente intercambiáveis, devido a uma série de razões descritas abaixo.

Do ponto de vista de reuso, os chamados workflows de negócio estão mais consolidados do que os workflows científicos. As diferenças entre workflows de negócio e workflows científicos começam pela motivação para seu desenvolvimento. Workflows de negócio possuem como usuário o setor empresarial. Analisando a proposta dos Sistemas Gerenciadores de Workflows de Negócio (SGWfNs), é fácil identificar que o reuso de workflows de negócio é algo interessante não só do ponto de vista dos empresários, como também dos desenvolvedores dos SGWfNs em si, que podem angariar novos clientes que desejam modificar seu ambiente de desenvolvimento de workflows.

Nesse contexto, a norma[3] Wf-XML (SWENSON et al., 2004) foi proposta pela *Workflow Management Coalition*[4] (WfMC), com o objetivo de integrar os processos de diferentes SGWfNs através de um mesmo protocolo. Ferramentas para a interoperação de SGWfNs, como a proposta por Muehlen e Klein (2000), começaram a ser desenvolvidas desde a primeira versão da norma Wf-XML.

Já em workflows científicos não existe uma norma comum adotada pelos SGWfCs, o que dificulta o intercâmbio de especificações e a interoperação entre SGWfCs. Soluções que

[3]Neste trabalho empregamos o termo "norma" para referenciar documentos oficiais produzidos por organismos de padronização. O intuito é evitar o uso do termo "padrão", que neste trabalho é usado para referenciar padrões de arquitetura e projeto de software (patterns).

[4]`http://www.wfmc.org/`

propõem um repositório público para *download* e *upload* de especificações de workflows científicos de diferentes SGWfCs, como o repositório myExperiment[5] ou a rede social proposta por Zhang et al. (2011), podem ser úteis para grupos que utilizam o mesmo SGWfC, mas não resolvem o problema do intercâmbio de especificações entre diferentes SGWfCs.

Nesse contexto, soluções para o reuso através da interoperação de workflows científicos devem utilizar técnicas que não dependam simplesmente da disponibilização das especificações de workflows científicos em um repositório físico. Terstyanszky et al. (2014) apontam quatro alternativas principais de interoperação de workflows científicos:

- Uso de recursos de compartilhamento de dados de workflows científicos,
- Normatização da linguagem de especificação de workflows científicos,
- Integração dos SGWfCs, e
- Tradução das linguagens de especificação.

Como já dito, não existe uma normatização das especificações de workflows científicos e o compartilhamento de dados entre workflows é feito através de repositórios físicos. As duas alternativas restantes são analisadas abaixo.

Existem abordagens "caixa-preta" de integração dos SGWfCs, como por exemplo as propostas por Zhao et al. (2006) e Terstyanszky et al. (2014), onde os workflows a serem reutilizados são encapsulados e executados como tarefas em outros workflows. Nessas soluções, não há a necessidade de uma mudança no ambiente de trabalho corrente do cientista. Contudo, se o cientista precisar fazer pequenas modificações nos workflows a serem reutilizados, isso não será possível sem que haja acesso ao SGWfC correspondente.

Nos casos em que o cientista deseja ter maior flexibilidade para modificar, em seu ambiente de trabalho, um workflow científico reutilizado, é necessário que exista um intercâmbio de especificações entre SGWfCs diferentes. Porém, a inexistência de normas ligadas a especificação de workflows científicos praticamente inviabiliza o estabelecimento de um arcabouço semântico comum entre SGWfCs. Para intercambiar especificações de workflows científicos é necessário, portanto, que um processo de transformação seja realizado. Nesse processo, uma especificação de workflow científico referente a um SGWfC é

[5] `http://www.myexperiment.org/`

dada como entrada e uma especificação de workflow científico referente a outro SGWfC, com semântica semelhante à entrada, é gerada como saída.

Existem algumas técnicas de transformação de modelos que podem ser utilizadas no intercâmbio de especificações entre SGWfCs. Porém, antes de serem analisadas algumas dessas técnicas, uma outra abordagem de reuso – a de reuso de *design* – é discutida.

2.2.1 PADRÕES PARA WORKFLOWS CIENTÍFICOS

Uma outra possibilidade de se abordar o reuso de especificações de workflows é por meio do emprego de padrões de projeto (*design patterns*). A iniciativa Workflow Patterns[6] apresenta alguns padrões comumente encontrados em workflows de negócio, incluindo exemplos de implementações para diversos SGWfNs.[7] Esses padrões representam, em sua maioria, comportamentos relacionados ao fluxo de controle de um workflow, o que reflete o fato de workflows de negócio enfocarem a ordem de execução das tarefas, e não os dados que podem ser eventualmente trocados entre elas. Dentre os padrões apresentados na iniciativa Workflow Patterns, os mais relevantes no contexto do presente trabalho são:

- *Sequence* - Executa a tarefa após a execução da tarefa anterior;
- *Parallel Split* - Executa duas ou mais tarefas em paralelo após a execução da tarefa anterior;
- *Simple Merge* - Executa a tarefa por duas ou mais vezes após a execução de duas ou mais tarefas anteriores;
- *Exclusive Choice* - Executa uma entre duas ou mais tarefas após a execução da tarefa anterior.
- *Synchronization* - Executa a tarefa uma única vez após aguardar a execução de duas ou mais tarefas anteriores.

Complementarmente, Yildiz et al. (2009) apresentam alguns padrões para workflows científicos, especificados na linguagem *Business Process Modeling Notation* (BPMN). Esses padrões se assemelham aos apresentados na iniciativa Workflows Patterns, porém, separam explicitamente os fluxos de controle dos fluxos de dados.

[6] `http://www.workflowpatterns.com/`
[7] `http://www.workflowpatterns.com/vendors/index.php`

Migliorini et al. (2011) fazem um comparativo entre vários SGWfCs usando os padrões apresentados na iniciativa Workflow Patterns e demonstram que muitos desses padrões podem ser implementados nos SGWfCs. No entanto, algumas funcionalidades necessárias a essas implementações não são nativas em todos os SGWfCs, sendo necessária uma adaptação na especificação dos workflows, como por exemplo a inclusão de tarefas extras para se obter o comportamento esperado para o padrão sendo implementado.

Além dos padrões mencionados acima, vale destacar o trabalho de Ogasawara et al. (2011) que propõe um conjunto de padrões baseados em dados inspirado em álgebra relacional. Vários dos padrões descritos nesse trabalho mapeiam nos padrões apresentados na iniciativa Workflow Patterns, porém, a noção de dados descrita nesses padrões é baseada no conceito de tuplas, tipicamente adotado em bancos de dados relacionais.

2.2.2 TRANSFORMAÇÕES DE MODELOS

Neste trabalho considera-se a seguinte classificação das técnicas de transformação de modelos, proposta por Stahl et al. (2006):

- templates+filtragem: Utiliza templates para identificar os elementos em um modelo que é utilizado como entrada, e aplica um processo de filtragem sobre esses elementos para gerar a saída desejada. Um exemplo desse tipo de técnica é o empregado na linguagem XSLT[8] para transformação de documentos XML em outros formatos de representação.

- templates+metamodelos: É similar à técnica de templates+filtragem na leitura dos modelos de entrada, porém, utiliza um metamodelo como base para gerar o modelo de destino. A vantagem desse tipo de técnica é que a correção sintática – ou mesmo semântica, se o metamodelo de destino possuir a expressividade necessária – do modelo de destino é garantida por construção. Um exemplo desse tipo de técnica é o empregado na linguagem ATL (JOUAULT et al., 2008), que apoia a transformação entre modelos baseados em metamodelos que sigam a especificação Ecore do EMF (Eclipse Foundation, 2012).

- baseadas em APIs (*Application Programming Interfaces*): Utiliza uma API com operações para manipulação de elementos do modelo de destino, tipicamente mantido

[8] `http://www.w3.org/TR/xslt`

na forma de uma árvore sintática abstrata que é depois processada para a geração do modelo de destino em uma forma sintática concreta. A API proposta neste trabalho e apresentada no Capítulo 3 é um exemplo de uso desse tipo de técnica.

Complementarmente às técnicas supracitadas, o uso de um modelo de intermediação em processos de transformação permite desvincular a técnica de transformação associada ao modelo de origem daquela associada ao modelo de destino, o que é interessante na medida em que diferentes modelos podem ser melhor adaptados a diferentes técnicas de transformação – seja pelo arcabouço sintático e semântico, seja pela existência de ferramental associado. Esse modelo de intermediação deve seguir algum formalismo. Neste trabalho consideramos para esse formalismo a definição de uma linguagem intermediária, o que implica em considerações acerca da expressividade que a linguagem deverá conter, do arcabouço sintático e semântico a ser empregado nesta, e do ferramental disponível ou a ser desenvolvido.

2.2.3 LINGUAGENS DE DESCRIÇÃO ARQUITETURAL

Linguagens de descrição arquitetural (ADLs) permitem a criação de representações de arquitetura de software baseadas na composição de componentes (locais de processamento e armazenamento) e conectores (locais de interação), sendo portanto, naturalmente mapeáveis para as tarefas e conexões em um workflow. Essas representações podem ser expressas por meio de linguagem textual e/ou gráfica e podem se basear em um ou mais estilos arquiteturais (SHAW, 1990), que impõem restrições na composição de componentes e conectores. Algumas ADLs, como Acme (GARLAN et al., 1997) e Wright (SULEIMAN et al., 2008), permitem ainda a definição de novos estilos arquiteturais. De uma forma geral, as ADLs permitem a configuração das interações entre componentes por meio do emprego de conectores como construções de primeira classe. Ou seja, essas ADLs definem sistemas de tipos tanto para componentes quanto para conectores e, quando dão suporte a estilos arquiteturais, permitem definir restrições na combinação de instâncias desses tipos.

2.2.3.1 Acme

A ADL Acme (GARLAN et al., 1997) foi criada com a proposta de ser utilizada como um formato comum para o intercâmbio de especificações arquiteturais; por isso, ela possui um

arcabouço sintático e semântico mais abrangente e extensível que a maioria das ADLs. Além dessa característica, a ADL Acme possui outros atributos, descritos a seguir, que a fizeram ser escolhida neste trabalho para servir como base para a especificação da linguagem de intercâmbio entre especificações de workflows científicos ora proposta.

Além de componentes e conectores, a ADL Acme oferece os seguintes elementos de modelagem:

- Portas: interfaces que permitem a interação entre componentes;
- Papéis: pontos de acesso aos serviços de comunicação providos pelos conectores;
- Sistemas: grafos onde os vértices representam os componentes e as arestas representam os conectores;
- Ligações: relacionamentos entre componentes e conectores, os quais se dão através de suas interfaces (portas e papéis);
- Representações: elementos que permitem o detalhamento interno de componentes e conectores de forma a permitir o suporte a descrições hierárquicas de arquiteturas em Acme;
- Mapeamentos: elementos que indicam a correspondência entre as portas de um componente e as portas internas à sua representação.

A Figura 2.1 ilustra a representação gráfica, na ferramenta AcmeStudio, dos diversos elementos de modelagem oferecidos por Acme. A Listagem 2.1 apresenta a descrição textual em Acme deste mesmo exemplo.

```
System exemplo = {
  Component componenteA  = {
    Port portaA  = {}
    Port portaB  = {}
  }
  Component componenteB  = {
    Port portaA  = {}
    Port portaB  = {}
    Representation componenteB_Rep = {
      System componenteB_Sis = {
        Component A  = {
          Port portaA1  = {}
          Port portaA2  = {}
        }
```

```
        Component B  = {
          Port portaB1  = {}
          Port portaB2  = {}
        }
        Connector con  = {
          Role papel1  = {}
          Role papel2  = {}
        }
        Attachment A.portaA2 to con.papel1;
        Attachment B.portaB1 to con.papel2;
      }
      Bindings {
        portaB to B.portaB2;
        portaA to A.portaA1;
      }
    }
  }
  Connector conector  = {
    Role papel1  = {}
    Role papel2  = {}
  }
  Attachment componenteA.portaB to conector.papel1;
  Attachment componenteB.portaA to conector.papel2;
}
```

Listagem 2.1: Descrição textual em Acme do exemplo da Figura 2.1

Em Acme, o conceito de família é utilizado para descrever um conjunto de tipos de elementos de modelagem e regras de composição (na forma de predicados em lógica de

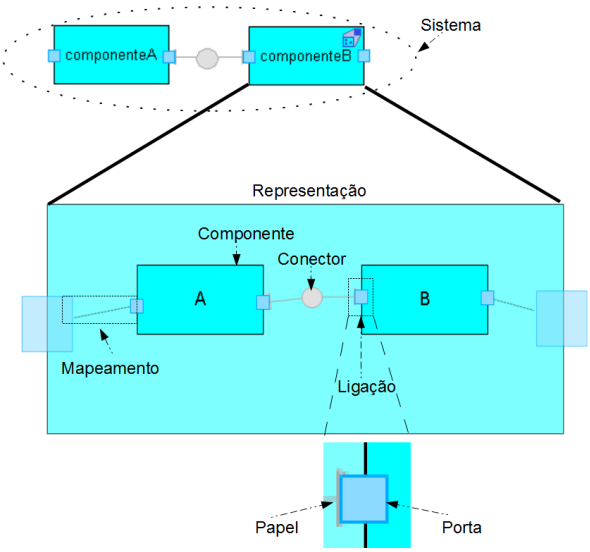


Figura 2.1: Representação gráfica de uma estrutura Acme (exemplo extraído de (MEDEIROS; GOMES, 2013)).

primeira ordem descritos na linguagem Armani (MONROE, 1999)) entre essas instâncias desses tipos. A Listagem 2.2 apresenta um exemplo de descrição de família em Acme para o estilo arquitetural "*Pipes and Filters*" fornecido na distribuição da ferramenta AcmeStudio[9]. O exemplo ilustra o uso de restrições para determinar certas características dos elementos de modelagem que forem derivados dos tipos declarados nessa família, como conectores do tipo *Pipe* terem que possuir dois papéis. Neste trabalho, o conceito de família é explorado para descrever os elementos de modelagem necessários para representar as especificações de workflows científicos (como será visto no Capítulo 3) e as regras de composição entre esses elementos.

```
Family PipesAndFiltersFam = {
  Component Type Filter = {
    Port input : inputT = new inputT extended with {}
    Port output : outputT = new outputT extended with {}
    Property function : string;
    rule portTypeConstraint = invariant forall e in self.PORTS |
      exists t in {inputT, outputT} |
        declaresType(e, t);
  }
  Port Type outputT = {
    Property protocol : string = ``char output'';
  }
  Port Type inputT = {
    Property protocol : string = ``char input'';
  }
  Connector Type Pipe = {
    Role source : sourceT = new sourceT extended with {
        Property protocol : string = ``char source'';
    }
    Role sink : sinkT = new sinkT extended with {
      Property protocol : string = ``char sink'';
    }
    Property bufferSize : int <<  default : int = 0; >> ;
    rule specifiedBufferSize = invariant self.bufferSize >= 0;
    rule exactlyTwoRoles = invariant size(self.ROLES) == 2;
  }
  Role Type sourceT = {
    Property protocol : string = ``char source'';
  }
  Role Type sinkT = {
    Property protocol : string = ``char sink'';
  }
}
```

[9] http://www.cs.cmu.edu/~acme/AcmeStudio/index.html

Listagem 2.2: Descrição Acme do estilo arquitetural "*Pipes and Filters*".

A adoção de Acme por si só não é solução para o problema da inexistência de uma base semântica comum entre os SGWfCs; contudo, podemos identificar alguns padrões (*patterns*) na construção de workflows científicos, independente do SGWfC em uso, que se devidamente representados na linguagem proposta, podem mitigar a perda de informação em uma transformação. Nesse sentido, foram utilizados neste trabalho os padrões desenvolvidos na iniciativa Workflow Patterns, porém, com um comportamento orientado a fluxos de dados e não de controle, já que como mencionado anteriormente, workflows científicos são tipicamente orientados a dados. A estratégia adotada para representar esses padrões (como será visto no Capítulo 3) foi a de definir diferentes tipos de conectores – e restrições associadas – para diferentes padrões, além de um tipo genérico de componente para representar as tarefas, com especializações desse tipo podendo ser definidas para cada SGWfC de modo a contemplar tipos particulares de tarefas de cada SGWfC.

2.3 TRABALHOS RELACIONADOS

Ao analisar o intercâmbio entre especificações de workflows científicos, é possível encontrar alguns trabalhos que propõem o intercâmbio de especificações de diferentes maneiras, com vantagens e desvantagens. Acredita-se que isso se deva ao fato de não existir uma norma seguida por todos os SGWfCs, o que dificulta uma solução que atenda todos os SGWfCs.

Iniciativas como OPM (MOREAU et al., 2011) e PROV (BELHAJJAME et al., 2012) buscam a normatização da proveniência de dados em workflows científicos, viabilizando, por exemplo, a correta reprodução de um experimento executado em um SGWfC. A proveniência pode ser de dois tipos: prospectiva e retrospectiva (DAVIDSON; FREIRE, 2008). A proveniência prospectiva captura a estrutura do workflow através de sua especificação. Essa proveniência não possui dados utilizados na execução do workflow, uma vez que é obtida antes de sua execução. A proveniência retrospectiva fornece estruturas para a catalogação dos dados de entrada e saída de workflows científicos, assim como os dados intermediários manipulados pelas tarefas que foram executadas nesses workflows e a ordem na qual essas tarefas foram executadas. Alguns trabalhos, como SciProv (GASPAR et al., 2015) e ProvManager (MARINHO et al., 2012), buscam interoperar a coleta

da proveniência de dados de diferentes SGWfCs em um ambiente colaborativo. Quando a proveniência coletada é prospectiva, esses trabalhos fornecem uma maneira indireta e de cobertura parcial de se intercambiar especificações de workflows por meio da representação intermediária fornecida pelas normas de proveniência. Em primeiro lugar, essa maneira demanda esforço considerável do cientista interessado em empregar esses trabalhos para reusar especificações provenientes de outros SGWfCs no SGWfC de seu interesse, pois é necessária a transformação das representações de proveniência para a linguagem de especificação suportada por esse SGWfC e essas representações de proveniência não carregam informações semânticas suficientes para que essas transformações possam ser feitas de forma automatizada. Em segundo lugar, essa maneira demanda que o SGWfC implemente mecanismos de proveniência prospectiva baseados nas normas OPM ou PROV para que seus workflows possam ser eventualmente reusados em outros SGWfCs.

Soluções específicas para o intercâmbio de especificações de workflows científicos visam intercambiar o workflow sem a necessidade de aprendizado de um novo SGWfC, ou então de replicar manualmente a estrutura e comportamento desse workflow em um SGWfC conhecido. A ferramenta Taverna 2-Galaxy (KARASAVVAS et al., 2012) é um dos trabalhos que propõe intercambiar especificações de workflows científicos. Essa ferramenta propõe uma abordagem de intercâmbio ponto-a-ponto, onde especificações desenvolvidas no SGWfC Taverna são convertidas para o SGWfC Galaxy (GIARDINE et al., 2005). Abordagens ponto-a-ponto podem garantir uma reprodução com alta fidelidade do workflow de origem no SGWfC de destino. Contudo, essas abordagens podem requerer demasiado esforço de desenvolvimento caso queira se dar suporte a outros SGWfCs, uma vez que o número de transformações necessárias cresce de acordo com o número de SGWfCs que deseja-se interoperar com outros. Caso um desenvolvedor deseje intercambiar especificações entre N SGWfCs diferentes, serão necessários $N(N-1)$ processos de transformação distintos, o que pode tornar a abordagem inviável em um cenário mais amplo de utilização.

Plankensteiner et al. (2011) apresentam uma solução para o intercâmbio de workflows científicos através de uma linguagem intermediária chamada *Interoperable Workflow Intermediate Representation* (IWIR). Essa solução é a que mais se assemelha com a proposta do presente trabalho, uma vez que utiliza uma linguagem intermediária desenvolvida especificamente para intercambiar workflows científicos entre diferentes SGWfCs. Essa solução, contudo, apresenta algumas limitações. A mais importante delas é que, ao analisar

a estrutura da linguagem IWIR, percebe-se claramente que ela herda estruturas e tipos de tarefas (em particular aqueles associados a tratamentos genéricos de fluxos de dados, como condições e laços de repetição) muito semelhantes àqueles providos pelo grupo específico de SGWfCs para as quais IWIR dá suporte: ASKALON, MOTEUR, P-GRADE e Triana (FAHRINGER et al., 2007; GLATARD et al., 2008; KACSUK, 2011; TAYLOR et al., 2003). A linguagem proposta no presente trabalho, por outro lado, se baseou em padrões, enfocando a captura da semântica associada aos fluxos de dados e tratando tarefas como "esqueletos" em que toda a lógica interna dessas tarefas está encapsulada na implementação particular das mesmas no SGWfC, e não na linguagem intermediária em si. No presente trabalho alguns SGWfCs foram usados na avaliação, porém, o trabalho não se baseou na especificação de nenhum desses SGWfCs. A linguagem ora proposta tem um arcabouço semântico independente e é necessário que o mecanismo de transformações associado a ela adapte as informações encontradas nas especificações dos SGWfCs para que possam ser representadas na linguagem ora proposta.

Abouelhoda et al. (2012) apresentam uma estratégia de interoperação baseada em padrões para os SGWfCs Taverna e Galaxy. Contudo, além dessa proposta se limitar a dois SGWfCs, um problema nessa interoperação é que cientistas têm que aprender a lidar com um novo SGWfC, uma vez que essa estratégia não foi projetada como uma ferramenta de intercâmbio e sim como um novo SGWfC em si. Considerando o contexto deste trabalho de dissertação, ou seja, o intercâmbio de especificações de workflows científicos, o usuário não precisa conhecer a linguagem intermediária, pois ela só será utilizada durante o processo de transformação de duas especificações. Apenas o desenvolvedor que for responsável por implementar os processos de transformação da linguagem de especificação de um SGWfC para a linguagem intermediária, e vice-versa, deve conhecer essa linguagem, representando os comportamentos desejados em cada um dos lados da transformação.

Outros trabalhos que não são diretamente relacionados ao intercâmbio de workflows científicos, no entanto, possuem alguma relação com o este trabalho são apresentados por Cerezo et al. (2013) e Gesing et al. (2013).

Cerezo et al. (2013) apresentam uma estratégia baseada em *model-driven engineering*, que propõe o uso de uma representação abstrata de um workflow que pode ser traduzida para uma especificação workflow concreta de diferentes SGWfCs. No entanto, esse trabalho não considera o intercâmbio de workflows legados, sendo feita apenas uma trans-

formação de uma especificação abstrata para uma especificação concreta.

Gesing et al. (2013) apresentam a proposta de um SGWfC multi-linguagem, com o principal objetivo de proporcionar a conversão de diferentes workflows, desenvolvidos em diferentes SGWfCs. Para atingir esse objetivo, o trabalho precisaria disponibilizar diferentes funcionalidades encontradas nos diferentes SGWfCs, o que se mostrou um grande desafio. Como resultado deste desafio, os autores decidiram não tratar o intercâmbio entre diferentes especificações de workflows científicos.

2.3.1 CONCLUSÕES DO CAPÍTULO

Este capítulo apresentou uma revisão da literatura acerca dos principais conceitos relacionados à proposta desta dissertação. Foi feita uma revisão dos conceitos de workflows científicos, padrões para workflows científicos, técnicas de transformação de modelos e ADLs. Além disso, foram analisados os principais trabalhos encontrados na literatura relacionados à abordagem apresentada nesta dissertação.

Vale ressaltar que, conforme será detalhado no Capítulo 3, a análise dos padrões para workflows científicos da iniciativa Workflow Patterns foi feita levando em consideração que fluxos em workflows científicos são tipicamente orientados a dados. Além disso, alguns padrões que são inteiramente voltados ao fluxo de controle não foram adicionados, em primeira instância, a essa linguagem, por essas construções não serem encontradas diretamente em workflows científicos e serem muitas vezes executadas como um comportamento interno das tarefas.

3 WISP: UMA PROPOSTA PARA INTERCÂMBIO DE ESPECIFICAÇÕES DE WORKFLOWS CIENTÍFICOS

3.1 INTRODUÇÃO

Este trabalho trata o problema do intercâmbio de especificações de workflows científicos por meio de uma linguagem intermediária denominada *Workflow Interchange in Science with Patterns* – WISP. O principal diferencial dessa linguagem em comparação com trabalhos similares apresentados no capítulo anterior está na estratégia de definir seus elementos de modelagem com base em padrões (*patterns*) de workflows científicos combinados a conceitos de arquitetura de software. As hipóteses, apresentadas no Capítulo 1, nas quais se baseia essa estratégia é que essa combinação permite mitigar a perda de informação semântica nesse intercâmbio.

Os padrões são utilizados na especificação da linguagem para capturar a semântica de um workflow e possibilitar sua reprodução em outro SGWfC. Assim, embora a construção desses padrões possa ser diferente em cada SGWfC, seu comportamento se dará de acordo com a semântica associada a cada padrão. Uma vez que a especificação de um workflow em um determinado SGWfC de origem é convertida para a linguagem intermediária proposta neste trabalho, sua informação semântica, baseada na identificação de um conjunto de padrões presentes nesse workflow, deve ser preservada. Assumindo que o SGWfC de destino dê suporte aos padrões encontrados no workflow original, mesmo que sua construção não seja nativa no SGWfC de destino, a especificação desse workflow na linguagem intermediária permite reproduzir seu comportamento no SGWfC de destino.

A ADL Acme foi escolhida neste trabalho para definir formalmente a estrutura da linguagem de intercâmbio em termos de conjuntos de padrões de workflows científicos descritos na forma de tipos de conectores. O uso de Acme permite estabelecer regras de formação de instâncias desses tipos, bem como adicionar funcionalidades importantes e que não são encontradas em todos os SGWfCs, como o tratamento de requisitos não-funcionais. O suporte à tipagem múltipla de instâncias de elementos arquiteturais em Acme também possibilita a composição de estruturas prescritas pelos tipos sem a neces-

sidade de criação de subtipos. Esse atributo de Acme torna-a qualificada para descrever uma linguagem intermediária devido à heterogeneidade encontrada nos SGWfCs.

Além da linguagem, um mecanismo de transformações é necessário para converter as especificações que estão na linguagem nativa de cada SGWfC. Este é também um resultado esperado deste trabalho. Inicialmente foi feita uma análise dos padrões propostos na iniciativa Workflow Patterns[1] e de workflows presentes no repositório myExperiment.[2] Em uma primeira abordagem foram incluídos na linguagem WISP e no mecanismo de transformações associado os padrões *Sequence, Parallel Split, Simple Merge, Synchronization* e *Exclusive Choice*, apresentados na iniciativa Workflows Patterns e brevemente descritos no Capítulo 2, a fim de serem utilizados para uma primeira validação das hipóteses apresentadas no Capítulo 1. Entretanto, como também discutido no Capítulo 2, embora esses padrões representem fluxos de controle de um workflow, no contexto deste trabalho eles serão empregados na representação de fluxos de dados, sendo um comportamento típico dos workflows científicos. Assim, os padrões supracitados são interpretados na linguagem WISP do seguinte modo:

- *Sequence* - Liga uma única porta de saída a uma única porta de entrada;
- *Parallel Split* - Liga uma única porta de saída a duas ou mais portas de entrada, replicando o mesmo dado em todas.
- *Simple Merge* - Liga duas ou mais portas de saída a uma única porta de entrada, preenchendo a porta com o dado recebido por cada porta de entrada por vez.
- *Synchronization* - Liga duas ou mais portas de saída a uma única porta de entrada. Similar ao padrão *Simple Merge* em sua estrutura, porém, a tarefa seguinte só deve ser executada assim que os dados de todas as tarefas anteriores forem recebidos e agrupados segundo algum critério, a fim de serem passados para a tarefa seguinte.
- *Exclusive Choice* - Liga uma única porta de saída a duas ou mais portas de entrada. Similar ao padrão *Parallel Split* em sua estrutura, porém, só uma das tarefas subsequentes deve receber o dado de acordo com alguma condição associada ao padrão.

É interessante observar que os dois últimos padrões – *Synchronization* e *Exclusive Choice* – apresentam um comportamento interno associado a um critério ou condição. Denomi-

[1] `http://www.workflowpatterns.com/`
[2] `http://www.myexperiment.org/`

namos esses padrões como **padrões comportamentais**, em contraposição aos demais padrões, que denominamos como **padrões estruturais**.

Como pode ser visto na Figura 3.1, emprega-se o conceito de família presente na ADL Acme para especificar os padrões. Uma família define um conjunto de tipos de elementos de modelagem arquitetural (componentes e portas, conectores e papéis) e regras de composição entre instâncias desses tipos. Na linguagem especificada neste trabalho, os tipos definidos na família Acme descrevem os padrões como conexões (conectores e papéis) entre tarefas (componentes) de um workflow, pois uma vez que workflows científicos são em geral orientados a dados, seu comportamento é observável no momento em que dados são trocados entre as portas das tarefas. A Figura 3.1 também ilustra dois tipos de tarefas *Task e WebServiceTask* que foram adicionadas a essa família. As tarefas do tipo *Task* representam apenas o esqueleto das tarefas de um workflow com suas portas de entrada e saída. O tipo *WebServiceTask* é uma especialização do tipo genérico *Task*, sendo usado para descrever tarefas implementas como serviços Web, característica comum à maioria dos SGWfCs estudados e presente em todos os SGWfCs suportados na versão atual da linguagem WISP – Taverna, Kepler e VisTrails. A tabela 3.1 ilustra a implementação dos padrões nos diferentes SGWfCs.

Padrão	SGWfC	Implementação
Sequence	Taverna	Nativo do SGWfC
	Kepler	Nativo do SGWfC
	VisTrails	Nativo do SGWfC
Parallel Split	Taverna	Nativo do SGWfC
	Kepler	Nativo do SGWfC
	VisTrails	Nativo do SGWfC
Simple Merge	Taverna	Nativo do SGWfC
	Kepler	Nativo do SGWfC
	VisTrails	Nativo do SGWfC
Synchronization	Taverna	Programaticamente, através de um script nativo do SGWfC
	Kepler	Tarefa nativa do SGWfC para dados do tipo String
	VisTrails	Programaticamente, através de um script nativo do SGWfC
Exclusive Choice	Taverna	Programaticamente, através de um script nativo do SGWfC
	Kepler	Tarefa nativa do SGWfC
	VisTrails	Programaticamente, através de um script nativo do SGWfC

Tabela 3.1: Implementação dos Padrões nos SGWfCs Taverna, Kepler e VisTrails

Esse tipo tem atributos específicos de serviços Web (como endereço do serviço e operação a ser executada) que devem ser obrigatoriamente preenchidos em uma instanciação desse tipo. É importante destacar que novos tipos especializados de tarefas podem ser

incluídos na linguagem. No Apêndice A.1 desta dissertação é apresentada a descrição textual detalhada dessa família, assim como todas as regras associadas aos elementos de modelagem que a compõem.

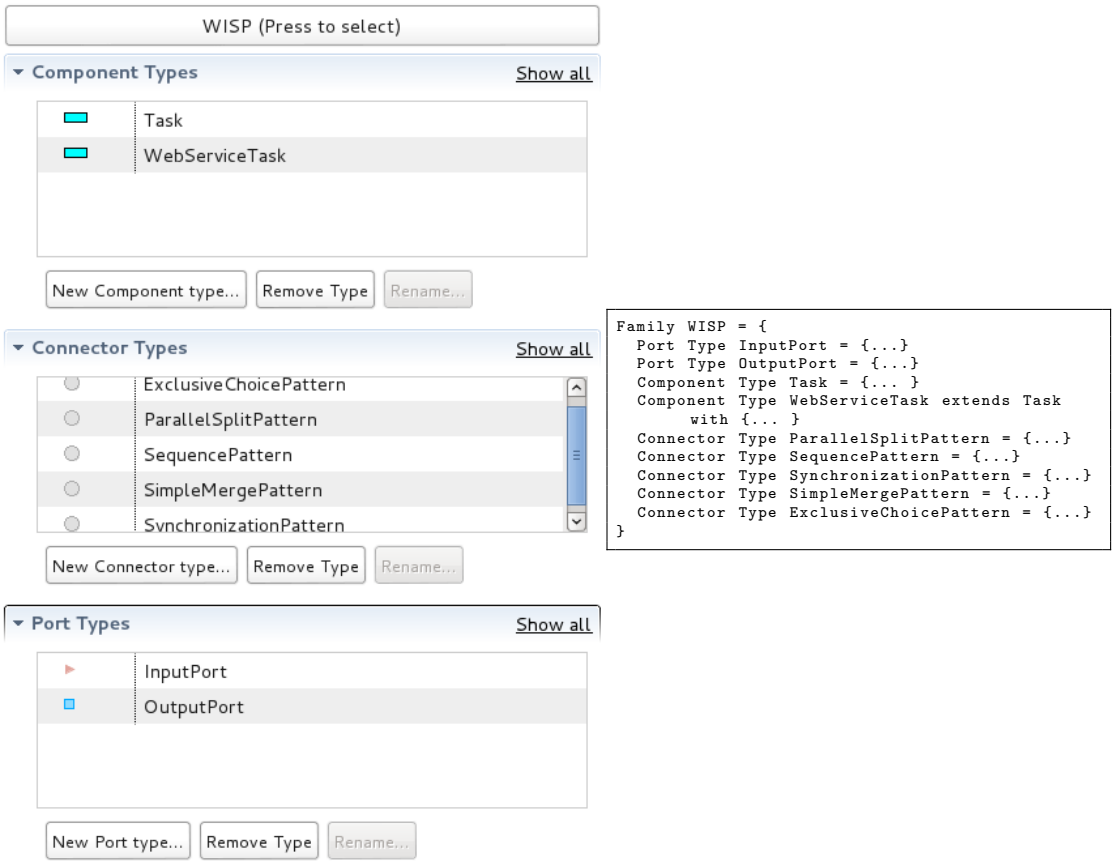
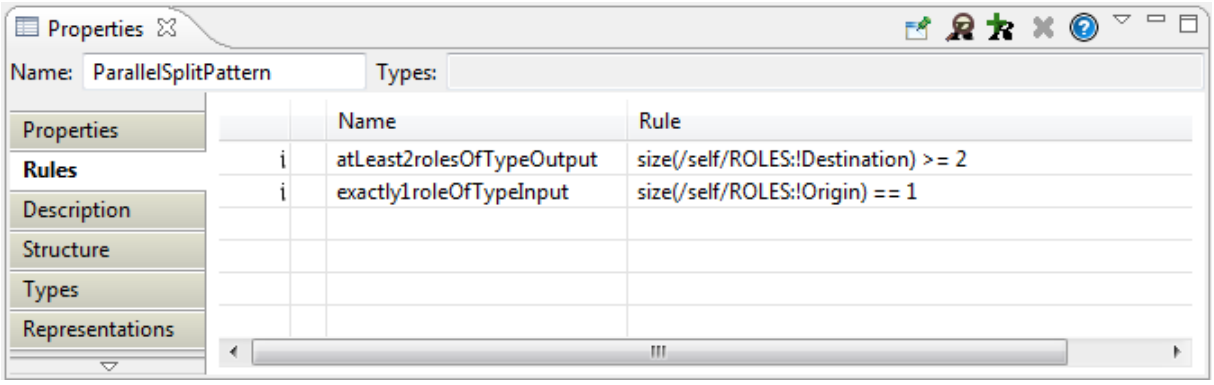


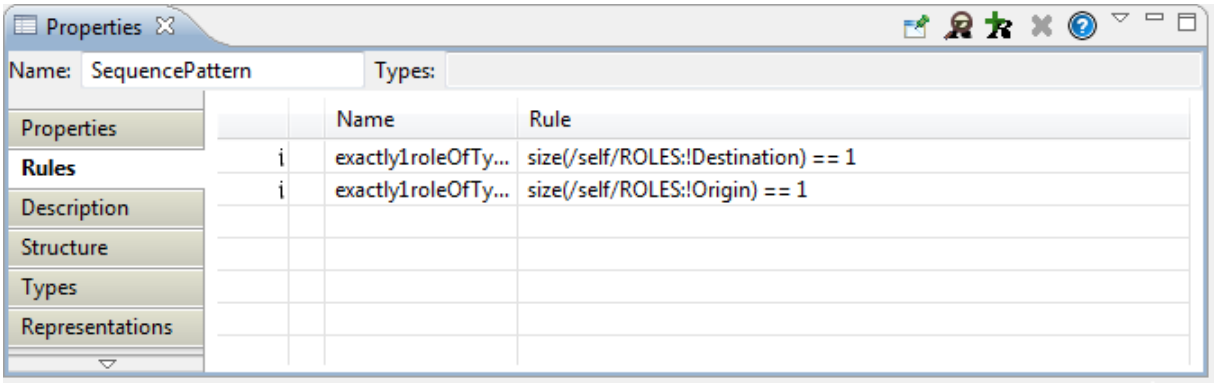
Figura 3.1: Família Acme da Linguagem WISP

Para garantir a estrutura e o comportamento esperados desses padrões, regras de formação foram adicionadas para cada um deles. Essas regras permitem restringir o número de conexões presentes em um padrão. A Figura 3.2 ilustra duas regras simples para os padrões *Parallel Split* e *Sequence*, estabelecendo a quantidade de portas de entrada e saída que instâncias desses padrões devem ter, garantindo assim, que os elementos necessários para realizar o comportamento do padrão sejam satisfeitos. O padrão *Parallel Split*, por exemplo, não pode ser implementado sem que ao menos duas portas de entrada sejam conectadas a uma única porta de saída, uma vez que seu comportamento espera que um mesmo dado recebido de uma porta de saída seja replicado em duas ou mais portas de entrada.

Ainda nesse contexto, regras que validam atributos de tarefas também podem ser adicionadas. A Figura 3.3 ilustra regras que restringem os atributos de uma tarefa do



(a) Regras do Padrão Parallel Split



(b) Regras do Padrão Sequence

Figura 3.2: Regras Armani para os Padrões *Parallel Split e Sequence*

tipo serviço Web (*WebServiceTask*), em que o endereço do serviço (*wsdl*) e a operação que será executada (*operation*) devem ser informadas para que a tarefa tenha sua semântica preservada.

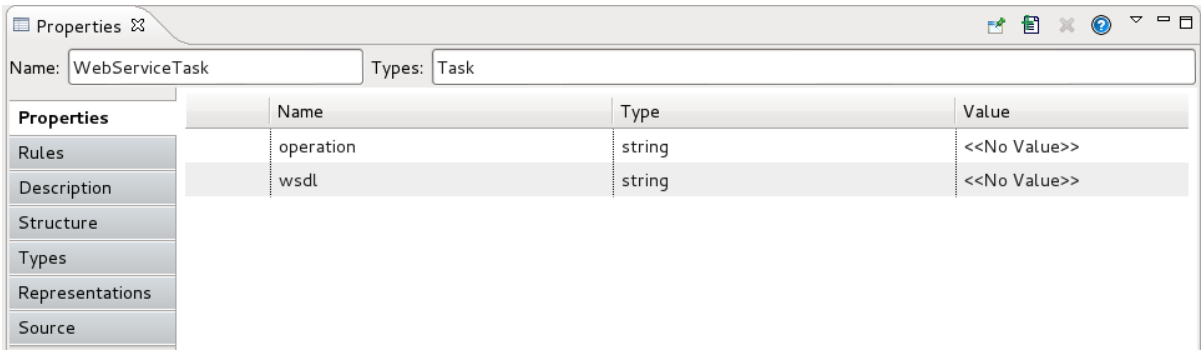


Figura 3.3: Regras para a Tarefa *WebServiceTask*

Uma vez que regras restrinjam a estrutura de um determinado padrão ou tarefa, é possível validar se sua construção está correta, assim como verificar a presença de todas as informações necessárias para representar seu comportamento. Uma especificação de workflow científico descrita na linguagem WISP garante assim as informações que um

padrão deve ter em um determinado trecho do workflow, abstraindo dos detalhes necessários para atingir esse comportamento em um SGWfC específico. Em alguns SGWfCs, é possível que a implementação de um padrão seja nativa em um SGWfC por meio de um elemento de modelagem específico, por exemplo, enquanto em outros pode ser necessária a composição de vários desses elementos. Porém, ao utilizar padrões e regras que restrinjam essas composições, é possível garantir que esse workflow terá o mesmo comportamento em qualquer SGWfC que implemente esses padrões.

Um exemplo desse cenário é a implementação do padrão estrutural *Simple Merge* nos SGWfCs Taverna, VisTrails e Kepler. No Taverna o uso de mais de uma conexão em uma porta de entrada não é possível, sendo necessária a utilização de uma junção (elemento especial de modelagem) entre duas ou mais conexões, que é criada pelo próprio SGWfC no momento do desenvolvimento do workflow. No VisTrails não existe o conceito de junção para as conexões, e elas são feitas diretamente nas portas das tarefas. No Kepler deve-se informar explicitamente que uma porta permite múltiplas conexões, acarretando em um erro caso essa porta receba mais de uma conexão e não esteja configurada corretamente para tal, ou então empregar uma junção, como no Taverna, que não é feita automaticamente pelo SGWfC. Ao utilizar uma representação abstrata desse tipo de estrutura por meio padrão *Simple Merge* na linguagem WISP, é delegado ao mecanismo de transformações a reificação desse tipo de estrutura no SGWfC alvo.

Um outro exemplo é o do padrão *Synchronization* que possui a mesma estrutura do padrão *Simple Merge*. É possível que em algum SGWfC tarefas possam ser executadas quando apenas uma de suas portas de entrada seja satisfeita – embora não tenhamos observado essa característica nos SGWfCs estudados, esse é o comportamento típico de workflows de negócio –, enquanto outros SGWfCs podem permitir a execução de uma tarefa apenas quando todas as suas portas de entrada forem satisfeitas. O uso de conexões simples de dados entre portas de origem e de destino para representar essas situações em uma linguagem de intercâmbio pode levar a ambiguidades de representação. Ao representar padrões na linguagem de intercâmbio, podemos claramente delinear seu papel de sincronização (ou de concatenação dos valores) de todas as portas de entrada de uma tarefa antes de ela ser executada. Embora as construções nos SGWfCs de destino possam ser diferentes, utilizando padrões estaremos deixando claro o comportamento que esperamos daquele workflow após ser transformado na especificação do SGWfC de destino.

Contudo, é importante observar que as regras associadas a cada padrão definem apenas sua estrutura em relação aos dados que devem ser trocados entre as portas de entrada e saída das tarefas. Ainda existe a semântica associada ao padrão, que embora não seja validada pelas regras Armani, uma vez que cada SGWfC pode representar um padrão de uma maneira própria, ainda deve ser capturada em uma transformação para garantir a semântica do workflow. Particularmente no caso de padrões comportamentais, a forma pela qual um padrão é implementado em diferentes SGWfCs faz com que sua identificação automática possa ser complexa.

Um exemplo desse cenário é a implementação do padrão *Exclusive Choice* nos SGWfCs Taverna e Kepler. O SGWfC Taverna não possui suporte nativo para a implementação desse padrão, porém, esse comportamento pode ser atingido através de scripts *BeanShell*. Ao implementar um script *BeanShell* o desenvolvedor do workflow deve declarar todas as possíveis portas de saída e atribuir seus valores dentro do script, de acordo com alguma condição associada à entrada da tarefa. Cabe ao desenvolvedor do workflow fazer uma verificação nas saídas do script e verificar se algum valor foi retornado e só assim executar a próxima tarefa. Um exemplo da implementação desse padrão no Taverna é ilustrado na Figura 3.4.

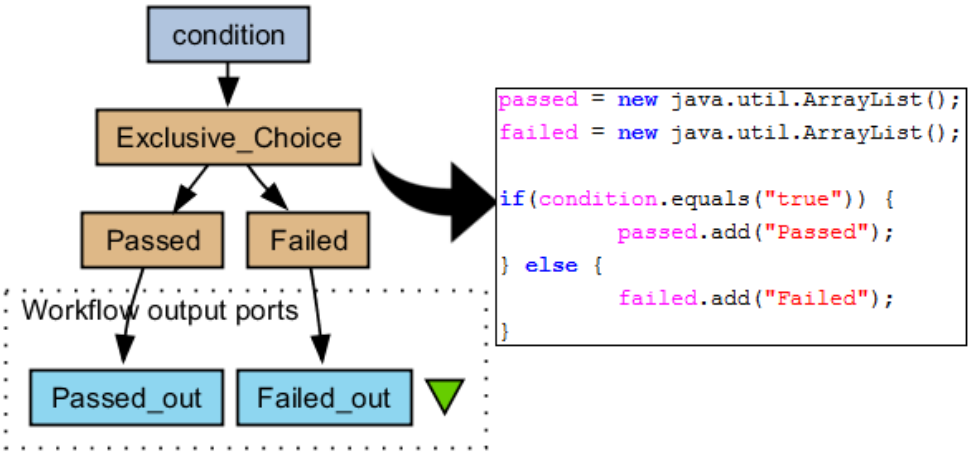


Figura 3.4: Padão *Exclusive Choice* no Taverna

No SGWfC Kepler, existem dois tipos pré-definidos de tarefas no SGWfC que possuem o comportamento do padrão *Exclusive Choice*. Um desses tipos recebe um valor booleano e possui exatamente duas saídas que são acionadas de acordo com o valor passado (verdadeiro ou falso). Já o segundo tipo implementa o mesmo comportamento exclusivo em um intervalo de números inteiros, começando em 0 e terminando em $N - 1$, onde N

é o número de tarefas dentre as quais apenas uma será executada. Um exemplo das duas versões da implementação desse padrão no Kepler é ilustrado na Figura 3.5

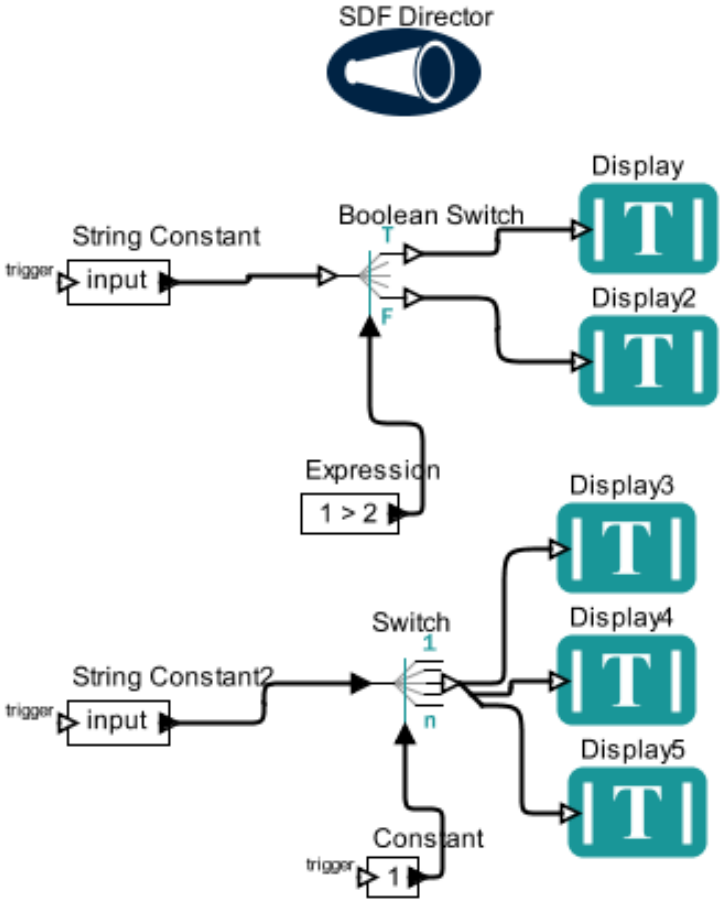


Figura 3.5: Padrão *Exclusive Choice* no Kepler

Os trechos completos das especificações acima nas linguagens do Taverna e do Kepler são apresentados no Apêndice B.3.

No exemplo apresentado acima, o padrão *Exclusive Choice* pode ser identificado em especificações Kepler apenas analisando a estrutura do workflow e os tipos de tarefas envolvidas; no caso do Taverna, pode não ser possível identificar a existência de um padrão comportamental sem alguma inteligência adicional associada ao mecanismo de transformação que permita inferir o comportamento associado ao trecho de especificação em questão (por exemplo, analisando as saídas geradas pelo script *BeanShell* para um conjunto de entradas de teste).

Nas seções e capítulos seguintes serão detalhadas as soluções possíveis para a identificação desses padrões, assim como os cenários que levaram às limitações encontradas nesta dissertação.

3.1.1 MECANISMO DE TRANSFORMAÇÃO

Confirme já dito, a proposta deste trabalho é viabilizar o reuso de workflows científicos através de transformações. Para atingir esse objetivo foi proposto, conjuntamente à linguagem WISP, um mecanismo de transformação extensível – na forma de um framework – que oferece operações que podem ser implementadas por diferentes desenvolvedores.

Os mecanismos de transformações de/para a linguagem WISP são definidos abstratamente por uma interface chamada `Parser`. Essa interface define duas operações principais chamadas `fromWISP()` e `toWISP()`. Essas operações são responsáveis por transformar uma especificação de workflow científico na linguagem WISP para a linguagem de um SGWfC e vice-versa. A Figura 3.6 ilustra, usando notação UML,[3] a estrutura dessa interface. A implementação das operações dessa interface, para cada SGWfC, deve ser feita por um desenvolvedor com amplo conhecimento da estrutura e semântica das especificações reconhecidas por esse SGWfC. Ao ler uma especificação de workflow científico na operação `toWISP()`, é necessário que os padrões sejam identificados e reproduzidos na linguagem WISP. Com o intuito de validar a proposta, foram implementadas três instâncias da interface `Parser` para os SGWfCs Taverna, Kepler e VisTrails. Essas implementações serão apresentadas em detalhes nas seções a seguir.

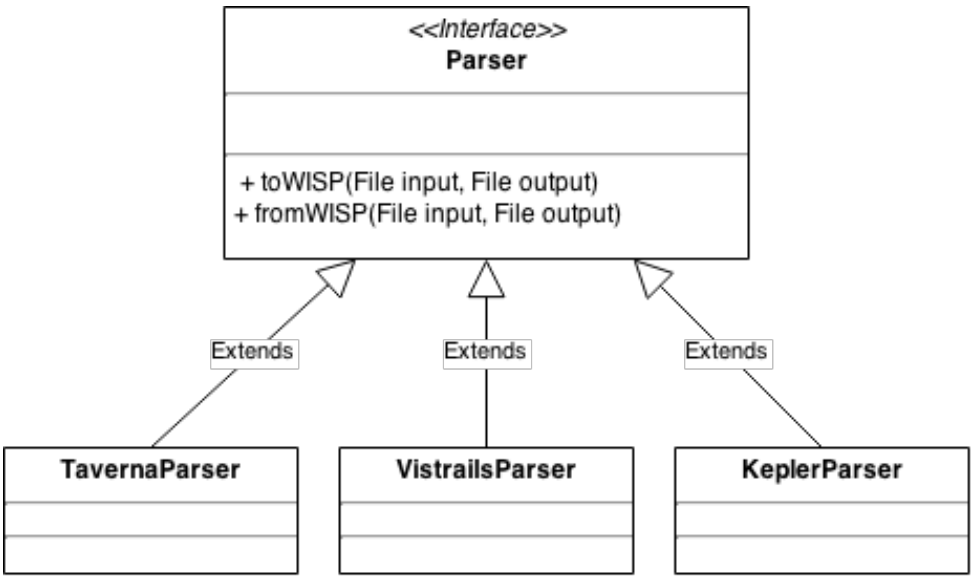


Figura 3.6: Interface *Parser* do Framework da linguagem WISP

Esse framework também oferece um modelo de objetos que representa um workflow científico na linguagem WISP. Esse modelo é ilustrado na Figura 3.7 através de um dia-

[3] `http://www.uml.org/`

grama de classes. A classe `Task` representa uma tarefa e possui portas de entrada e saída (classes `InputPort` e `OutputPort`). A classe `Workflow`, que por sua vez pode ser vista como uma tarefa (ou workflow interno), possui uma lista de ligações (classe `Binding`) e possui uma lista de padrões (classe `Pattern`) que representam as conexões entre as tarefas e cada padrão descrito na linguagem WISP. As ligações de um workflow (`Binding`) são utilizadas quando um workflow interno deseja mapear uma de suas portas de entrada ou saída com portas de entrada ou saída de suas tarefas internas. Diferente das conexões (padrões), suas ligações devem ser feitas com o mesmo tipo de porta (porta de entrada com porta de entrada e porta de saída com porta de saída) e por isso não podem ser representadas como um padrão *Sequence*, pois suas regras não seriam validadas.

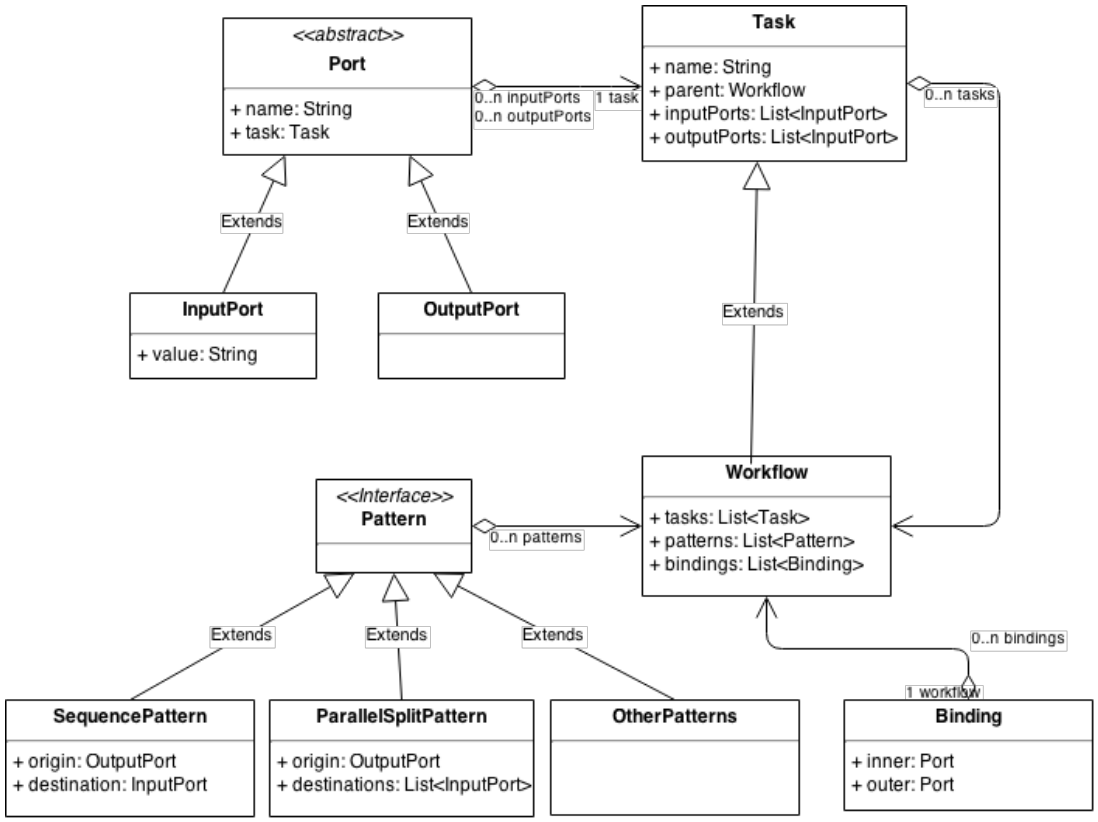


Figura 3.7: Estrutura que representa um Workflow na linguagem WISP

3.2 USO DO FRAMEWORK

Para ilustrar o funcionamento do framework, foi criado um workflow simples por meio da IDE AcmeStudio[4] com a família da linguagem WISP. Esse workflow foi criado no

[4] `http://www.cs.cmu.edu/~./acme/AcmeStudio/index.html`

contexto desta dissertação para ilustrar o funcionamento do framework e é composto de quatro operações matemáticas que são representadas por tarefas do tipo "serviços Web". A Figura 3.8 ilustra a representação desse workflow, cuja proposta é gerar dois números aleatórios, que são gerados a partir de dois intervalos passados como entrada ao workflow, e representados pelas duas portas de entrada da tarefa `random_1` e pelas duas portas de entrada da tarefa `random_2`. Após gerados, esses números são retornados pelas portas de saída de suas tarefas e utilizados como entrada da tarefa `sum`, que, por sua vez, retorna a soma desses números em sua porta de saída. Um dos números aleatórios (retorno da tarefa `random_2`), em conjunto com o retorno da tarefa `sum`, também são utilizados como entrada para a tarefa `sub` e sua subtração é o resultado desse workflow.

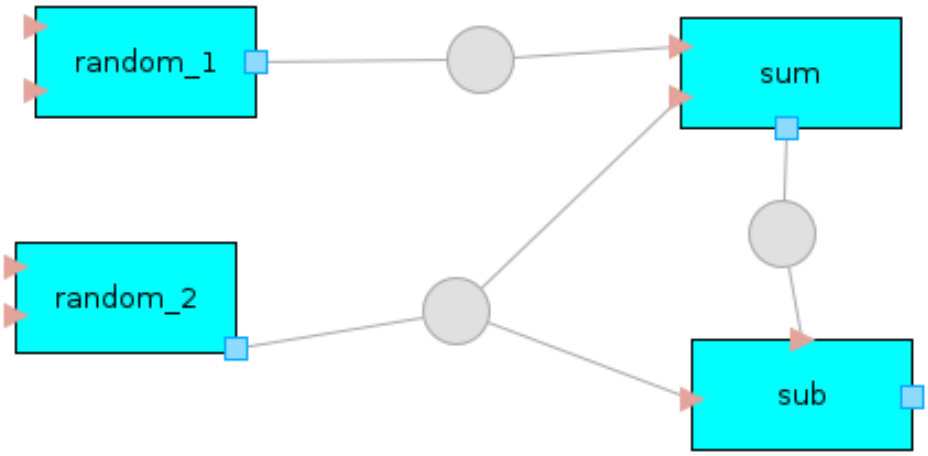


Figura 3.8: Exemplo de Workflow - WISP

Pretende-se representar ao máximo a semântica associada a esse workflow na linguagem WISP, para que sua reprodução seja possível em outros SGWfCs. Basicamente, o comportamento desse workflow pode ser observado no momento em que os dados são trocados entre as tarefas. Diferente da especificação de alguns SGWfCs, tarefas auxiliares não são representadas na linguagem WISP. Além disso, tarefas do tipo "Constantes" também não são representadas nessa linguagem, uma vez que essas constantes podem ser representadas como um valor associado a uma porta de entrada de uma tarefa. O código desse workflow é ilustrado em 3.1 e é possível identificar que as portas de entrada das tarefas `random_1` e `random_2` possuem valores constantes, e que as conexões entre as tarefas desse workflow são representadas por três padrões, sendo dois padrões *Sequence* e um padrão *ParallelSplit*.

```
import families/WISP.acme;
System Workflow : WISP = new WISP extended with {
  Component random_2 : WebServiceTask = new WebServiceTask extended with {
    Port i_min : InputPort = new InputPort extended with {
      Property value = "1";
    }
    Port i_max : InputPort = new InputPort extended with {
      Property value = "14";
    }
    Port o_return : OutputPort = new OutputPort extended with {
    }
    Property wsdl = "http://192.241.177.47:8080/engine-service/RandomService?wsdl";
    Property operation = "randomPrimitive";
  }
  Component random_1 : WebServiceTask = new WebServiceTask extended with {
    Port i_min : InputPort = new InputPort extended with {
      Property value = "1";
    }
    Port i_max : InputPort = new InputPort extended with {
      Property value = "14";
    }
    Port o_return : OutputPort = new OutputPort extended with {
    }
    Property wsdl = "http://192.241.177.47:8080/engine-service/RandomService?wsdl";
    Property operation = "randomPrimitive";
  }
  Component sum : WebServiceTask = new WebServiceTask extended with {
    Port i_a : InputPort = new InputPort extended with {
    }
    Port i_b : InputPort = new InputPort extended with {
    }
    Port o_return : OutputPort = new OutputPort extended with {
    }
    Property wsdl = "http://192.241.177.47:8080/engine-service/RandomService?wsdl";
    Property operation = "sumPrimitive";
  }
  Component sub : WebServiceTask = new WebServiceTask extended with {
    Port i_a : InputPort = new InputPort extended with {
    }
    Port i_b : InputPort = new InputPort extended with {
    }
    Port o_return : OutputPort = new OutputPort extended with {
    }
    Property wsdl = "http://192.241.177.47:8080/engine-service/RandomService?wsdl";
    Property operation = "subPrimitive";
  }
  Connector pattern_1 : ParallelSplitPattern = new ParallelSplitPattern extended with {
    Role input : Origin = new Origin extended with {
```

```
    }
    Role output_1 : Destination = new Destination extended with {
    }
    Role output_2 : Destination = new Destination extended with {
    }
  }
  Connector pattern_2 : SequencePattern = new SequencePattern extended with {
    Role input : Origin = new Origin extended with {
    }
    Role output : Destination = new Destination extended with {
    }
  }
  Connector pattern_3 : SequencePattern = new SequencePattern extended with {
    Role input : Origin = new Origin extended with {
    }
    Role output : Destination = new Destination extended with {
    }
  }
  Attachment random_2.o_return to pattern_1.input;
  Attachment sum.i_a to pattern_1.output_1;
  Attachment sub.i_b to pattern_1.output_2;
  Attachment random_1.o_return to pattern_2.input;
  Attachment sum.i_b to pattern_2.output;
  Attachment sum.o_return to pattern_3.input;
  Attachment sub.i_a to pattern_3.output;
}
```

Listagem 3.1: Código Acme do Workflow WISP

Ao aplicar o mecanismo de transformações de WISP para os SGWfCs Taverna, Kepler e VisTrails (operação `fromWISP()`) é possível identificar algumas particularidades de cada SGWfC que serão apresentadas nas seções seguintes.

3.2.1 TRANSFORMAÇÃO WISP PARA TAVERNA

O workflow ilustrado pela Figura 3.9 foi o resultado da transformação WISP para Taverna. Nesse exemplo, é possível identificar que uma grande quantidade de tarefas foi adicionada, em relação ao workflow original especificado em WISP. Essas tarefas representam dois comportamentos peculiares do SGWfC Taverna, onde valores constantes, associados a portas de entradas das tarefas, são representados como outras tarefas, e onde cada tarefa do tipo serviço Web pode possuir tarefas auxiliares, responsáveis por converter o XML resultante das entradas e saídas dessas tarefas (envelopes SOAP), em parâmetros da tarefa. Nessa transformação, nenhuma intervenção manual é necessária para executar o workflow,

pois o Taverna não possui nenhuma configuração específica para esses tipos de tarefas.

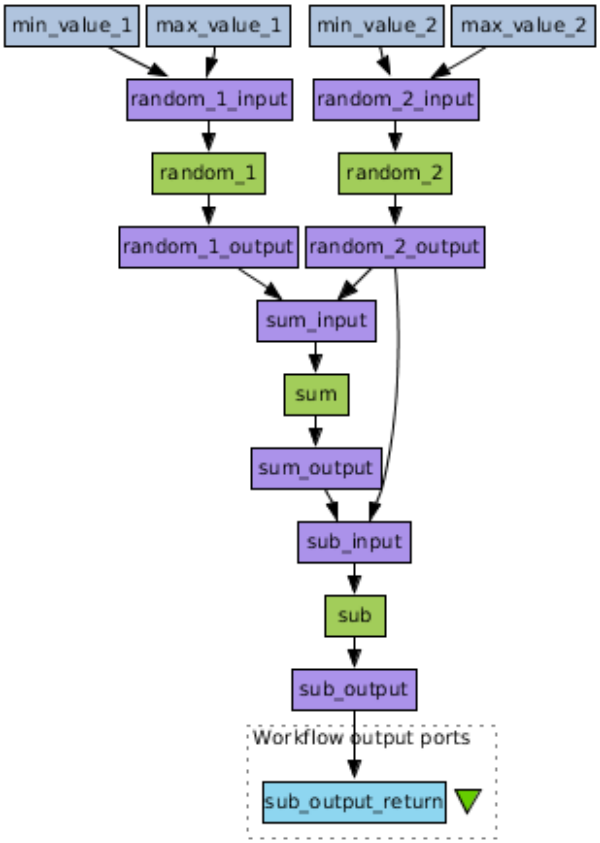


Figura 3.9: Exemplo de Workflow - Taverna

3.2.2 TRANSFORMAÇÃO WISP PARA KEPLER

No SGWfC Kepler, o conceito de diretores (*Directors*) é empregado em todos os workflows que são executados pelo SGWfC. Esses diretores são responsáveis por controlar a orquestração e execução do workflow e possuem diferentes implementações, como por exemplo, diretores que controlam execuções de workflows síncronos (ex. `SDF Director`) ou paralelos (ex. `PN Director`). A escolha desses diretores deve ser feita pelo usuário do SGWfC, antes da execução do workflow. A Figura 3.10 ilustra o resultado da transformação WISP para Kepler, onde o workflow resultante não possui nenhum diretor. Embora valores constantes associados a portas de entradas também sejam representados como tarefas nesse SGWfC, não é necessária a criação de uma tarefa auxiliar para tratar as entradas e saídas das tarefas do tipo serviço Web, quando esses serviços trabalharem apenas com tipos pri-

mitivos ou strings. Para serviços Web complexos [5] o SGWfC Kepler também faz uso de tarefas auxiliares para representar os atributos dos objetos como parâmetros da tarefa.

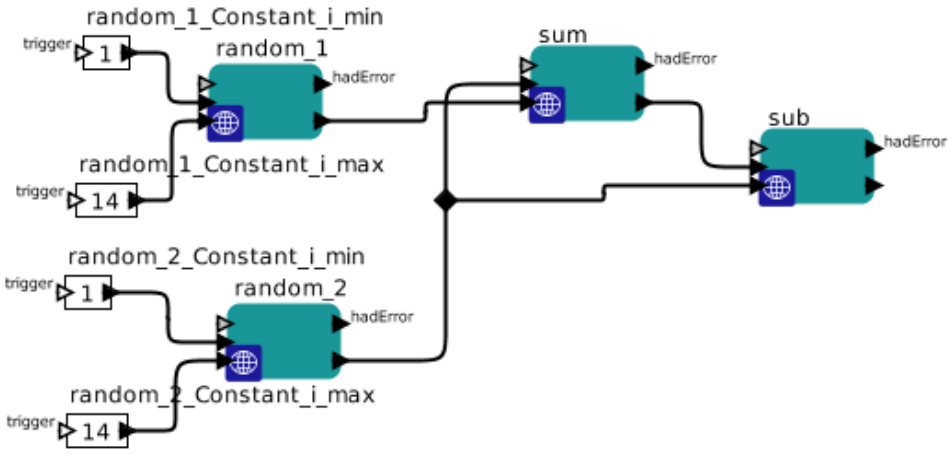


Figura 3.10: Exemplo de Workflow - Kepler

Ainda no contexto do SGWfC Kepler, é importante lembrar que a linguagem WISP não dá suporte a informações semânticas relacionadas à orquestração de um workflow. Diretores que orquestrem a execução de tarefas com um comportamento independente das conexões de um workflow (podendo ser um comportamento de execução de workflows distribuídos, por exemplo) devem ser manualmente adicionados por um usuário, pois não serão representados na linguagem WISP.

3.2.3 TRANSFORMAÇÃO WISP PARA VISTRAILS

O SGWfC VisTrails também possui particularidades que devem ser tratadas manualmente por um usuário, como a ativação de um módulo responsável por executar serviços Web. Porém, essa ativação é requisitada automaticamente pelo SGWfC e não depende de nenhuma configuração prévia para rodar esses serviços, o que torna este processo simples.

A Figura 3.11 ilustra o resultado da transformação WISP para VisTrails. Workflows VisTrails também podem ser executados diretamente (como workflows Taverna), sem a necessidade de edição do workflow e, embora também possuam tarefas constantes, apenas os dados de saída de uma tarefa do tipo serviço Web devem ser tratados por uma tarefa auxiliar e não os dados de entrada. No caso de serviços Web complexos, tanto os dados de entrada quanto os de saída devem ser tratados, podendo ter mais do que uma tarefa para

[5] Adotamos o termo **serviços Web complexos** para serviços Web que utilizem objetos como entrada ou saída para suas operações, ao invés de apenas tipos primitivos ou strings.

esse tratamento, de forma aninhada (de acordo com a composição do tipo complexo).

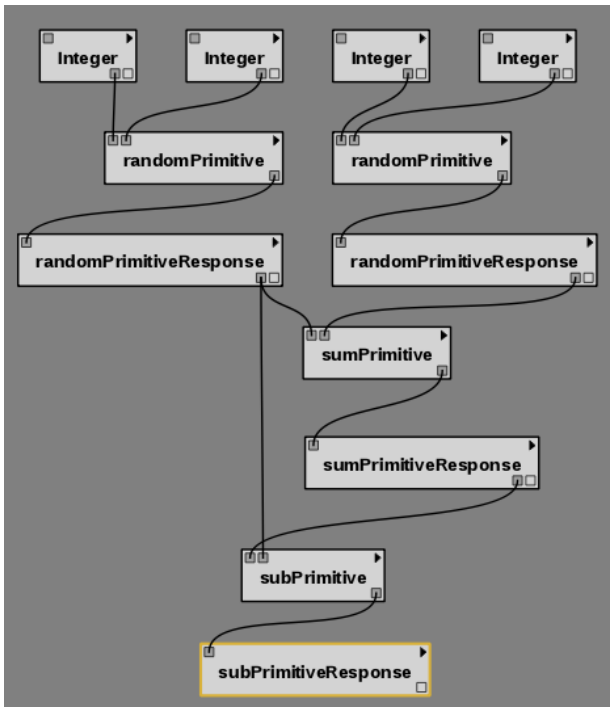


Figura 3.11: Exemplo de Workflow - VisTrails

3.2.4 CONSIDERAÇÕES INICIAIS SOBRE O USO DO FRAMEWORK

É importante lembrar que a linguagem WISP visa intercambiar o maior número de SGWfCs possível, e não carrega consigo comportamentos particulares de um SGWfC específico. As implementações dos SGWfCs Taverna, Kepler e VisTrails dão suporte nativo a tarefas do tipo serviços Web e o suporte a serviços Web foi uma maneira de validar inicialmente a linguagem proposta com o exemplo simples da Figura 3.8, assumindo-se que esse tipo de tarefa é representativo de cenários reais. O Capítulo 4 apresenta uma avaliação mais detalhada da proposta, incluindo uma análise da representatividade de tarefas do tipo serviços Web.

Na seção seguinte é detalhado o uso do framework desenvolvido para a utilização da linguagem WISP programaticamente, com exemplos da criação de workflows e o uso do engenho criado para executar as transformações.

3.3 IMPLEMENTAÇÃO

Considerando a necessidade do framework facilitar o suporte a novos SGWfCs (e novas versões de SGWfCs), foi criado um engenho cuja função é armazenar referências às implementações da interface `Parser`. Esse engenho utiliza o mecanismo de reflexões da plataforma Java e armazena as instâncias dessas interfaces de maneira textual, criando novas instâncias por demanda, quando são solicitadas. A Figura 3.12 ilustra a arquitetura do engenho que foi criado, onde desenvolvedores publicam as implementações concretas da interface `Parser` que são armazenadas. Essas implementações são identificadas pelo nome do SGWfC e sua versão, e podem ser acessadas por diferentes consumidores, podendo ser usuários que desejam transformar especificações de workflow científico em outras, ferramentas que tenham interesse em utilizar desse mecanismo de transformação automaticamente ou até mesmo outros desenvolvedores que desejam utilizar essa transformação como parte do desenvolvimento de seus aplicativos.

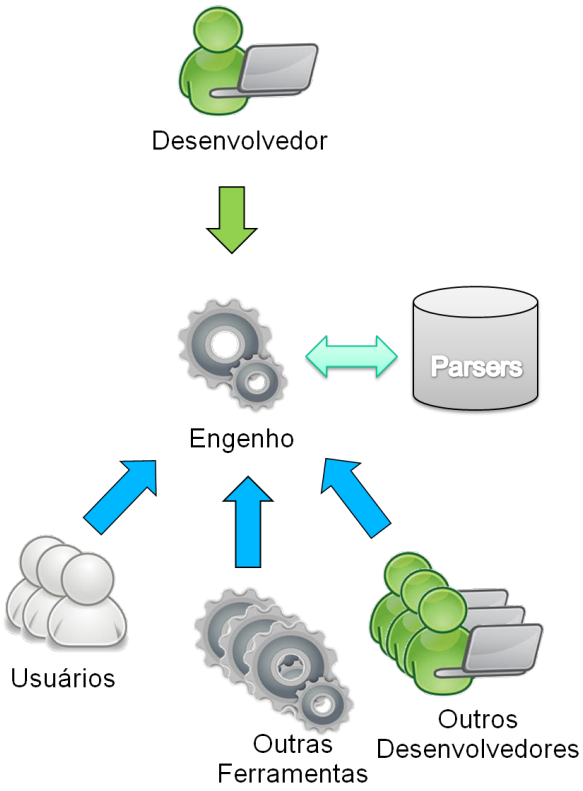


Figura 3.12: Engenho responsável por armazenar e prover instâncias da interface `Parser`

O framework desenvolvido oferece três classes utilitárias (Figura 3.13) denominadas `WISPReader`, `WISPWriter` e `WISPPatternFinder`, que possuem métodos para a geração de

um objeto da classe `Workflow` a partir de uma especificação WISP, e vice-versa, e identificação de padrões a partir das conexões de um workflow científico (classe `Connection`).

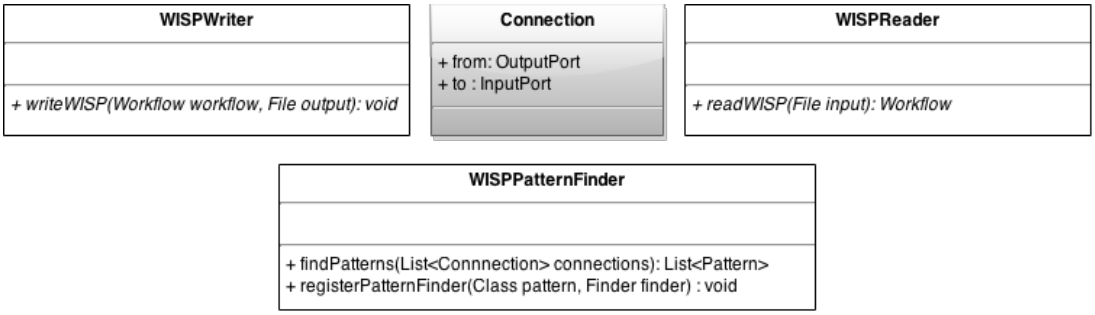


Figura 3.13: Classes Utilitárias da Linguagem WISP

A classe utilitária `WISPPatternsFinder` executa instâncias da interface `Finder` para cada um dos padrões existentes na linguagem WISP, utilizando conexões ponto a ponto (conexões que ligam uma porta de saída a uma porta de entrada), para sua identificação. Atualmente, o engenho possui implementações concretas para os padrões *Sequence*, *Parallel Split* e *Simple Merge* (Figura 3.14), que são padrões estruturais. Para a identificação dos padrões *Exclusive Choice* e *Synchronization*, que são padrões comportamentais, a implementação de uma nova classe é necessária.

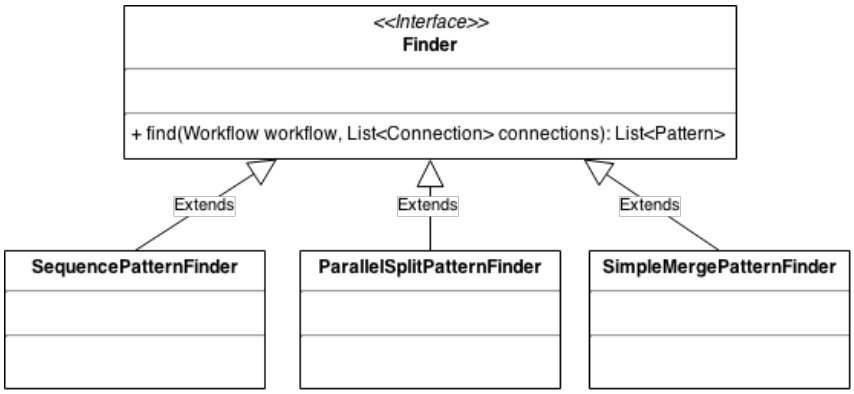


Figura 3.14: Implementação de Estratégias para Identificar Padrões Estruturais

Padrões com comportamento associado devem implementar um algoritmo para sua identificação. Esses algoritmos podem ser implementados através de uma estratégia (classe `Finder` que implementa o padrão de projeto *Strategy* (GAMMA et al., 1995)) que deve ser adicionada à esse utilitário através do método estático `registerPatternFinder` da classe `WISPPatternFinder`. Uma vez que as implementações concretas disponibilizadas pelo framework assumem algumas premissas com relação ao comportamento orientado a dados dos workflows científicos, que podem não serem válidas para todos os SGWfCs, é

possível que as estratégias para identificação de padrões estruturais também tenham que ser modificadas. A definição dessas estratégias, assim como a implementação do `Parser` do SGWfC, deve ser feita por um desenvolvedor que tenha amplo conhecimento do SGWfC em questão.

A identificação do padrão *Exclusive Choice* no SGWfC Taverna pode ser utilizada para ilustrar o uso da classe utilitária `WISPPatternFinder`. O SGWfC Taverna não provê tarefas ou conexões que realizem um comportamento exclusivo (*ifs* ou *switchs*). Porém, como já dito, esse comportamento pode ser atingido através de tarefas do tipo *BeanShell* que são proprietárias desse SGWfC. Para identificar esse padrão, a estratégia (`Finder`) implementada para o padrão *Exclusive Choice* executa todas as tarefas do tipo *BeanShell*, encontradas em uma especificação de workflow Taverna, que possuam a estrutura de um padrão *Exclusive Choice*, passando como entrada um determinado intervalo de valores, e identificando se apenas uma das portas de saída dessa tarefa gerou dados. Se esse comportamento se realizar para **todos** os valores no intervalo informado, essa tarefa será marcada como um padrão *Exclusive Choice*.

Uma estratégia parecida com a anterior também foi desenvolvida para o padrão *Synchronization*, porém, a verificação feita em sua única porta de saída é se os valores dados como entrada para o padrão foram de alguma maneira concatenados (podendo também serem agrupados em uma coleção). Os valores utilizados como entrada para a identificação dos padrões *Exclusive Choice* e *Synchronization* pertencem a um conjunto pequeno de dados de teste gerados aleatoriamente. Como incremento futuro pode-se empregar ferramentas como Randoop [6] que geram grande quantidade de dados aleatórios automaticamente e que poderiam ser utilizados como entrada para a identificação desses padrões.

3.3.1 INTEGRAÇÃO COM O ARCABOUÇO ECOS-COLLABORATIVE PL-SCIENCE

O arcabouço ECOS-Collaborative PL-Science é um ecossistema de software criado para apoiar as atividades de cientistas durante o ciclo de vida de um workflow científico (PEREIRA, 2014; WEIDT et al., 2015). O trabalho apresentado por (BELLOUM et al., 2011) define que o ciclo de vida colaborativo de um workflow científico é um processo interativo dividido em quatro passos: (i) investigação do problema, (ii) prototipagem do

[6] `http://mernst.github.io/randoop/`

experimento, (iii) execução do experimento e (iv) resultados e publicações. Baseado nessa definição, o ciclo de vida do arcabouço ECOS-Collaborative PL-Science é apresentado na Figura 3.15.

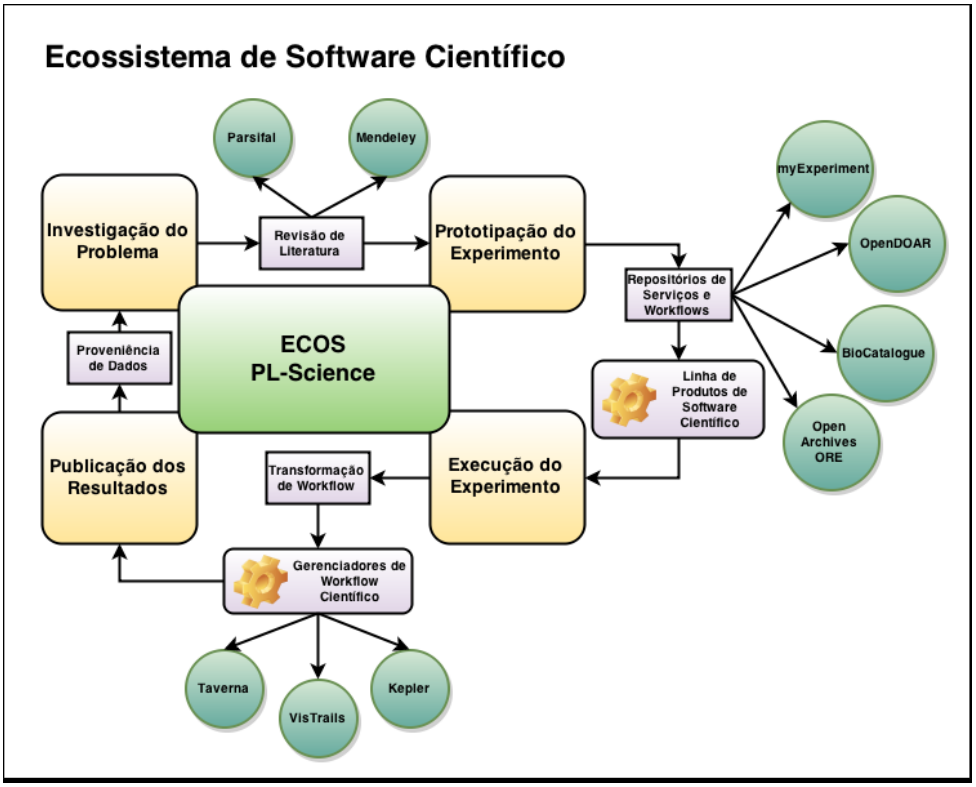


Figura 3.15: Ciclo de Vida de um Experimento Científico no ECOS-Collaborative PL-Science

Na investigação do problema, cientistas procuram por experimentos semelhantes aos seus, interagem com outros cientistas, definem suas metas e dividem o experimento em partes menores. Neste passo, uma revisão sistemática pode ser feita, a fim de apoiar outros cientistas. Na prototipagem do experimento, cientistas constroem um protótipo criando um workflow científico utilizando componentes disponíveis no próprio ECOS-Collaborative PL-Science ou em repositórios públicos (ex: myExperiment). Na execução do experimento, o workflow é gerado na linguagem intermediária proposta nessa dissertação. Neste passo, o workflow pode ser convertido para a especificação de um dos SGWfCs Taverna, Kepler ou VisTrails e posteriormente executado. No último passo, cientistas podem analisar os resultados de seus experimentos e publicá-los.

A Figura 3.16 ilustra a arquitetura do arcabouço ECOS-Collaborative PL-Science. Ao utilizar esse arcabouço novas aplicações podem ser criadas, compostas de diferentes serviços, e disponibilizadas como novos artefatos de uma linha de produto de Software.

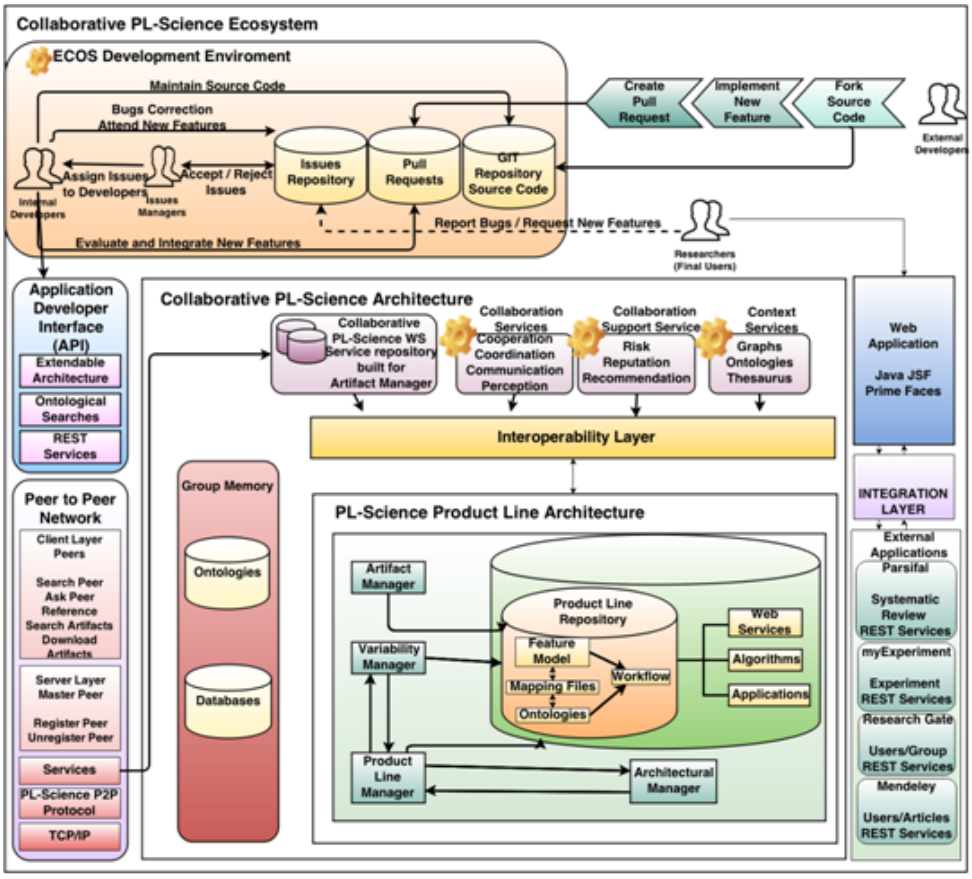


Figura 3.16: Arquitetura do ECOS-Collaborative PL-Science

3.3.2 CAPACIDADE DE ESCRITA DO FRAMEWORK DA LINGUAGEM WISP

Como já dito, para transformar especificações de workflows científicos para a linguagem WISP é necessário que o desenvolvedor conheça a semântica dessa linguagem. Nessa seção são apresentados alguns exemplos para facilitar o aprendizado da API oferecida pelo framework da linguagem WISP, que foi desenvolvida utilizando a linguagem de programação Java.

Ao implementar um novo `Parser`, o desenvolvedor deve ter conhecimento dos métodos disponibilizados pela API. Para isso um JavaDoc é disponibilizado em `https://www.lncc.br/sinapad/wisp-api`.

3.3.2.1 Criação de um workflow através de um código Java

O workflow ilustrado na Figura 3.17 pode ser representado pelo código da Java 3.2. Nesse código instâncias de tarefas, portas, workflows internos e padrões são criadas e suas cone-

xões são atribuídas ao objeto principal (classe `Workflow`) cuja instância tem o nome de **workflow**. Após fazer todas as ligações necessárias, utiliza-se a classe `WISPWriter` para escrever a estrutura Java em um arquivo Acme, utilizando a família linguagem WISP. É possível notar que esse código Java necessário para representar esse workflow é bastante verboso, mesmo que represente um workflow pequeno. Recomenda-se o uso de padrões (como o padrão *Builder* (GAMMA et al., 1995)) para a criação de instâncias desses workflows a partir da leitura da especificação do SGWfC a fim de reduzir o código necessário para a leitura da especificação do SGWfC de destino para a linguagem WISP.

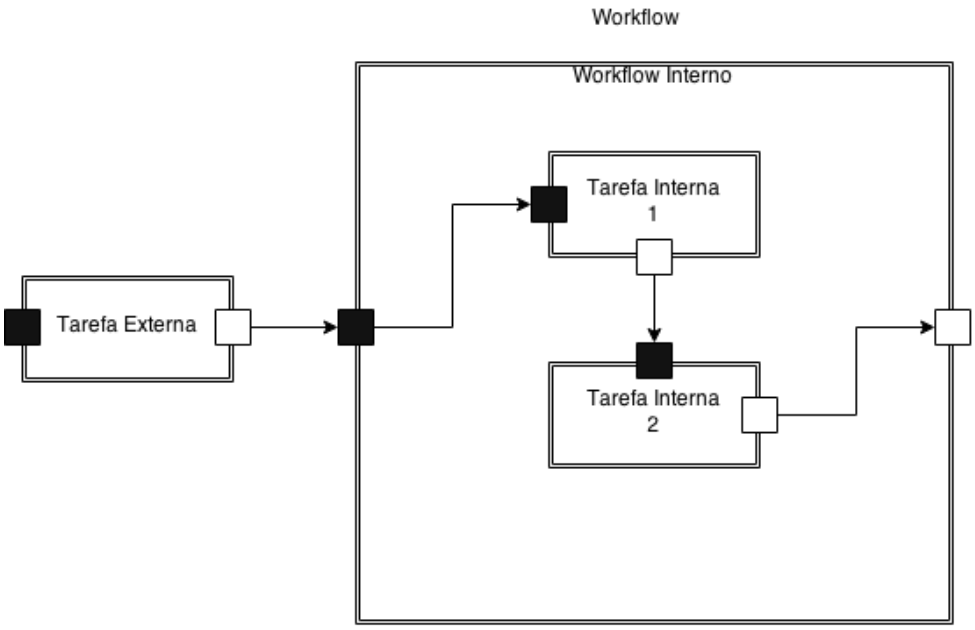


Figura 3.17: Exemplo de Workflow

```
Workflow workflow = new Workflow();
workflow.setName("Workflow");
Task outerTask = new Task("Outer Task 1");
Port outerTaskInputPort = new Port("input");
Port outerTaskOutputPort = new Port("output");
outerTask.addInputPort(outerTaskInputPort);
outerTask.addOutputPort(outerTaskOutputPort);

Workflow innerWorkflow = new Workflow();
innerWorkflow.setName("Workflow Interno");
Port innerWorkflowInputPort = new Port("input");
Port innerWorkflowOutputPort = new Port("output");
innerWorkflow.addInputPort(innerWorkflowInputPort);
innerWorkflow.addOutputPort(innerWorkflowOutputPort);

Task innerTask1 = new Task("Tarefa Interna 1");
Task innerTask2 = new Task("Tarefa Interna 2");
```

```
Port innerTask1InputPort = new Port("input");
Port innerTask1OutputPort = new Port("output");
innerTask1.addInputPort(innerTask1InputPort);
innerTask1.addOutputPort(innerTask1OutputPort);

Port innerTask2InputPort = new Port("input");
Port innerTask2OutputPort = new Port("output");
innerTask2.addInputPort(innerTask2InputPort);
innerTask2.addOutputPort(innerTask2OutputPort);

SequencePattern p1 = new SequencePattern();
p1.setInput(innerTask1OutputPort);
p1.setOutput(innerTask2InputPort);

Binding binding1 = new Binding();
binding1.setOuter(innerWorkflowInputPort);
binding1.setInner(innerTask1InputPort);

Binding binding2 = new Binding();
binding2.setOuter(innerWorkflowOutputPort);
binding2.setInner(innerTask2OutputPort);

innerWorkflow.addTask(innerTask1);
innerWorkflow.addTask(innerTask2);
innerWorkflow.addPattern(p1);
innerWorkflow.addBinding(binding1);
innerWorkflow.addBinding(binding2);

SequencePattern p2 = new SequencePattern();
p2.setInput(outerTaskOutputPort);
p2.setOutput(innerWorkflowInputPort);

workflow.addInputPort(outerTaskInputPort);
workflow.addOutputPort(innerWorkflowOutputPort);
workflow.addTask(outerTask);
workflow.addTask(innerWorkflow);
workflow.addPattern(p2);

WISPWriter.writeWISP(workflow, new File(".../Workflow.acme"));
```

Listagem 3.2: Código Java para Criação de um Workflow

3.3.3 UTILIZAÇÃO DO ENGENHO DO MECANISMO DE TRANSFORMAÇÕES

Antes de utilizar o engenho do mecanismo de transformações é necessário que as implementações da interface `Parser` sejam registradas nesse engenho, como ilustrado no código 3.3. As implementações para os SGWfCs Taverna, Kepler e VisTrails são registradas, assim como suas versões. Esse engenho é implementado como um padrão *Singleton* (GAMMA et al., 1995), para deixar essas instâncias disponíveis para qualquer aplicação cliente.

```
Engine engine = Engine.getInstance();
engine.registerParser("Kepler", "2.4",
        "br.ufjf.pgcc.framework.sgwfc.kepler.v2_4.parser.KeplerParser");
engine.registerParser("Taverna", "2.4",
        "br.ufjf.pgcc.framework.sgwfc.taverna.v2_4.parser.TavernaParser");
engine.registerParser("VisTrails", "2.1",
        "br.ufjf.pgcc.framework.sgwfc.vistrails.v2_1.parser.VisTrailsParser");
```

Listagem 3.3: Código Java para Registro de Implementações da Interface `Parser`

O código 3.4 ilustra a implementação de uma transformação entre dois SGWfCs, passando pela linguagem WISP. Nesse exemplo uma especificação de workflow desenvolvida no SGWfC VisTrails é convertida para uma especificação de workflow do SGWfC Kepler.

```
Parser vistrailsParser = engine.retrieveParser("VisTrails", "2.1");
File vistrailsInput = new File("../Workflow.xml");
File vistrailsOutput = new File("../Vistrails_Kepler_Workflow.acme");
vistrailsParser.toWISP(vistrailsInput, vistrailsOutput);

Parser keplerParser = engine.retrieveParser("Kepler", "2.4");
File keplerInput = new File("../Vistrails_Kepler_Workflow.acme");
File keplerOutput = new File("../Vistrails_Kepler_Workflow.xml");
keplerParser.fromWISP(keplerInput, keplerOutput);
```

Listagem 3.4: Código Java para Transformação do SGWfC VisTrails para o SGWfC Kepler

Nesse código são executados 4 passos seguintes:

1. A instância da interface `Parser` para o SGWfC VisTrails na versão 2.1 que foi previamente registrada no engenho é obtida;

2. O arquivo **Workflow.xml** que contém a especificação do workflow no SGWfC VisTrails é utilizado como entrada para o método `toWISP()` do `parser` do SGWfC

VisTrails, assim como o arquivo de saída **Vistrails_Kepler_Workflow.acme** que será gerado;

3. A instância da interface `Parser` para o SGWfC Kepler na versão 2.4 que foi previamente registrada no engenho é obtida;

4. O arquivo **Vistrails_Kepler_Workflow.acme** que contém a especificação do workflow em WISP é utilizado como entrada para o método `fromWISP()` do `parser` do SGWfC Kepler, assim como o arquivo de saída **Vistrails_Kepler_Workflow.xml** que será gerado.

Após esses passos o arquivo **Vistrails_Kepler_Workflow.xml**, contendo uma especificação do workflow em Kepler, poderá ser aberto nesse SGWfC e o usuário poderá realizar as mudanças necessárias (se necessárias) para sua execução. Essas mudanças devem ocorrer em casos onde tarefas que não sejam intercambiadas através de da linguagem WISP sejam representadas como tarefas vazias (*placeholders*). Nesse caso, o usuário da ferramenta teria que modificar essas tarefas vazias por tarefas que de fato executem o comportamento esperado neste passo do workflow. A vantagem dessa transformação é que o workflow passa a poder ser executado em um ambiente conhecido do cientista que utilizou o engenho para a transformação.

3.3.4 PORTAL CIENTÍFICO

A fim de oferecer um serviço para usuários que desejam executar o intercâmbio de workflows científicos especificados em diferentes SGWfCs, um portal científico foi desenvolvido com a proposta de executar o intercâmbio entre especificações dos SGWfCs Taverna, Kepler e VisTrails. Esse portal pode ser acessado na URL `https://www.lncc.br/sinapad/wisp-portal/`.

Para utilizar o portal, o usuário deve selecionar o SGWfC de origem e sua versão, e o SGWfC de destino e sua versão, como ilustrado na Figura 3.18.

Após selecionar o SGWfC de origem e destino, o usuário poderá iniciar a transformação. A Figura 3.19 ilustra um indicador informando que a transformação está sendo processada.

Ao final do processamento dessa transformação o usuário poderá receber dois avisos. O primeiro aviso ilustrado pela Figura 3.20 será mostrado quando a transformação ocorrer

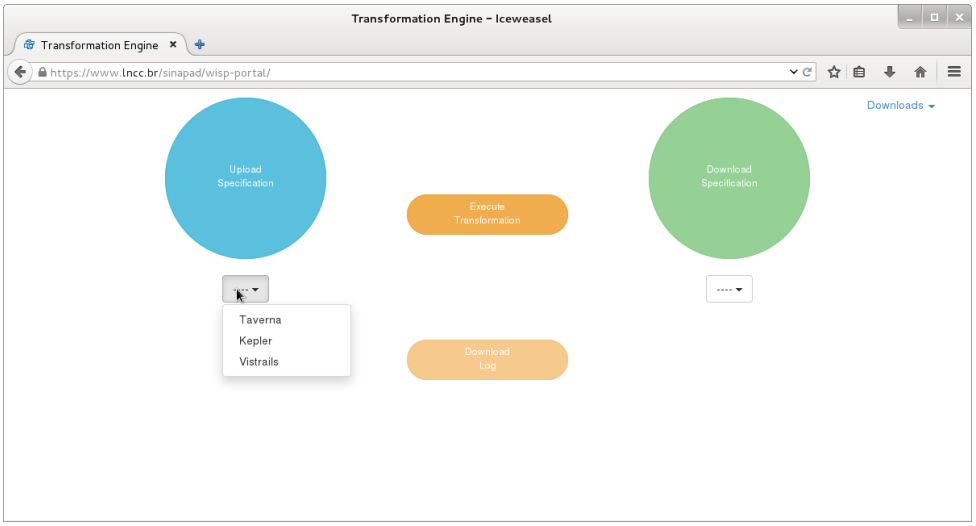


Figura 3.18: Página Inicial do Portal

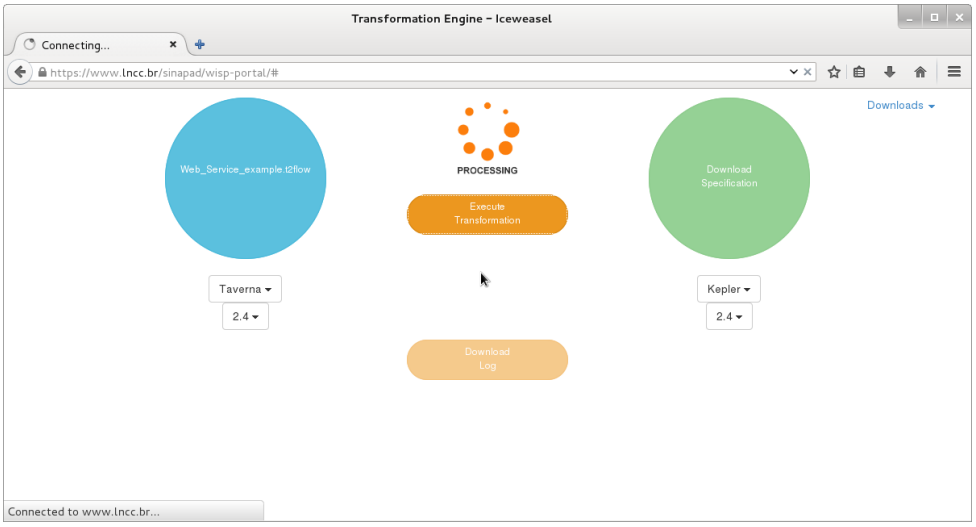


Figura 3.19: Execução da Transformação Através do Portal

com sucesso, e os botões **Download Specification** e **Download Log** serão habilitados, possibilitando que o usuário faça download dos arquivos resultantes. O segundo aviso ilustrado pela Figura 3.21 será mostrado caso haja algum erro na transformação, e somente o botão **Download Log** será habilitado, possibilitando que o usuário possa fazer download do arquivo de log e verifique o erro ao tentar realizar a transformação.

Os arquivos resultantes da transformação são ilustrados na Figura 3.22: a Figura 3.22(a) ilustra os arquivos gerados em uma transformação executada com sucesso e a Figura 3.22(b) ilustra os arquivos gerados em uma transformação executada com erro. Os arquivos resultantes são detalhados a seguir:

- `input_file` Especificação de workflow científico utilizado como entrada (no exem-

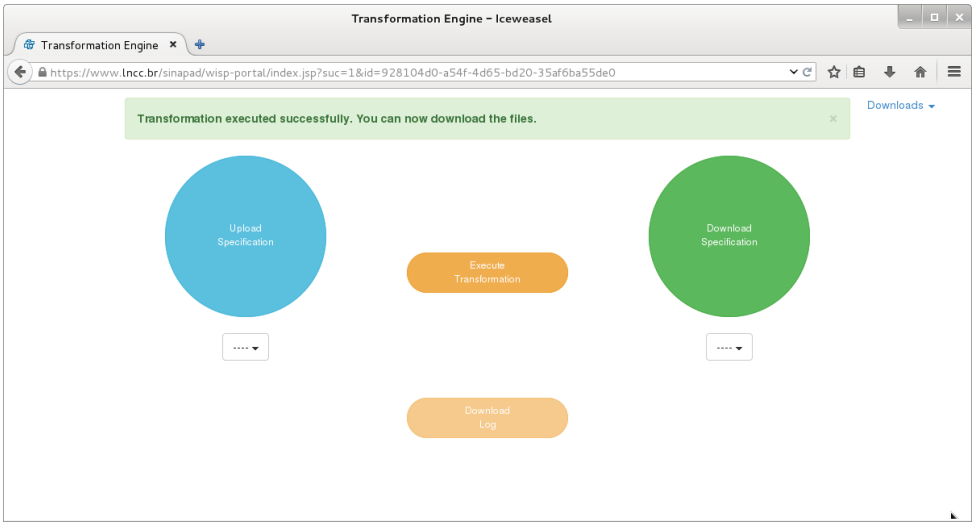


Figura 3.20: Mensagem de Sucesso após a Transformação

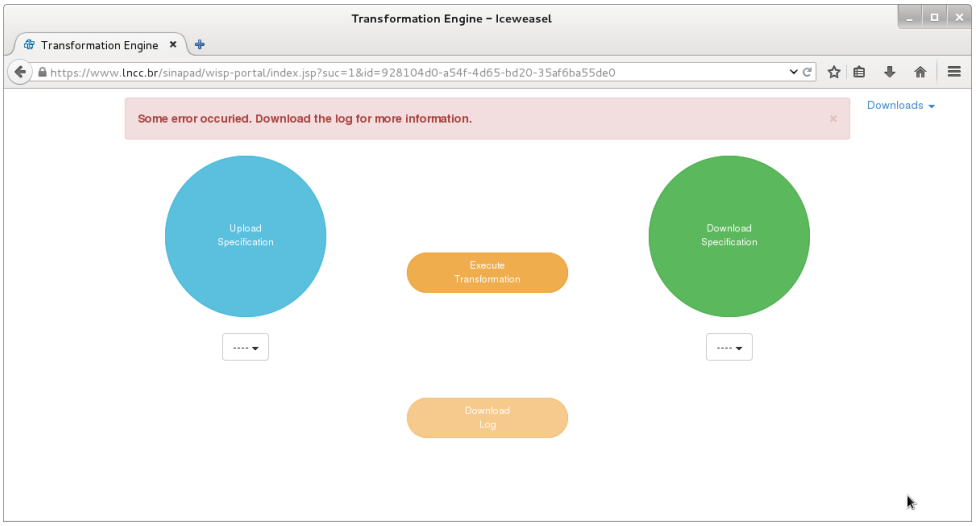
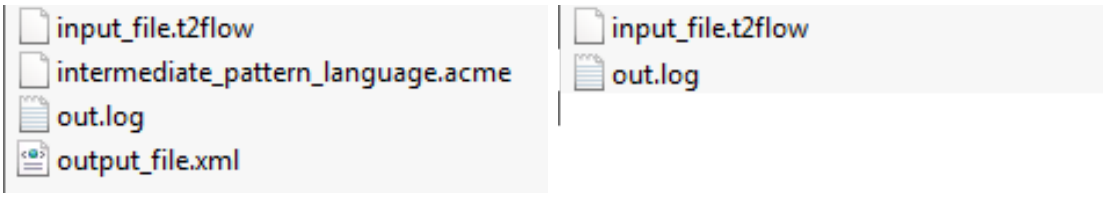


Figura 3.21: Mensagem de Erro após a Transformação

plo é um arquivo .t2flow que representa uma especificação de workflow científico desenvolvida no SGWfC Taverna)

- `intermediate_pattern_language.acme` Representação do workflow científico na linguagem WISP
- `out.log` Arquivo de log com informações sobre a transformação
- `output_file` Especificação de workflow científico gerada como saída (no exemplo é um arquivo .xml que representa uma especificação de workflow científico do SGWfC Kepler)

O portal também oferece uma opção onde um desenvolvedor pode fazer download da



(a) Aquivos Gerado na Execução com Sucesso (b) Arquivos Gerados na Execução com Erro

Figura 3.22: Arquivos Gerados pelo Portal após Execução

especificação Acme da família da linguagem WISP e de um workflow de exemplo composto inteiramente por serviços Web, como ilustrado na Figura 3.23.

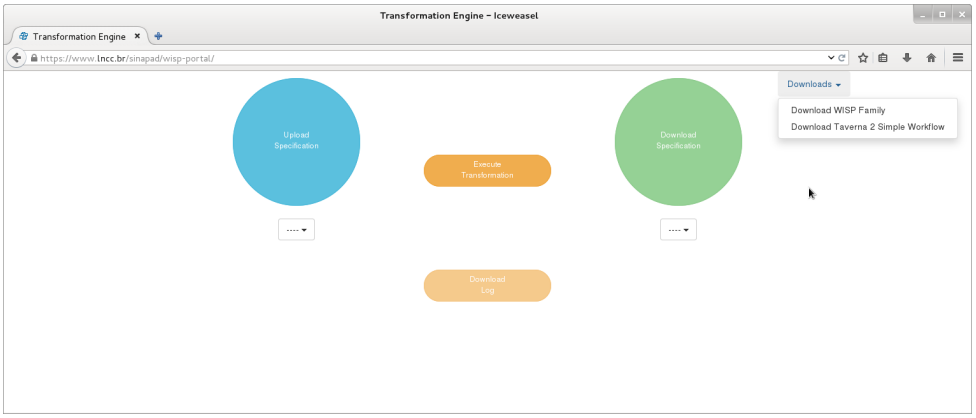


Figura 3.23: Menu para Usuários e Desenvolvedores

Esse portal é disponibilizado junto com um serviço REST para desenvolvedores que desejarem utilizar sua funcionalidade através de outras aplicações.

3.4 CONCLUSÕES DO CAPÍTULO

Este trabalho detalha uma pesquisa exploratória com o objetivo de identificar se é possível representar workflows científicos através de padrões utilizando-os para realizar o intercâmbio entre especificações de diferentes SGWfCs.

Para ilustrar um cenário funcional, o suporte a tarefas do tipo serviços Web foi adicionada à linguagem WISP, uma vez que serviços Web são compostos da localização do serviço (normalmente uma URL) e a operação que será realizada, trabalham sob um mesmo protocolo e são suportados por diversos SGWfCs. Ao adicionar esse tipo de tarefa, a intenção era reproduzir um workflow que só utilize tarefas do tipo serviços Web inteiramente em outro SGWfC e obter os mesmos resultados facilitando a validação da proposta, embora (como pode ser visto nos capítulos e seções seguintes), a maneira pela

qual serviços Web são tratados nos SGWfCs tenha sido, também, um grande desafio.

O capítulo detalhou o ferramental proposto por esta dissertação, que tem o objetivo de auxiliar a colaboração entre cientistas e grupos de pesquisa que utilizem diferentes SGWfCs para a construção de workflows. Através do ferramental proposto, um cientista pode intercambiar um workflow desenvolvido por outro para seu ambiente de trabalho conhecido, e realizar as mudanças necessárias (se necessárias) para a execução desse serviço.

Neste contexto, os exemplos de workflows apresentados nesse capítulo ajudaram a ilustrar o uso da linguagem WISP para a transformação de especificações de workflows científicos entre diferentes SGWfCs. Assim, consideramos que a proposta desta dissertação é útil para workflows científicos que utilizem tarefas conhecidas por um cientista, porém, que tenha sido desenvolvido em um SGWfC no qual esse cientista não possui nenhum conhecimento. Nesse cenário, embora esse workflow possa ser útil para a pesquisa desse cientista, o aprendizado de um novo SGWfC pode não ser trivial. Assim, mudanças necessárias nesse workflow para comprovar a validação de seus experimentos podem não ser facilmente implementadas. Quando um workflow científico é intercambiado para o SGWfC utilizado pelo cientista, as mudanças necessárias para que esse workflow esteja totalmente operacional serão em tese bem mais simples, e novas funcionalidades poderão ser adicionadas pelo próprio cientista a fim de validar novos experimentos.

Este capítulo apresentou ainda detalhes de implementação do ferramental proposto. No próximo capítulo serão apresentados alguns experimentos com workflows científicos reais utilizados pela comunidade científica e disponibilizados no repositório myExperiment. Esses experimentos visam identificar as vantagens e limitações da abordagem proposta.

4 AVALIAÇÃO

Este capítulo apresenta uma avaliação, considerando a utilização da linguagem WISP. A definição e o planejamento para esta avaliação são apresentadas na Seção 4.1. Para a avaliação, foi desenvolvido um protótipo, conforme detalhado no Capítulo 3, do ferramental computacional necessário nos processos de transformação de workflows especificados em diferentes SGWfCs de e para a linguagem de intercâmbio proposta neste trabalho.

Para esta avaliação, foi formulada a hipótese H_T que gerou a hipótese nula H_{T0} e a hipótese alternativa H_{T1} apresentadas a seguir:

- H_{T1} : O uso de padrões (*patterns*) combinados a conceitos de arquitetura de software para representar as informações semânticas mais importantes contidas nos workflows científicos permite auxiliar no estabelecimento de processos automáticos de transformação de especificações desses workflows entre diferentes SGWfCs, o que possibilita reduzir, em comparação com um intercâmbio manual, o esforço de um cientista para reutilizar workflows científicos desenvolvidos por outros grupos de pesquisa em SGWfCs que não fazem parte de seu ambiente de trabalho.

- H_{T0} : O uso de padrões (*patterns*) combinados a conceitos de arquitetura de software para representar as informações semânticas mais importantes contidas nos workflows científicos **NÃO** permite auxiliar no estabelecimento de processos automáticos de transformação de especificações desses workflows entre diferentes SGWfCs.

Para avaliar a hipótese será levado em consideração o percentual de workflows intercambiados, que foi adotado como métrica da redução de esforço obtida ao utilizar a abordagem proposta.

4.1 PLANEJAMENTO

Para realizar a avaliação, foram utilizados os SGWfCs: Taverna na versão 2.4, Kepler na versão 2.4 e VisTrails na versão 2.1. Estes SGWfCS foram escolhidos pois são os únicos SGWfCs de propósito geral com especificações disponibilizadas no repositório myExperiment (`http://www.myexperiment.org/`) e que também possuem um número considerável de resultados em uma busca no *Google Scholar*, como pode ser visto na Tabela 4.1.

SGWfCs	Número de Workflows	Número de Resultados	String de Busca
Taverna 2	1,482 [a]	5,250	taverna workflow
Kepler	46	5,750	kepler workflow
VisTrails	4	1,150	VisTrails workflow
Swift	0	8,240	swift workflow system
Triana	0	1,860	taverna workflow
ASKALON	0	1,030	askalon workflow
WS-PGRADE	0	229	ws-pgrade workflow

[a]Embora o myExperiment possua 1482 workflows Taverna 2, 30 desses workflows estão privados e 1,452 possuem acesso público.

Tabela 4.1: Número de Workflows no myExperiment e Resultados da Busca no Google Scholar (Jun/15)

Para obter os workflows utilizados nos experimentos apresentados neste capítulo, foi utilizado um script para fazer download de todos os workflows públicos dos SGWfCs Taverna 2, Kepler e VisTrails do repositório myExperiment. Os workflows obtidos foram coletados em Junho de 2015.

Para a execução da avaliação foram utilizadas as seguintes configurações em um computador pessoal:

- Sistema Operacional - Linux Debian 7 Squeeze
- Processador - Intel i3
- Memória Ram - 6Gb DDR3

4.2 EXECUÇÃO

Para a condução da avaliação, o protótipo do mecanismo de transformações foi utilizado para transformar as especificações para WISP considerando todos os workflows obtidos e foi realizada uma análise automática dos resultados com relação ao tipo de suas tarefas. Como a linguagem WISP se preocupa em intercambiar a estrutura dos workflows científicos, foram selecionados alguns workflows para uma análise manual [1] e validação dos padrões encontrados.

[1]Por análise manual entende-se a abertura do workflow no SGWfC de destino e verificação da estrutura do workflow e tentativa de execução.

Uma vez que workflows científicos foram desenvolvidos para diferentes áreas de *e-science*, era necessário que os workflows analisados manualmente possuíssem uma descrição de suas entradas, para que a tentativa de sua reprodução fosse possível. Além disso, em casos onde workflows não eram totalmente intercambiáveis, é necessário saber o que será executado em tarefas que não forem intercambiadas em um determinado trecho do workflow. Para mitigar esse problema, a tentativa de reprodução de workflow em um SGWfC diferente ao qual este foi desenvolvido só aconteceu com workflows que utilizavam apenas serviços Web, constantes ou workflows internos em sua composição.

O repositório myExperiment conta com diversas implementações de workflows em diferentes SGWfCs, sendo em sua maioria workflows desenvolvidos no SGWfC Taverna. Os workflows dos SGWfCs Kepler e VisTrails, disponíveis no myExperiment 4.1, foram executados. Esse primeiro passo mostrou que os 4 workflows VisTrails se propõem a executar o mesmo workflow, porém, em versões diferentes. Entretanto, a tarefa principal dos workflows é um serviço Web que não está mais disponível, o que impossibilita a reprodução desses workflows. Já ao executar os 46 workflows desenvolvidos no SGWfC Kepler, foi possível identificar 278 tarefas de 45 tipos diferentes, porém, somente uma tarefa era um workflow interno, 88 eram constantes e nenhuma delas era um serviço Web. Embora a estrutura desses workflows possa ser reproduzida em WISP, a execução desses workflows em outros SGWfCs não seria possível, então workflows dos SGWfCs VisTrails e Kepler não foram utilizados na amostragem inicial das transformações.

Como a quantidade de workflows científicos Taverna é muito maior que a dos outros SGWfCs, esses workflows não foram executados manualmente e esta será a amostragem utilizada para analisar o framework proposto. Com esta análise supõe-se avaliar também se seria possível reproduzir a estrutura e a troca de dados entre todas as tarefas de todos os workflows científicos compreendidos nesse experimento. Porém, considerando que será necessário fazer uma intervenção manual em especificações que não forem inteiramente compostas de tarefas intercambiáveis para validar o resultado. No contexto deste trabalho o termo “tarefas intercambiáveis” será utilizado para tarefas representadas como workflows internos, serviços Web ou constantes.

4.3 ANÁLISE DAS HIPÓTESES

4.3.1 HIPÓTESE H_T

Ao executar o engenho proposto neste trabalho, considerando os 1.452 workflows Taverna, foi possível identificar que existem um total de 16.492 tarefas. Dentre essas tarefas, 8.761 (53%), são nativamente representáveis na linguagem WISP. A Tabela 4.2 ilustra os tipos de tarefas mais comuns nesses workflows. Foi possível identificar também que tarefas intercambiáveis através da linguagem WISP ocupam o topo da lista, porém, tarefas do tipo `LocalworkerActivity` que são serviços locais proprietários do SGWfC Taverna e tarefas do tipo `BeanshellActivity` que são scripts Taverna desenvolvidos por usuários (na linguagem de programação Java), também são muito utilizadas e não são nativamente intercambiáveis para a linguagem WISP. O Apêndice B.2 apresenta uma tabela com todos os tipos de tarefas encontradas nos workflows do SGWfC Taverna. No total são mais de 100 tipos diferentes de tarefas encontradas nesses 1.452 workflows.

Tarefa Taverna	Representação WISP	Quantidade
`StringConstantActivity`	Valor atribuído a uma porta de entrada	3.902
`LocalworkerActivity`	Tarefa do tipo "Task"	2.836
`BeanshellActivity`	Tarefa do tipo "Task"	2.363
`WSDLActivity`	Tarefa do tipo "WebServiceTask"	1.466
`DataflowActivity`	Representação interna de uma tarefa	1.285
`XMLInputSplitterActivity`	Omitida, pois é uma tarefa auxiliar	1.212
`XMLOutputSplitterActivity`	Omitida, pois é uma tarefa auxiliar	896
Outras	Tarefa do tipo "Task"	7.731
Total		**16.492**

Tabela 4.2: Tarefas dos Workflows Taverna Disponíveis no Repositório myExperiment

As tarefas `StringConstantActivity` representam constantes que são passadas como entrada para algumas tarefas. Na linguagem WISP essas constantes são representadas como valores atribuídos a portas de entrada, e não como tarefas. As tarefas `DataflowActivity` são workflows internos, que são tratados como tarefas pelo SGWfC Taverna e na linguagem WISP são tarefas com representações internas associadas. Essas representações internas podem conter outras tarefas (componentes) e padrões (conectores e papéis), assim como a representação principal do workflow e são um elemento de modelagem naturalmente presente na ADL Acme (GARLAN et al., 1997). As tarefas `WSDLActivity` são serviços Web e possuem informações do endereço do WSDL e a operação que será exe-

#porcentagem	#workflows
100%	179
90-99.99%	26
80-89.99%	73
70-79.99%	82
60-69.99%	99
50-59.99%	164
40-49.99%	101
30-39.99%	149
20-29.99%	111
10-19.99%	40
0-9.99%	428

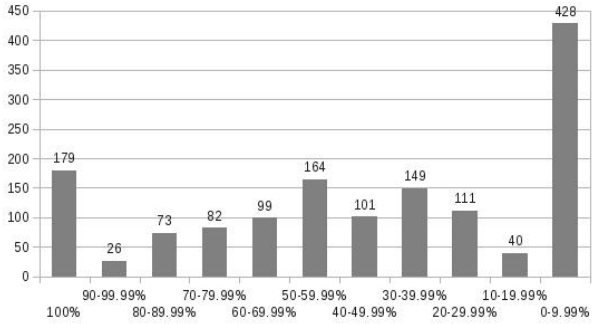


Tabela 4.3: Porcentagem de Workflows Intercambiáveis

cutada e são representadas como tarefas do tipo `WebServiceTask` na linguagem WISP. A diferença entre tarefas do tipo `Task` e `WebServiceTask` na linguagem WISP é que as tarefas do tipo `WebServiceTask` devem possuir obrigatoriamente o valor do endereço do WSDL e operação do serviço Web. As tarefas `XMLInputSplitterActivity` e `XMLOutputSplitterActivity` são tarefas auxiliares, utilizadas em conjunto com serviços Web, cuja função é mapear os parâmetros de entrada e saída de tarefas `WSDLActivity`, pois o protocolo SOAP, sob o qual os serviços Web trabalham, envia e recebe mensagens no formato XML. Essas tarefas auxiliares não são representadas na linguagem WISP como tarefas, mas sim como portas de tarefas.

Dentre os 1.452 workflows Taverna, 179 (12%) deles utilizam apenas tarefas que podem ser representadas na linguagem WISP sem perda de informação semântica e reproduzidas nos SGWfCs VisTrails e Kepler. Contudo, é importante destacar que, embora serviços Web devam ser interoperáveis entre diferentes linguagens de programação, ainda é necessário que os SGWfCs envolvidos na transformação dêem suporte à versão correta do serviço Web (protocolo SOAP) que é utilizado na implementação concreta do serviço. Esse percentual corresponde portanto a um limite superior da métrica de redução de esforço adotada para avaliar a hipótese H_T .

A Tabela 4.3 ilustra a porcentagem de tarefas interoperáveis nos workflows Taverna.

Considerando as análises apresentadas acima e o universo dos workflows disponibilizados no repositório não é possível provar a falsidade da hipótese H_{T0} , uma vez que não é possível manter, de maneira automática, a informação semântica associada a workflows que não sejam compostos exclusivamente de tarefas do tipo “serviços Web”. Como pôde ser visto, embora tarefas intercambiáveis componham mais de 50% das tarefas encontradas

#tarefas	#workflows	#porcentagem
0	8	4,46%
1	15	8,37%
2	15	8,37%
3	55	30,72%
4	19	10,61%
5	13	7,26%
6	16	8,93%
7	3	3,91%
8	2	1,11%
9	2	1,11%
10	1	0,55%
> 10	30	16,75%

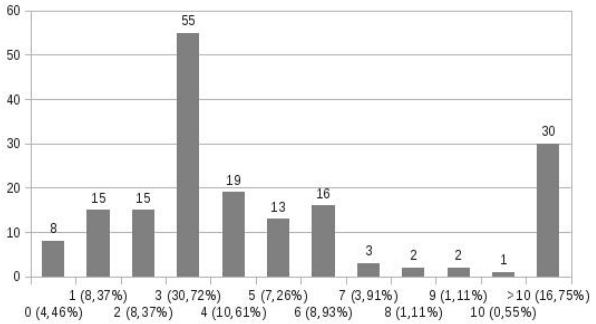


Tabela 4.4: Quantidade de Tarefas dos Workflows Intercambiáveis em Taverna

em todos os workflows analisados, o número de workflows compostos exclusivamente por estas tarefas é pequeno, o que impossibilita um processo automático de intercâmbio que poderia reproduzir um workflow em um SGWfC diferente do qual este foi desenvolvido sem nenhuma intervenção manual ou perda de informação semântica.

Além disso, ao analisar a quantidade de tarefas de todos os 179 workflows Taverna integralmente intercambiáveis em nossa abordagem, podemos observar que mais da metade desses workflows possui até 4 tarefas, o que é um número para o qual o processo manual de intercâmbio não é supostamente custoso para o cientista. Uma certa quantidade de workflows (16%) possui mais de 10 tarefas, o que pode tornar o processo manual de intercâmbio complexo, ainda mais para cientistas que não estejam familiarizados com o SGWfC de origem do workflow. Esses números podem ser vistos na Tabela 4.4.

Entretanto, ao fazer uma análise nesses mesmos 179 workflows, após sua conversão para WISP (vide Tabela 4.5), podemos identificar que o maior workflow é composto por 8 tarefas, enquanto a grande maioria desses workflows (67%) são compostos de apenas uma única tarefa, o que enfraquece ainda mais a hipótese alternativa H_{T1} , uma vez que um processo manual de intercâmbio de uma única tarefa em um SGWfC diferente não deve ser complexo. A diferença no número de tarefas entre especificações Taverna e WISP se deve ao fato de constantes e tarefas auxiliares utilizadas para o tratamento das entradas e saída de tarefas do tipo serviços Web não serem representadas em WISP. Desta forma, a hipótese inicial H_T foi refutada e reformulada, gerando uma hipótese mais fraca H_E , porém de resultado prático potencialmente útil, com será apresentado na Subseção 4.3.2.

É importante destacar que alguns workflows ilustrados nas tabelas anteriores são compostos de 0 tarefas. Nestes casos, esses workflows podem ser compostos de tarefas cons-

#tarefas	#workflows	#porcentagem
0	15	8,37%
1	121	67,59%
2	9	5,02%
3	6	3,35%
4	7	3,91%
5	9	5,02%
6	9	5,02%
7	0	0,00%
8	2	1,11%

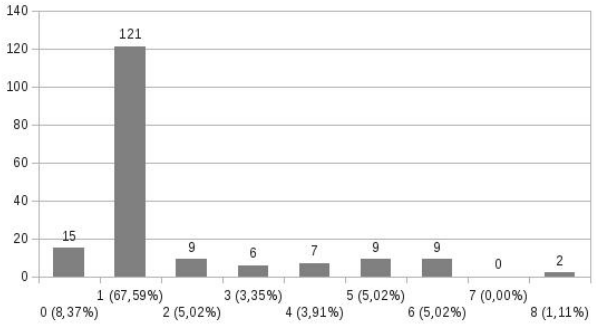


Tabela 4.5: Quantidade de Tarefas dos Workflows Intercambiáveis em WISP

tantes, que não são representadas em WISP, ou até mesmo workflows vazios que foram enviados para o repositório myExperiment, uma vez que este repositório não faz nenhuma validação nas especificações desses workflows em relação ao número de tarefas ou seus atributos.

4.3.2 HIPÓTESE H_E

A segunda hipótese deste trabalho, denominada H_E , gerou a hipótese nula H_{E0} e a hipótese alternativa H_{E1} apresentadas a seguir:

- H_{E1} : O uso de padrões (*patterns*) combinados a conceitos de arquitetura de software para representar as informações semânticas mais importantes contidas **na estrutura** dos workflows científicos permite auxiliar no estabelecimento de processos **semi-automáticos** de transformação de especificações desses workflows entre diferentes SGWfCs, o que possibilita reduzir, em comparação com um intercâmbio manual, o esforço de um cientista para reutilizar workflows científicos **complexos** (no sentido de terem um grande número de tarefas e relacionamentos entre elas) desenvolvidos por outros grupos de pesquisa em SGWfCs que não fazem parte de seu ambiente de trabalho.

- H_{E0} : O uso de padrões (*patterns*) combinados a conceitos de arquitetura de software para representar as informações semânticas mais importantes contidas **na estrutura** dos workflows científicos **NÃO** permite auxiliar no estabelecimento de processos **semi-automáticos** de transformação de especificações desses workflows entre diferentes SGWfCs.

#tarefas	#workflows	#porcentagem
0	8	0,55%
1	154	10,60%
2	191	13,15%
3	155	10,67%
4	115	7,92%
5	100	6,88%
6	83	5,71%
7	49	3,37%
8	60	4,13%
9	39	2,68%
10	37	2,61%
> 10	461	31,68%

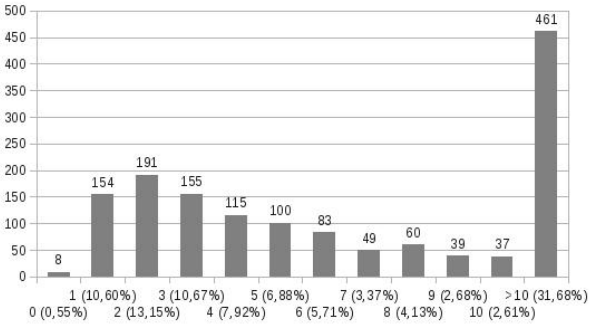


Tabela 4.6: Quantidade de Tarefas do Total de Workflows em Taverna

Assim como na hipótese H_T , para avaliar a hipótese H_E será levado em consideração o percentual de workflows intercambiados, que foi adotado como métrica da redução de esforço obtida ao utilizar a abordagem proposta.

Ao analisar todos os workflows Taverna disponíveis no repositório myExperiment, observa-se que existe uma quantidade considerável de workflows (31%) com mais de 10 tarefas. Os percentuais de distribuição de workflows pelo seu número de tarefas (na especificação original em Taverna) pode ser visto na Tabela 4.6.

Ao transformar *todos* os workflows Taverna para WISP, podemos identificar que o número de workflows que são compostos de uma única tarefa não é tão grande (quando comparado aos workflows 100% intercambiáveis apresentados na Tabela 4.5), sendo apenas 23% dos workflows (vide Tabela 4.7). Essas métricas são importantes para validar a hipótese H_E , uma vez que quanto mais tarefas um workflow possuir, mais complexa será sua reprodução manual em outro SGWfC, e o uso de uma ferramenta auxiliar para a conversão desses workflows automaticamente, mesmo que essa conversão seja parcial[2], é útil.

A Tabela 4.7 ilustra também que foram identificadas falhas no momento da transformação. Essas falhas podem ocorrer no momento da validação das regras na linguagem WISP devido a uma falha na identificação de algum padrão, ou então devido a dependências necessárias para a execução do workflow, que pode estar usando tarefas que acessam recursos que não estão mais disponíveis. Um exemplo de recursos indisponíveis pode ser visto no workflow `http://www.myexperiment.org/workflows/1201.html` que pos-

[2]Em uma transformação parcial, tarefas que não sejam nativamente intercambiáveis em WISP serão representadas como tarefas vazias, mantendo a estrutura do workflow.

#tarefas	#workflows	#porcentagem
0	15	1,03%
1	340	23,43%
2	251	17,29%
3	115	7,92%
4	121	8,33%
5	106	7,30%
6	75	5,16%
7	61	4,20%
8	41	2,82%
9	47	3,23%
10	41	2,82%
> 10	214	14,67%
falhas	25	1,72%

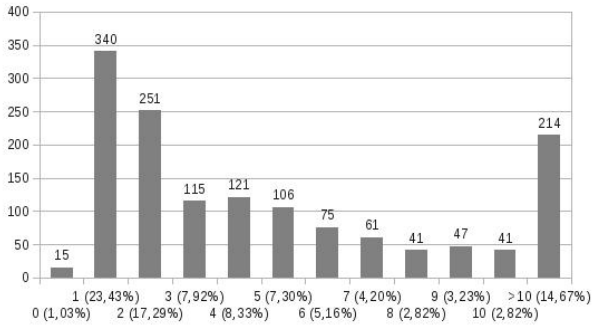


Tabela 4.7: Quantidade de Tarefas do Total de Workflows em WISP

sui um script *BeanShell* que tenta acessar uma base de dados SQLite[3]. Uma vez que a dependência do SQLite não é disponibilizada junto com a especificação desse workflow no repositório myExperiment, e a estratégia para identificação de padrões comportamentais Taverna executa os scripts *BeanShell* do workflow, a transformação semi-automática dessa especificação resulta em uma falha. Ao avaliar a abordagem proposta considerando a hipótese H_E é possível identificar que 98,28% dos workflows analisados podem ser parcialmente ou totalmente intercambiados (de acordo com a quantidade de elementos intercambiáveis contidos nesses workflows), possibilitando uma melhor redução de esforço se comparado com os resultados obtidos para a hipótese H_T .

4.4 ANÁLISE QUALITATIVA DOS RESULTADOS

Após os primeiros resultados do mecanismo de transformações, uma análise manual foi realizada nos 179 workflows compostos exclusivamente de tarefas intercambiáveis. Nesta análise foi possível identificar diversos problemas em relação ao intercâmbio destes workflows. O primeiro problema identificado foi que nos 179 workflows existiam muitos serviços Web inacessíveis. Em alguns casos workflows eram carregados para o repositório myExperiment com o endereço do serviço apontando para *localhost*, o que impossibilita a reprodução desse workflow. Esse problema poderia ser resolvido com uma validação nesses serviços no momento do upload da especificação no próprio repositório, porém, como não há nenhuma validação destas informações e muitos desses workflows não podem ser reproduzidos nem mesmo no SGWfC em que foram desenvolvidos.

[3]https://www.sqlite.org

Além disso, ao analisar o nome das tarefas que os workflows executavam, foi possível identificar que muitos deles se tratavam de testes e, além disso, não possuíam um detalhamento de quais entradas seriam necessárias para a execução do workflow. A grande maioria dos workflows que deveriam ser 100% intercambiáveis e disponíveis para a execução no SGWfC de origem após o download de sua especificação do repositório myExperiment eram compostos de uma única tarefa, o que não traz grandes ganhos ao realizar o intercâmbio através da linguagem WISP. Para workflows compostos de um único serviço Web, cientistas podem facilmente criar tarefas do tipo serviços Web no SGWfC que seja de seu conhecimento e copiar manualmente o endereço e operação que deve ser realizada.

Dentre os 179 workflows, apenas 4 workflows possuíam mais do que uma única tarefa e tinham seus serviços Web acessíveis, porém, todos apresentaram algum problema ao tentar reproduzi-los. Os problemas encontrados foram:

- O workflow era meramente para testes quando possuía todas as entradas, ou;
- O workflow requisitava credenciais de usuário (usuário e senha ou um certificado) que não eram informadas em sua descrição, ou;
- O workflow não possuía descrição das entradas necessárias para a execução do serviço.

Esse cenário impossibilitou a reprodução direta dos workflows disponibilizados no myExperiment em outros SGWfCs. No entanto, ainda é possível analisar se a estrutura do workflow foi mantida e para isso foram utilizados como alvos os workflows com o maior número de tarefas. Esses workflows são analisados a seguir.

4.4.1 ANÁLISE DOS PADRÕES IDENTIFICADOS

Para a análise dos padrões encontrados foram selecionados os 461 workflows com mais de 10 tarefas apresentados na Tabela 4.6 para uma análise manual. Esses workflows possuem um grande número de tarefas e padrões comportamentais e estruturais em sua composição. Podemos utilizar como exemplo um workflow obtido através do repositório myExperiment que está disponibilizado no endereço `http://www.myexperiment.org/workflows/16.html`. Esse workflow possui 61 tarefas. Algumas dessas tarefas são serviços Web. Porém, estes serviços não estão acessíveis e serão tratados como tarefas comuns no momento

da transformação *fromWISP*, uma vez que não é possível obter informações da tarefa ao acessar o endereço do serviço, que são fundamentais para a reprodução do workflow em outros SGWfCs.

O workflow original foi desenvolvido no SGWfC Taverna e é ilustrado na Figura 4.1, onde é possível identificar a existência de diversos tipos de tarefas – por exemplo, os serviços Web são representados pela cor verde.

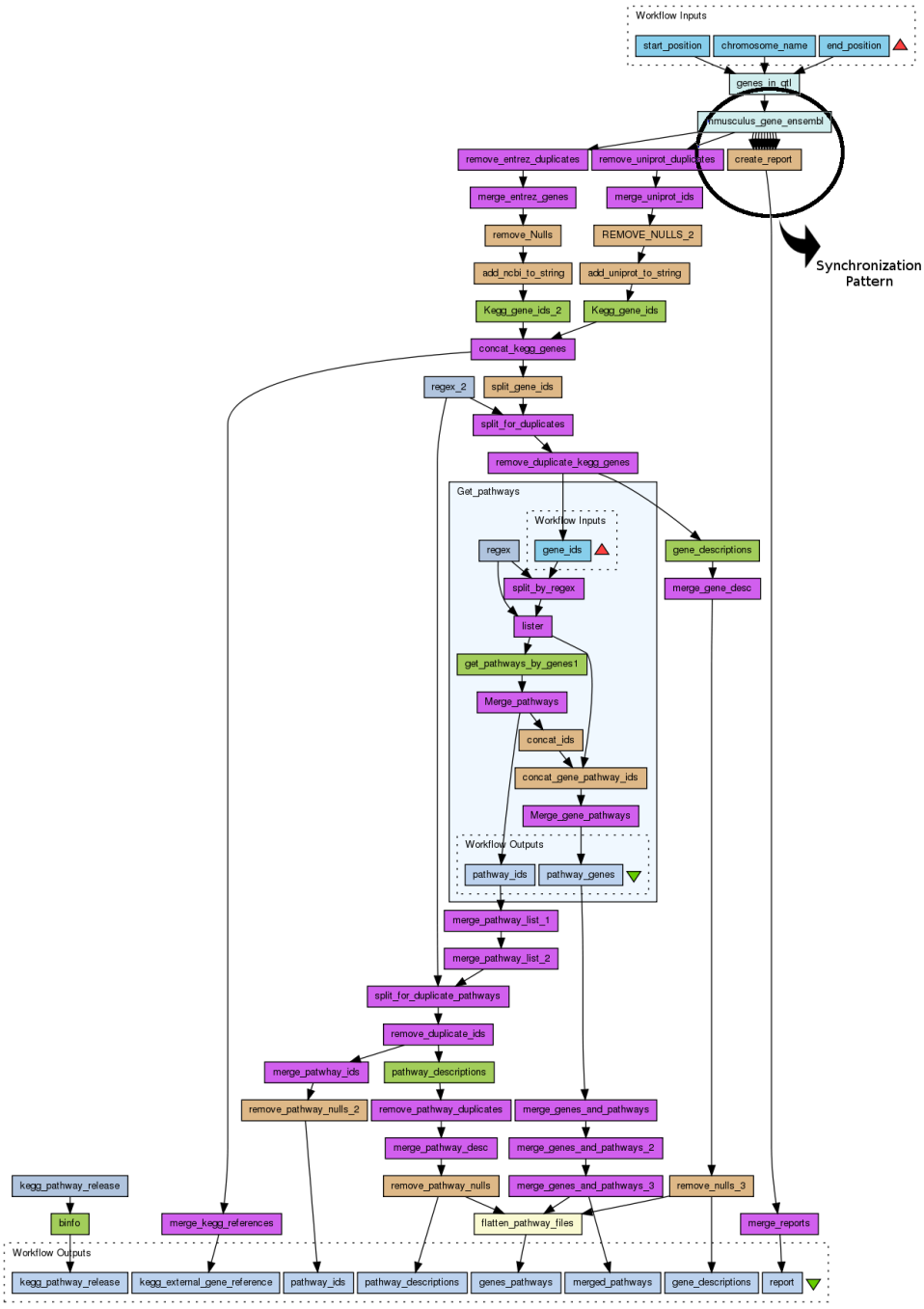


Figura 4.1: Workflow Original Taverna

O mesmo workflow após passar pela linguagem WISP e voltar para o SGWfC Taverna é ilustrado na Figura 4.2, onde é possível identificar que embora o "desenho" do workflow tenha mudado em relação ao workflow original suas conexões permanecem as mesmas. A mudança nesse desenho deve-se ao fato da transformação da linguagem WISP para o SGWfC Taverna ser um processo automático, que por não ter a intervenção do usuário, não reutiliza valores constantes para portas diferentes, ou oculta portas de entrada e saída de um workflow. Em alguns casos, portas de entrada e saída de tarefas que não são obrigatórias e não estão ligadas a outras tarefas podem ser ocultadas em uma especificação, porém, em um processo automático não é possível identificar quais portas podem ser ocultadas.

Na Figura 4.2 é possível também identificar que no momento da conversão Taverna x WISP, foi identificada a existência de um padrão *Synchronization*. Ao transformar a especificação WISP novamente para uma especificação Taverna, esse padrão é representado como uma tarefa *BeanShell*, assim como na especificação de origem, e possui o nome *Synchronization_X*, onde X é uma identificação numérica e sequencial do padrão para evitar nomes duplicados. Cabe ao desenvolvedor do workflow implementar a lógica de sincronização desse padrão, dizendo como os dados de entrada dessa tarefa serão sincronizados.

Essa responsabilidade é do desenvolvedor do workflow pois, a linguagem WISP não possui a informação de como os dados serão sincronizados. Por exemplo, ao sincronizar dados do tipo "string", é possível que seja incluído um caractere de separação (seja um ';' em arquivos CSV ou tabulações em arquivos texto, por exemplo), que são de fundamental importância para a próxima tarefa, que deve saber exatamente o formato do dado que será recebido.

A estrutura desse mesmo workflow também pode ser convertida para o SGWfC Kepler, ilustrada na Figura 4.3, ou para o SGWfC VisTrails, ilustrada na Figura 4.4. No Apêndice B.4 é apresentada uma transformação similar em um workflow que possui o padrão *Exclusive Choice*.

4.5 LIMITAÇÕES ENCONTRADAS

Ao executar os experimentos, algumas limitações foram identificadas, desde limitações na solução proposta, até limitações nos SGWfCs utilizados no experimento. Grande parte dessas limitações se dão devido à inexistência de um arcabouço semântico comum entre

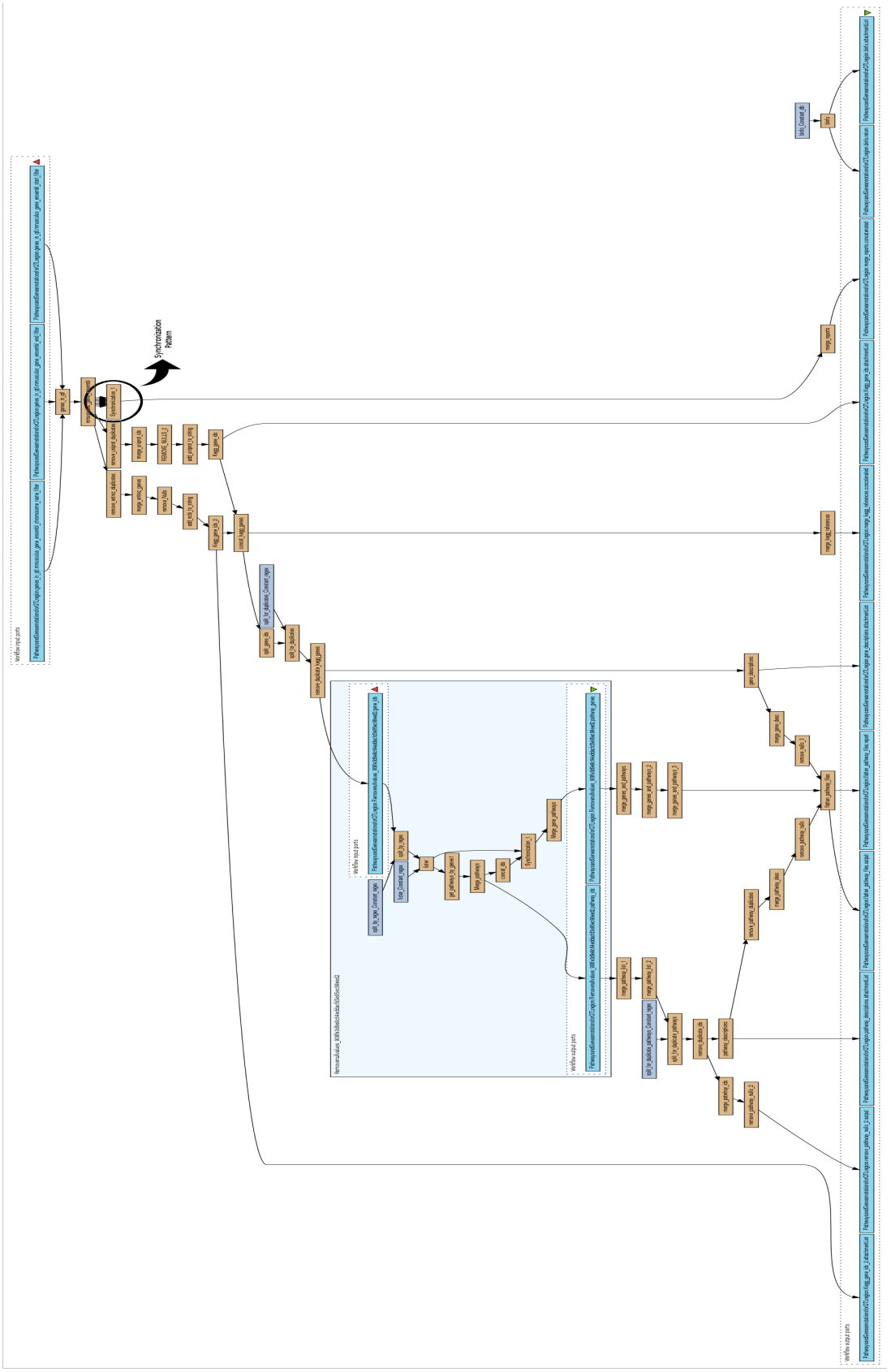


Figura 4.2: Workflow Taverna após ser Convertido para WISP e Voltar para o Taverna

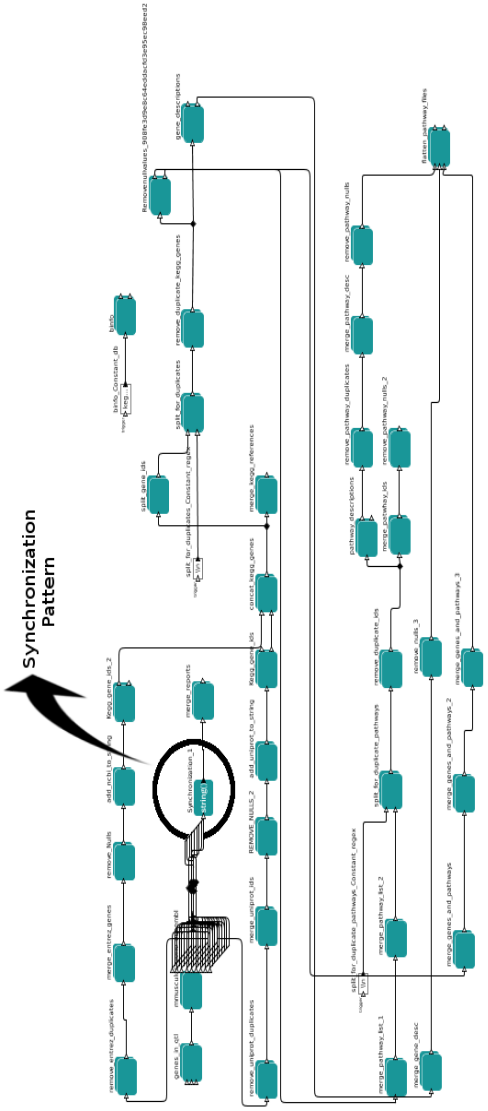


Figura 4.3: Workflow Taverna após ser Convertido para WISP e em Seguida para o Kepler os SGWfCs.

4.5.1 LIMITAÇÕES DA PROPOSTA COM O SGWFC TAVERNA

No SGWfC Taverna, constantes ligadas a uma determinada porta de entrada são representadas como tarefas, assim como os padrões *Exclusive Choice* e *Synchronization*. Já em WISP, esses padrões são representados como conexões e constantes são representadas por valores atribuídos às portas das tarefas. Em alguns casos, workflows Taverna executam um comportamento de um padrão *Exclusive Choice* ou *Synchronization* no início do workflow, com os valores recebidos de constantes. Como não há uma ligação entre essas tarefas em WISP, já que constantes não são representadas como tarefas e padrões são

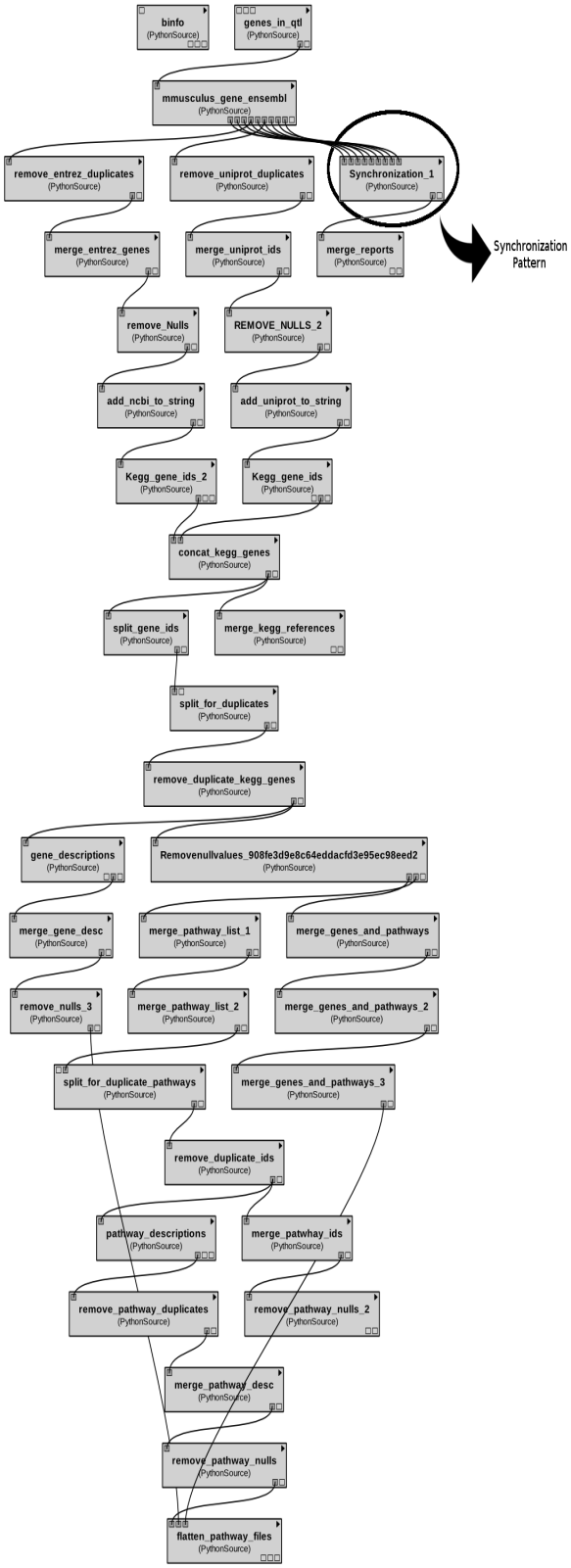


Figura 4.4: Workflow Taverna após ser Convertido para WISP e em Seguida para o VisTrails

representados como conexões, regras associadas aos padrões não podem ser validadas. Ao tentar validar as regras associadas às estruturas desses padrões, não é possível encontrar a tarefa de origem da(s) conexão(ões) do padrão, uma vez que era uma tarefa constante que não foi representada em WISP.

Além disso, a identificação dos padrões comportamentais no Taverna é feita através da execução de seus scripts *BeanShell* e identificação de suas saídas. Um intervalo de valores é utilizado como entrada para a tarefa e é identificado se sua saída realizou o comportamento de um dos padrões comportamentais em **todas** as execuções. Caso isso aconteça, essa tarefa é removida do workflow e tratada como uma conexão de um padrão comportamental. Porém, ainda é possível que uma tarefa seja erroneamente identificada como um padrão *Exclusive Choice*, pois não há garantias que o intervalo de valores utilizados para a identificação desse padrão seja aplicável para todos os casos. Esse comportamento foi implementado apenas para o SGWfC Taverna pois os workflows utilizados como entrada para a avaliação foram desenvolvidos neste SGWfC.

Como não é possível saber o comportamento interno desse padrão, ao transformar esses workflows a partir da linguagem WISP, é necessário que o usuário faça uma intervenção manual dizendo o que deve acontecer quando um determinado dado for recebido nas portas de entrada do padrão (tarefa no Taverna). Isso ocorre porque nos casos de um padrão *Exclusive Choice*, a condição exclusiva do padrão pode ser diferente para cada uma das instâncias do padrão, assim como um padrão *Synchronization* pode sincronizar os valores recebidos das portas de saída de diferentes maneiras (concatenação de strings, CSV, arrays, etc).

4.5.2 LIMITAÇÕES DA PROPOSTA COM O SGWFC KEPLER

O SGWfC Kepler possui diferentes maneiras de orquestrar a execução de um workflow. Um workflow desenvolvido em Kepler pode conter informações semânticas relacionadas ao seu comportamento embutidas em um diretor (*Director*). Como o conceito de diretor não é empregado em WISP ou nos outros SGWfCs estudados, essa informação pode ser perdida.

Além disso, a informação semântica relacionada a troca de dados das tarefas, pode estar contida na própria conexão. Embora a estrutura do workflow seja a mesma, algumas dessas informações podem ser fundamentais para a semântica do workflow e também pode

ser perdida em uma transformação. Um exemplo é o atributo *width* que pode limitar o que será passado para a próxima tarefa de acordo com o tipo de dado que está trafegando na conexão.

4.5.3 LIMITAÇÕES DA PROPOSTA COM O SGWFC VISTRAILS

No SGWfC VisTrails foi identificado um comportamento diferente para o tratamento de workflows internos. Quando um usuário VisTrails diz que um conjunto de tarefas é na verdade um workflow interno, o VisTrails automaticamente transforma esse workflow interno em um módulo na ferramenta. Esse módulo é tratado como uma tarefa e pode ser utilizado em diferentes workflows.

Entretanto, ao transformar um conjunto de tarefas em um workflow interno, esse módulo passa a fazer parte do VisTrails e não mais do workflow. Ao exportar esse workflow, é necessário que esse módulo esteja previamente instalado na máquina onde esse workflow será importado, ou então a informação de suas tarefas não será encontrada. Esse comportamento impossibilita o reuso de um workflow que possua workflows internos, até mesmo entre diferentes instâncias do SGWfC VisTrails.

4.6 DISCUSSÕES E ANÁLISE DAS HIPÓTESES E OBJETIVOS DA WISP

Analisando os resultados obtidos e as hipóteses apresentadas anteriormente é possível identificar que embora a afirmação da hipótese H_T implicasse em ganhos significativos a cientistas e laboratórios colaborativos, o intercâmbio de workflows científicos através da identificação de padrões em um processo automático geraria demasiado esforço devido à inexistência de uma norma (ou padronização) comum a todos os SGWfCs e a maneira que determinadas funcionalidades são implementadas nestes SGWfCs. Além disso, muitas das informações contidas em um SGWfC estão presentes em um comportamento interno das tarefas o que nem sempre pode ser extraído em um processo de transformação, mesmo que este seja manual.

A hipótese H_T foi então reformulada gerando a hipótese H_E , onde o foco é manter a semântica da estrutura de workflows científicos, observando o momento em que os dados são trocados entre as tarefas, por meio de um processo semi-automático de transforma-

ção. Embora a execução interna das tarefas de um workflow possa ser perdida em uma transformação, uma intervenção manual pode ser realizada para trazer esse comportamento para o workflow no novo SGWfC. Essa reformulação foi motivada considerando a hipótese de que cientistas iriam se beneficiar em reutilizar especificações de workflows que foram modelados em outros SGWfCs em um ambiente que estejam familiarizados.

Os resultados apresentados anteriormente viabilizam a afirmação da hipótese H_{E1} , pois é possível observar que o número de tarefas em um workflow científico pode dificultar o intercâmbio manual do mesmo. Além de reproduzir a estrutura de um workflow, o mecanismo se propõe a identificar padrões comportamentais que podem ser de fundamental importância para o workflow e, devido a maneira pela qual esses padrões são implementados nos SGWfCs, podem ser dificilmente identificados em um intercâmbio manual por um cientista que não conheça o SGWfC de origem. Ao identificar esses padrões e armazenar a semântica de um workflow em WISP, é possível obter informações cuja identificação manual pode não ser trivial para um cientista, uma vez que o workflow pode ter sido desenvolvido em um SGWfC que não seja de seu conhecimento.

Considerando o espaço amostral disponível no repositório myExperiment para os experimentos realizados nesta dissertação, podemos obter indícios que a hipótese H_E pode ser validada. Contudo, algumas limitações ainda podem ser identificadas na abordagem. Essas limitações podem ser encontradas em funcionalidades que existem em um SGWfC e que não possam ser representadas em outros SGWfCs, como por exemplo conceito de *diretores*, apresentado pelo Kepler. A ausência de informações auxiliares à execução dos workflows obtidos para os experimentos realizados também impossibilitaram a validação completa de um processo semi-automático de transformação, uma vez que muitos serviços estavam indisponíveis e parâmetros necessários para a execução do workflows muitas vezes não eram informados, impossibilitando assim a execução dos workflows no SGWfC que foi desenvolvido ou em outro SGWfC após uma transformação seguida de uma intervenção manual.

4.6.1 AMEAÇAS À VALIDADE

As ameaças à validade desse estudo serão tratadas em relação à: (1) validade do constructo, que é confiança que se tem de que as medidas que estão sendo estudadas realmente representam aquilo que o pesquisador tem em mente e o que é investigado de acordo com

as questões de pesquisa; (2) validade interna, que é a confiança que se tem de que o efeito observado é realmente devido à manipulação feita, e não a outros fatores; (3) à validade externa que é a confiança que se tem que o efeito observável é generalizável (WOHLIN et al., 2012).

Em relação ao constructo essa proposta se baseia em facilitar o intercâmbio de especificações de workflows científicos entre diferentes cientistas ou laboratórios colaborativos. Embora a reprodução de um workflows científicos desenvolvido em um SGWfC diferente do qual um cientista está familiarizado seja possível em alguns cenários, experimentos não foram realizados com cientistas para analisar qual a complexidade de adaptar o workflow após uma transformação, mesmo que sua estrutura tenha sido mantida.

Em relação à validade interna, o protótipo implementado nessa dissertação não foi especificado pelos desenvolvedores do SGWfC, o que pode levar essa implementação a conter características dos workflows utilizados para os testes no momento do desenvolvimento e não o comportamento interno característico do SGWfC. Como em alguns SGWfCs é possível atingir um mesmo comportamento de maneiras diferentes, seria necessário que o comportamento esperado pelos desenvolvedores do SGWfCs fosse utilizado internamente no protótipo ao realizar uma transformação.

Em relação à validade externa, o protótipo apresentado e disponibilizado na Web pode ser acessado por diferentes cientistas. Porém, a semântica de workflow pode não ser totalmente preservada o que pode levar um cientista a um erro no momento de execução e validação dos resultados do workflow. Como a semântica pode estar contida internamente em algumas tarefas e essas tarefas podem não ser intercambiadas, o processo de intercâmbio do workflow científico pode não trazer ganhos a um cientista.

No entanto, mesmo considerando essas ameças a validade, consideramos que o trabalho traz ganhos para um cientista que necessitam trabalhar colaborativamente com outros grupos de pesquisa, uma vez que o uso da WISP facilita o intercâmbio e o entedimento de diferentes workflows, desenvolvidos em diferentes SGWfCs. Nesse contexto, este trabalho contribui para a ideia de laboratórios colaborativos (OLSON, 2009), onde o intercâmbio de informações científicas é essencial e o uso da WISP pode contribuir para este propósito.

4.7 CONCLUSÕES DO CAPÍTULO

Nesse capítulo foram apresentados os resultados dos experimentos realizados com a linguagem WISP. Esses resultados judam a um maior entendimento a respeito das especificações de workflows científicos, não só em relação ao que o SGWfC oferece, mas também em relação a maneira como os workflows são desenvolvidos.

Em muitos casos, é possível notar que a falta de conhecimento de uma determinada funcionalidade de um SGWfC faz com que o usuário prefira implementar essa funcionalidade como um comportamento interno de uma tarefa. Esse comportamento acaba prejudicando a identificação da semântica do workflow, uma vez que é necessário executar a tarefa e ler sua implementação interna para saber como ela se comporta. Além disso, a falta de conhecimento dos tipos possíveis de entrada, pode levar a uma identificação errada da tarefa.

Também foram apresentadas algumas limitações da proposta, tanto em relação aos SGWfCS, quanto na implementação da própria linguagem. Essas limitações se devem desde a inexistência de uma padronização entre SGWfCs, o que faz com que cada SGWfC específico possua funcionalidades que não podem ser mapeadas em outros SGWfCS, até a maneira pela qual os workflows são desenvolvidos, o que pode impossibilitar a identificação de funcionalidades importantes do workflow.

No capítulo seguinte serão apresentadas as considerações finais do trabalho, incluindo contribuições, limitações e trabalhos futuros.

5 CONSIDERAÇÕES FINAIS

O reuso de workflows científicos é um tema complexo e recebe atenção crescente em *e-science*. Esse reuso pode ser atingido de diferentes maneiras, que foram pontuadas nesta dissertação e que podem trazer um grande ganho para a comunidade científica. Neste contexto, o reuso através do intercâmbio de especificações de workflows científicos tem como objetivo principal permitir que o cientista utilize seu ambiente de desenvolvimento usual, ao invés de ter que utilizar um ambiente com o qual não esteja familiarizado.

Para atingir esse reuso, a hipótese inicial (H_T) deste trabalho se baseou em um processo automático de transformações, com a utilização de padrões (*patterns*) encontrados nas especificações de workflows científicos para intercambiar diferentes especificações. Ao utilizar esses padrões, este trabalho se propôs a manter o máximo das informações semânticas contidas em um workflow – e desta forma poder reproduzir o mais próximo o comportamento original desse workflow – em um SGWfC de destino.

Técnicas de transformações podem ser utilizadas em especificações de workflows científicos para reescrever os padrões identificados em uma especificação descrita segundo uma linguagem em uma outra linguagem. Foi proposta então uma linguagem intermediária denominada *Workflow Interchange in Science with Patterns* - WISP. Essa linguagem foi desenvolvida sobre o arcabouço sintático e semântico da linguagem Acme. WISP define um conjunto de tipos de tarefas e de padrões, e composições de instâncias desses tipos são restringidas através de regras Armani. A validação dessas regras garante que algum desenvolvedor que esteja empregando o framework apresentado na Seção 3.3 para implementar um processo automatizado de transformação (por exemplo, para dar suporte a um novo SGWfC) não gere especificações WISP que sejam sintaticamente válidas em Acme mas que estruturalmente não façam sentido do ponto de vista dos *patterns* de interesse.

Para validar a hipótese (H_T), de que especificações de workflows científicos podem ter sua especificação intercambiável através de um processo automático, este trabalho realizou uma pesquisa exploratória, utilizando workflows reais, disponíveis no repositório myExperiment. Porém, ao analisar esses workflows diversas limitações para a validação da hipótese H_T foram encontradas. Desta forma, foi proposta uma hipótese alternativa (H_E), mais fraca, porém, de resultado prático potencialmente útil, que propôs a utilização

de um processo semi-automático para o intercâmbio da estrutura de workflows científicos. Através dessa estrutura, é possível reproduzir um workflow em outro SGWfC, mesmo que uma intervenção manual seja necessária, trazendo ganhos aos cientistas que estejam familiarizados com um ambiente de trabalho.

A reprodução de workflows que possuem uma grande quantidade de tarefas é um processo complexo, ainda mais quando o usuário não conhece o SGWfC de origem. Dito isso, acredita-se que a utilidade da linguagem intermediária proposta, em conjunto com um processo semi-automático de transformação, aumente de forma proporcional com o tamanho das especificações de workflow reutilizadas, em termos do número de suas tarefas. Embora uma intervenção manual possa ser necessária para que o workflow funcione no SGWfC de destino, parte considerável do esforço de intercâmbio pode ser evitado com a abordagem apresentada. Para validar a hipótese H_E , os mesmos workflows usados na tentativa de validação da hipótese H_T foram empregados. Constatou-se (conforme apresentado na Subseção 4.3.2) que em mais de 98% dos casos a estrutura do workflow foi mantida. Além disso, também foi possível identificar padrões que não são exclusivamente estruturais (mais especificamente, *Exclusive Choice* e *Synchronization*) o que pode não ser trivial em um processo manual de intercâmbio.

5.1 CONTRIBUIÇÕES

O aprendizado de um novo SGWfC pode não ser trivial para um cientista que já está familiarizado com seu ambiente de trabalho. Porém, ao analisar workflows científicos desenvolvidos em outros SGWfC, o cientista pode identificar ganhos no reuso de workflows desenvolvidos por colegas e que não estejam especificados no SGWfC com o qual o cientista esteja familiarizado. A possibilidade de intercambiar a estrutura de um workflow científico desenvolvida em um SGWfC qualquer para um SGWfC conhecido, com o qual o cientista esteja familiarizado, pode trazer ganhos funcionais e não-funcionais para a pesquisa, embora talvez ainda seja necessário fazer ajustes manuais relacionados ao comportamento interno das tarefas.

A utilidade do intercâmbio da estrutura de workflows científicos desenvolvidos em diferentes SGWfCs aumenta de acordo com o número de tarefas encontradas em um workflow. Ao analisar os workflows do repositório myExperiment, foi possível identificar que uma quantidade considerável de workflows possui mais de 10 tarefas, e o intercâmbio

manual desses workflows pode ser complexo, visto que o cientista pode ter pouco, ou nenhum, conhecimento do SGWfC onde o workflow foi originalmente desenvolvido.

O intercâmbio de especificações de workflows científicos não é trivial devido às diferentes funcionalidades encontradas em SGWfCs. O uso de uma linguagem intermediária pode limitar a expressividade contida em uma especificação no momento de uma transformação. A linguagem WISP foca na representação da estrutura de um workflow, de forma independente da implementação interna das tarefas. Desta forma, como a linguagem WISP se propõe a manter a semântica do workflow através de padrões, a identificação de padrões comportamentais pode ajudar ainda mais um cientista que não esteja familiarizado com o SGWfC onde o workflow foi desenvolvido, pois a implementação de padrões comportamentais pode variar de acordo com as funcionalidades do SGWfC.

Para isso, foi necessário que a linguagem WISP mantivesse seu foco na estrutura de um workflow e não no comportamento interno das tarefas, inicialmente. Porém, para tentar realizar transformações completas em cenários específicos, o suporte ao intercâmbio de tarefas do tipo serviços Web, que compõem uma parte não negligenciável das tarefas encontradas em workflows científicos, WISP pode trazer ainda mais ganhos a cientistas que não quiserem sair de seu ambiente de trabalho conhecido.

5.2 LIMITAÇÕES

A heterogeneidade dos SGWfCs foi uma característica que dificultou a implementação da proposta deste trabalho desde o início. Muitas funcionalidades existentes em SGWfCs não são diretamente mapeáveis para outros, uma vez que esses SGWfCs não seguem uma norma (ou padronização) comum para suas especificações. Devido a isso, a linguagem proposta nesse trabalho sofreu algumas limitações, visando atender o maior número de SGWfCs, porém, ciente de que alguns comportamentos não poderiam ser representados.

A primeira limitação é em relação ao uso de padrões. Embora existam diferentes padrões estruturais e comportamentais que podem ser encontrados nos SGWfCs, foi necessário escolher um conjunto inicial de padrões para uma primeira versão da linguagem. Esse conjunto utilizou padrões estruturais e comportamentais. O uso do padrão *Sequence* torna possível a representação mínima de um workflow, onde seriam representadas apenas suas conexões entre portas de entrada e saída, sem nenhuma semântica associada ao comportamento dos dados gerados e consumidos pelas portas das tarefas.

A segunda limitação é em relação a representação das tarefas de um SGWfC. Devido à grande quantidade de tarefas encontradas em um SGWfC, foi necessário limitar as tarefas que seriam representadas na linguagem WISP. Para isso, a linguagem se propõe a representar apenas a estrutura dos workflows, salvo em casos onde o tipo das tarefas seja um serviço Web, que trabalha sob um protocolo comum e que é possível ser utilizado em qualquer linguagem de programação ou sistema operacional.

A terceira e última limitação é em relação aos SGWfCs e aos seus usuários, que ao não seguir padrões arquiteturais para a criação de seus workflows dificultam a identificação da semântica do workflow de maneira automática através de sua especificação. Como muitos SGWfCS oferecem scripts nativos como uma de suas tarefas, é possível implementar a lógica do workflow utilizando esses scripts, mesmo em casos que existam tarefas pré-definidas no SGWfC para realizar o mesmo comportamento. Ao ler uma especificação de workflow científico que utiliza muitos desses scripts pode não ser possível identificar a semântica do workflow e representá-la em WISP, o que torna o intercâmbio inviável e pode, até mesmo, em alguns casos dificultar a leitura do workflow intercambiado por um cientista.

5.3 TRABALHOS FUTUROS

Um ponto que deve ser abordado é a abrangência dos padrões oferecidos na linguagem. Para uma primeira versão, o alvo dos estudos foram os SGWfCs Taverna, Kepler e VisTrails. Porém, existe uma grande quantidade de SGWfCs disponíveis, que podem, por sua vez, conter padrões estruturais diferentes que sejam de fundamental importância para seus workflows e possam ser representados em outros SGWfCs. Os padrões utilizados neste trabalho formularam uma primeira versão da proposta, porém, padrões como os propostos por Garijo et al. (2014) podem trazer mais semântica a linguagem WISP, proporcionando uma menor intervenção manual após a realização de um intercâmbio.

Esta dissertação também identificou vários problemas relacionados ao repositório myExperiment. Muitos desses problemas se devem ao fato desse repositório ter um grande número de workflows de testes e workflows com pouca documentação. A validação dos workflows no momento em que são enviados ao myExperiment resolveria grande parte dos problemas encontrados ao longo desta dissertação. Essa validação poderia ser feita desde a necessidade de uma descrição para o workflow e seus parâmetros de entrada e saída, até

validações internas da disponibilidade das tarefas de um workflow, que pode ser obtida através da leitura de suas especificações. Considerações semelhantes às levantadas nesta dissertação são listadas pelo projeto Wf4Ever[1], que apresenta como um de seus objetivos a criação de uma nova versão do repositório myExperiment destinada a resolver grande parte dos problemas mencionados anteriormente.

Finalmente, um estudo de casos deve ser conduzido com cientistas, o que possibilitará a identificação de diversos problemas ilustrados nas ameaças à validade dessa proposta.

[1] `http://www.wf4ever-project.org/`

REFERÊNCIAS

ABOUELHODA, M.; ISSA, S.; GHANEM, M. Tavaxy: Integrating taverna and galaxy workflows with cloud computing support. **BMC Bioinformatics**, v. 13, p. 77, 2012.

ALTINTAS, I.; BARNEY, O.; CHENG, Z.; CRITCHLOW, T.; LUDAESCHER, B.; PARKER, S.; SHOSHANI, A.; VOUK, M. Accelerating the scientific exploration process with scientific workflows. **J. Phys. : Conf. Ser.**, v. 46, p. 468–478, 2006.

ALTINTAS, I.; BERKLEY, C.; JAEGER, E.; JONES, M.; LUDASCHER, B.; MOCK, S. Kepler: an extensible system for design and execution of scientific workflows. In: **Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on**, 2004. p. 423–424.

BELHAJJAME, K.; DEUS, H.; GARIJO, D.; KLYNE, G.; MISSIER, P.; SOILAND-REYES, S.; ZEDNIK, S. **PROV Model Primer**, 2012. Disponível em: <http://www.w3.org/TR/prov-primer/>.

BELLOUM, A.; INDA, M. A.; VASUNIN, D.; KORKHOV, V.; ZHAO, Z.; RAUWERDA, H.; BREIT, T. M.; BUBAK, M.; HERTZBERGER, L. O. Collaborative e-Science Experiments and Scientific Workflows. **IEEE Internet Computing**, IEEE Computer Society, Los Alamitos, CA, USA, v. 15, p. 39–47, 2011. ISSN 1089-7801. Disponível em: <http://dx.doi.org/10.1109/mic.2011.87>.

CALLAHAN, S. P.; FREIRE, J.; SANTOS, E.; SCHEIDEGGER, C. E.; SILVA, C. T.; VO, H. T. Vistrails: Visualization meets data management. In: **Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data**, 2006. (SIGMOD '06), p. 745–747. ISBN 1-59593-434-0. Disponível em: <http://doi.acm.org/10.1145/1142473.1142574>.

CEREZO, N.; MONTAGNAT, J.; BLAY-FORNARINO, M. Computer-assisted scientific workflow design. **Journal of Grid Computing**, Springer Netherlands, v. 11, n. 3, p. 585–612, 2013.

DAVIDSON, S. B.; FREIRE, J. Provenance and scientific workflows: Challenges and opportunities. In: **Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data**, 2008. (SIGMOD '08), p. 1345–1350. ISBN 978-1-60558-102-6. Disponível em: <http://doi.acm.org/10.1145/1376616.1376772>.

Eclipse Foundation. **EMF: Eclipse Modeling Framework**. 2012. Http://www.eclipse.org/modeling/emf/.

EMEAKAROHA, V. C.; LABAJ, P. P.; MAURER, M.; BRANDIC, I.; KREIL, D. P. Optimizing bioinformatics workflows for data analysis using cloud management techniques. In: **Proceedings of the 6th Workshop on Workflows in Support of Large-scale Science**, 2011. (WORKS '11), p. 37–46. ISBN 978-1-4503-1100-7. Disponível em: <http://doi.acm.org/10.1145/2110497.2110503>.

FAHRINGER, T.; PRODAN, R.; DUAN, R.; HOFER, J.; NADEEM, F.; NERIERI, F.; PODLIPNIG, S.; QIN, J.; SIDDIQUI, M.; TRUONG, H.-L.; VILLAZON, A.; WIECZOREK, M. Askalon: A development and grid computing environment for scientific workflows. In: TAYLOR, I.; DEELMAN, E.; GANNON, D.; SHIELDS, M. (Ed.). **Workflows for e-Science**, 2007. p. 450–471. ISBN 978-1-84628-519-6.

FREUDLING, W.; ROMANIELLO, M.; BRAMICH, D. M.; BALLESTER, P.; FORCHI, V.; GARCIA-DABLO, C. E.; MOEHLER, S.; NEESER, M. J. Automated data reduction workflows for astronomy. **Astronomy & Astrophysics**, v. 559, p. A96+, nov. 2013. ISSN 0004-6361. Disponível em: <http://dx.doi.org/10.1051/0004-6361/201322494>.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design Patterns: Elements of Reusable Object-oriented Software**, 1995. ISBN 0-201-63361-2.

GARIJO, D.; ALPER, P.; BELHAJJAME, K.; CORCHO, O.; GIL, Y.; GOBLE, C. Common motifs in scientific workflows: An empirical analysis. **Future Generation Computer Systems**, v. 36, n. 0, p. 338 – 351, 2014.

GARLAN, D.; MONROE, R.; WILE, D. Acme: An architecture description interchange language. In: **Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research**, 1997. (CASCON '97), p. 7–. Disponível em: <http://dl.acm.org/citation.cfm?id=782010.782017>.

GASPAR, W.; BRAGA, R.; CAMPOS, F.; ORNELAS, T. Scientific provenance metadata capture and management using semantic web. In: **International Journal of Metadata, Semantics and Ontologies (Print)**, 2015.

GESING, S.; ATKINSON, M.; KLAMPANOS, I.; GALEA, M.; BERTHOLD, M. R.; BARBERA, R.; SCARDACI, D.; TERSTYANSZKY, G.; KISS, T.; KACSUK, P. The demand for consistent web-based workflow editors. In: **Proceedings of the 8th Workshop on Workflows in Support of Large-Scale Science**, 2013. (WORKS '13), p. 112–123.

GIARDINE, B.; RIEMER, C.; HARDISON, R. C.; BURHANS, R.; ELNITSKI, L.; SHAH, P.; ZHANG, Y.; BLANKENBERG, D.; ALBERT, I.; TAYLOR, J.; MILLER, W.; KENT, W. J.; NEKRUTENKO, A. Galaxy: a platform for interactive large-scale genome analysis. **Genome Res**, v. 15, n. 10, p. 1451–5, 2005.

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**, v. 40, n. 12, p. 24–32, 2007.

GLATARD, T.; MONTAGNAT, J.; LINGRAND, D.; PENNEC, X. Flexible and efficient workflow deployment of data-intensive applications on grids with moteur. **Int. J. High Perform. Comput. Appl.**, Sage Publications, Inc., Thousand Oaks, CA, USA, v. 22, n. 3, p. 347–360, ago. 2008. ISSN 1094-3420.

GUAN, Z.; HERNáNDEZ, F.; BANGALORE, P.; GRAY, J. G.; SKJELLUM, A.; VELUSAMY, V.; LIU, Y. Grid-flow: a grid-enabled scientific workflow system with a petri-net-based interface. **Concurrency and Computation: Practice and Experience**, v. 18, n. 10, p. 1115–1140, 2006.

JOUAULT, F.; ALLILAIRE, F.; BÉZIVIN, J.; KURTEV, I. Atl: A model transformation tool. **Sci. Comput. Program.**, Elsevier North-Holland, Inc., Amsterdam, The Netherlands, The Netherlands, v. 72, n. 1-2, p. 31–39, jun. 2008. ISSN 0167-6423. Disponível em: <http://dx.doi.org/10.1016/j.scico.2007.08.002>.

KACSUK, P. P-grade portal family for grid infrastructures. **Concurr. Comput. : Pract. Exper.**, John Wiley and Sons Ltd., Chichester, UK, v. 23, n. 3, p. 235–245, mar. 2011. ISSN 1532-0626.

KARASAVVAS, K.; WOLSTENCROFT, K.; MINA, E.; CRUICKSHANK, D.; WILLIAMS, A.; ROURE, D. D.; GOBLE, C.; ROOS, M. Opening new gateways to workflows for life scientists. **Studies in health technology and informatics**, v. 175, p. 131–141, 2012.

MARINHO, A.; MURTA, L.; WERNER, C.; BRAGANHOLO, V.; CRUZ, S. M. S. d.; OGASAWARA, E.; MATTOSO, M. Provmanager: a provenance management system for scientific workflows. **Concurr. Comput. : Pract. Exper.**, John Wiley and Sons Ltd., Chichester, UK, v. 24, n. 13, p. 1513–1530, set. 2012. ISSN 1532-0626. Disponível em: <http://dx.doi.org/10.1002/cpe.1870>.

MEDEIROS, V.; GOMES, A. T. A. Expressando atributos não-funcionais em workflows científicos. In: **Anais do VII Brazillian e-Science Workshop (BreSci)**, 2013.

MIGLIORINI, S.; GAMBINI, M.; ROSA, M. L.; HOFSTEDE, A. H. ter. Pattern-based evaluation of scientific workflow management systems. February 2011. Disponível em: <http://eprints.qut.edu.au/39935/>.

MONROE, R. T. **Rapid development of custom software architecture design environments**. Tese (Doutorado) — Carnegie Mellon University, Pittsburgh, PA, USA, 1999.

MOREAU, L.; CLIFFORD, B.; FREIRE, J.; FUTRELLE, J.; GIL, Y.; GROTH, P.; KWASNIKOWSKA, N.; MILES, S.; MISSIER, P.; MYERS, J.; PLALE, B.; SIMMHAN, Y.; STEPHAN, E.; BUSSCHE, J. V. den. The open provenance model core specification (v1.1). **Future Gener. Comput. Syst.**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 27, n. 6, p. 743–756, jun. 2011. ISSN 0167-739X. Disponível em: <http://dx.doi.org/10.1016/j.future.2010.07.005>.

MUEHLEN, M. Z.; KLEIN, F. AFRICA: Workflow interoperability based on xml-messages. In: **Proceedings of CAiSE'00 workshop on Infrastructures for Dynamic Businessto-Business Service Outsourcing (IDSO'00)**, 2000. p. 5–6.

OGASAWARA, E.; PAULINO, C.; MURTA, L.; WERNER, C.; MATTOSO, M. Experiment Line: Software Reuse in Scientific Workflows. In: , 2009. p. 264–272.

OGASAWARA, E. S.; OLIVEIRA, D. de; VALDURIEZ, P.; DIAS, J.; PORTO, F.; MATTOSO, M. An algebraic approach for data-centric scientific workflows. **PVLDB**, v. 4, n. 12, p. 1328–1339, 2011.

OINN, T.; ADDIS, M.; FERRIS, J.; MARVIN, D.; CARVER, T.; POCOCK, M. R.; WIPAT, A. Taverna: A tool for the composition and enactment of bioinformatics workflows. **Bioinformatics**, v. 20, p. 2004, 2004.

OLSON, G. M. The next generation of science collaboratories. In: **Proceedings of the 2009 International Symposium on Collaborative Technologies and Systems**, 2009. (CTS '09), p. xv–xvi. ISBN 978-1-4244-4584-4. Disponível em: <http://dx.doi.org/10.1109/CTS.2009.5067429>.

PEREIRA, A. F. **Collaborative PL-Science: Utilizando Elementos de Colaboração em uma Linha de Produtos de Software Científico**. Dissertação (Mestrado) — Universidade Federal de Juiz de Fora - UFJF, 2014.

PLANKENSTEINER, K.; MONTAGNAT, J.; PRODAN, R. Iwir: A language enabling portability across grid workflow systems. In: **Proceedings of the 6th Workshop on Workflows in Support of Large-scale Science**, 2011. (WORKS '11), p. 97–106. ISBN 978-1-4503-1100-7. Disponível em: <http://doi.acm.org/10.1145/2110497.2110509>.

PLANKENSTEINER, K.; PRODAN, R.; JANETSCHEK, M.; FAHRINGER, T.; MONTAGNAT, J.; ROGERS, D.; HARVEY, I.; TAYLOR, I.; BALASKÓ, Á.; KACSUK, P. Fine-grain interoperability of scientific workflows in distributed computing infrastructures. **J. Grid Comput.**, v. 11, n. 3, p. 429–455, 2013.

SHAW, M. Toward higher-level abstractions for software systems. **Data Knowl. Eng.**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 5, n. 2, p. 119–128, jul. 1990. ISSN 0169-023X. Disponível em: <http://dx.doi.org/10.1016/0169-023X(90)90008-2>.

SILVA, F. N. da. **In Services: Um sistema para gerenciamento de dados intermediários em workflows científicos na bioinformática**. Dissertação (Mestrado) — Instituto Militar de Engenharia - IME, 2006.

STAHL, T.; VOELTER, M.; CZARNECKI, K. **Model-Driven Software Development: Technology, Engineering, Management**, 2006. ISBN 0470025700.

SULEIMAN, B.; TOSIC, V.; ALIEV, E. Non-functional property specifications for wright adl. In: WU, Q.; HE, X.; NGUYEN, Q. V.; JIA, W.; HUANG, M. L. (Ed.). **International Conference on Computer and Information Technology**, 2008. p. 766–771. ISBN 978-1-4244-2357-6.

SWENSON, K. D.; PRADHAN, S.; GILGER, M. D. Wf-XML 2.0 - xml based protocol for run-time integration of process engines. 2004.

TAYLOR, I.; SHIELDS, M.; WANG, I.; RANA, O. Triana applications within grid computing and peer to peer environments. **Journal of Grid Computing**, Kluwer Academic Publishers, v. 1, n. 2, p. 199–217, 2003. ISSN 1570-7873.

TERSTYANSZKY, G.; KUKLA, T.; KISS, T.; KACSUK, P.; BALASKO, A.; FARKAS, Z. Enabling scientific workflow sharing through coarse-grained interoperability. **Future Generation Computer Systems**, mar. 2014. ISSN 0167739X. Disponível em: <http://dx.doi.org/10.1016/j.future.2014.02.016>.

WASSINK, I.; VANDERVET, D. P.; WOLSTENCROFT, K.; NEERINCX, P.; ROOS, M.; RAUWERDA, H.; BREIT, T. Analysing scientific workflows: why workflows not only connect web services. In: Zhang, L. (Ed.). **IEEE Congress on Services 2009**, 2009. p. 314–321. Disponível em: <http://doc.utwente.nl/67519/>.

WEIDT, F.; DAVID, J. M.; BRAGA, R.; CAMPOS, F. PRIME: Pragmatic Interoperability Architecture to Support Collaborative Development of Scientific Workflows. In: Simpósio Brasileiro de Componentes, Arquiteturas e Reuso de software. **In: CBSOFT: IX Braziliam Symposium on Components, Architectures and Reuse Software (SBCARS 2015)**, p. 50 – 60, 2015.

WFMC, W. M. C. **Workflow Management Coalition Terminology Glossary**. 1999. Disponível em: <http://www.wfmc.org/Download-document/WFMC-TC-1011-Ver-3-Terminology-and-Glossary-English.html>.

WOHLIN, C.; RUNESON, P.; HöST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in Software Engineering**, 2012.

YILDIZ, U.; GUABTNI, A.; NGU, A. H. H. Towards scientific workflow patterns. In: **Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science**, 2009. (WORKS '09), p. 13:1–13:10. ISBN 978-1-60558-717-2. Disponível em: <http://doi.acm.org/10.1145/1645164.1645177>.

ZHANG, J.; TAN, W.; ALEXANDER, J.; FOSTER, I. T.; MADDURI, R. K. Recommend-as-you-go: A novel approach supporting services-oriented scientific workflow reuse. In: JACOBSEN, H.-A.; WANG, Y.; HUNG, P. (Ed.). **IEEE SCC**, 2011. p. 48–55. ISBN 978-1-4577-0863-3.

ZHAO, Y.; HATEGAN, M.; CLIFFORD, B.; FOSTER, I.; LASZEWSKI, G. von; NEFEDOVA, V.; RAICU, I.; STEF-PRAUN, T.; WILDE, M. Swift: Fast, Reliable, Loosely Coupled Parallel Computation. In: **Services, 2007 IEEE Congress on**, 2007. p. 199–206.

ZHAO, Z.; BOOMS, S.; BELLOUM, A.; LAAT, C. de; HERTZBERGER, B. VLE-WFBus: a scientific workflow bus for multi e-Science domains. In: **Proceedings of the 2nd IEEE International conference on e-Science and Grid computing**, 2006. v. 0, p. 11–19. Disponível em: <http://dx.doi.org/10.1109/E-SCIENCE.2006.154>.

Apêndice A - LINGUAGEM WISP

A.1 FAMÍLIA ACME DA LINGUAGEM WISP

```
Family WISP = {
  Port Type InputPort = {
    Property value : string;
    Property ptype : string <<  default : string = "any"; >> ;
    rule onlyFrom = invariant forall p in self.ATTACHEDROLES |
      declaresType(p, Destination);
  }
  Port Type OutputPort = {
    Property ptype : string <<  default : string = "any"; >> ;
    rule onlyTo = invariant forall p in self.ATTACHEDROLES |
      declaresType(p, Origin);
  }
  Component Type Task = {
  }
  Component Type WebServiceTask extends Task with {
    Property wsdl : string;
    Property operation : string;
    rule hasWsdl = invariant wsdl != "";
    rule hasOperation = invariant operation != "";
  }
  Role Type Origin = {
  }
  Role Type Destination = {
  }
  Connector Type ParallelSplitPattern = {
    rule exactly1roleOfTypeInput = invariant size(/self/ROLES:!Origin)
       == 1;
    rule atLeast2rolesOfTypeOutput = invariant size(/self/ROLES:!
       Destination) >= 2;
  }
  Connector Type SequencePattern = {
    rule exactly1roleOfTypeInput = invariant size(/self/ROLES:!Origin)
       == 1;
```

```
    rule exactly1roleOfTypeOutput = invariant size(/self/ROLES:!
        Destination) == 1;
  }
  Connector Type SynchronizationPattern = {
    rule atLeast2rolesOfTypeInput = invariant size(/self/ROLES:!Origin)
        >= 2;
    rule exactly1roleOfTypeOutput = invariant size(/self/ROLES:!
        Destination) == 1;
  }
  Connector Type SimpleMergePattern = {
    rule atLeast2rolesOfTypeInput = invariant size(/self/ROLES:!Origin)
        >= 2;
    rule exactly1roleOfTypeOutput = invariant size(/self/ROLES:!
        Destination) == 1;
  }
  Connector Type ExclusiveChoicePattern = {
        rule exactly1roleOfTypeInput = invariant size(/self/ROLES:!
            Origin) == 1;
    rule atLeast2rolesOfTypeOutput = invariant size(/self/ROLES:!
        Destination) >= 2;
  }
}
```

Listagem A.1: Código Acme que Representa a Família da Linguagem WISP

Apêndice B - INFORMAÇÕES COMPLEMENTARES

B.1 TABELA COM WEBSITES DOS SGWFCS

SGWfC	Website
Anduril	http://www.anduril.org/anduril/site/
ASKALON	http://www.askalon.org/
Apache Airavata	https://airavata.apache.org/community/projects-using.html
Bioclipse	http://www.bioclipse.net/
Egatis	http://ergatis.sourceforge.net/
Galaxy	http://galaxyproject.org/
Kepler	https://kepler-project.org/
KNIME	https://www.knime.org/
Mobyle	http://mobyle.pasteur.fr/workflow
OnlineHPC	http://onlinehpc.com/
OpenMOLE	http://www.openmole.org/
Orange	http://orange.biolab.si/
Pegasus	http://pegasus.isi.edu/
P-GRADE	http://portal.p-grade.hu/
Ptolemy II	http://ptolemy.eecs.berkeley.edu/ptolemyII/
Swift	http://swift-lang.org/main/
Taverna	http://www.taverna.org.uk/
Triana	http://www.trianacode.org/
VisTrails	http://www.vistrails.org/index.php/Main_Page
Yabi	https://code.google.com/p/yabi/
YAWL	http://www.yawlfoundation.org/

Tabela B.1: Páginas dos Sistemas Gerenciadores de Workflow Científicos

B.2 TABELA COMPLETA DE TAREFAS DO SGWFC TAVERNA NO REPOSITÓRIO MYEXPERIMENT

Tarefa Taverna	Quantidade
net.sf.taverna.t2.activities.stringconstant.StringConstantActivity	3902
net.sf.taverna.t2.activities.localworker.LocalworkerActivity	2836
net.sf.taverna.t2.activities.beanshell.BeanshellActivity	2363
net.sf.taverna.t2.activities.wsdl.WSDLActivity	1466

net.sf.taverna.t2.activities.dataflow.DataflowActivity	1285
net.sf.taverna.t2.activities.wsdl.xmlsplitter.XMLInputSplitterActivity	1212
net.sf.taverna.t2.activities.wsdl.xmlsplitter.XMLOutputSplitterActivity	896
net.sf.taverna.t2.activities.externaltool.ExternalToolActivity	509
net.sf.taverna.t2.activities.rest.RESTActivity	322
net.sf.taverna.t2.activities.xpath.XPathActivity	292
net.sf.taverna.t2.activities.rshell.RshellActivity	256
net.sf.taverna.t2.activities.soaplab.SoaplabActivity	139
net.sf.taverna.t2.component.ComponentActivity	131
net.sf.taverna.t2.activities.biomart.BiomartActivity	120
net.sf.taverna.t2.activities.apiconsumer.ApiConsumerActivity	74
net.sf.taverna.t2.activities.interaction.InteractionActivity	66
org.purl.wf4ever.astrotaverna.voutils.GetListFromColumnActivity	55
org.purl.wf4ever.astrotaverna.tpipe.AddColumnByExpressionActivity	55
org.purl.wf4ever.astrotaverna.tjoin.TjoinActivity	47
net.sf.taverna.t2.activities.spreadsheet.SpreadsheetImportActivity	44
org.purl.wf4ever.astrotaverna.tcat.TcatListActivity	37
net.sf.taverna.t2.activities.biomoby.BiomobyActivity	33
org.purl.wf4ever.astrotaverna.tpipe.FormatConversionActivity	28
net.sf.taverna.t2.activities.biomoby.MobyParseDatatypeActivity	24
org.purl.wf4ever.astrotaverna.tpipe.SelectColumnsActivity	23
org.purl.wf4ever.astrotaverna.tpipe.SelectRowsActivity	20
net.sf.taverna.t2.activities.biomoby.BiomobyObjectActivity	14
...taverna.vamdc_taverna_suite.TapXSamsQueryHelperActivity	12
...cs.elico.rmservicetype.taverna.RapidMinerExampleActivity	11
net.sf.taverna.t2.activities.sadi.SADIActivity	10
de.fzj.unicore.taverna.gridservice.activity.UnicoreAppActivity	9
...cdk.applications.taverna.qsar.CSVToQSARVectorActivity	8
net.sf.taverna.cagrid.activity.xmlsplitter.XMLInputSplitterActivity	8
org.openscience.cdk.applications.taverna.io.MDLSDFileWriterActivity	8
org.purl.wf4ever.astrotaverna.voutils.TemplateFillerActivity	7
org.openscience.cdk.applications.taverna.io.CSVFileWriterActivity	7

org.purl.wf4ever.astrotaverna.pdl.PDLServiceActivity	7
org.vamdc.taverna.vamdc_taverna_suite.TapXSamsActivity	7
org.openscience.cdk.applications.taverna.io.XRFFFileWriterActivity	6
net.sf.taverna.cagrid.activity.CaGridActivity	6
org.purl.wf4ever.astrotaverna.tjoin.CrossMatch2Activity	6
org.purl.wf4ever.astrotaverna.voutils.CheckTemplateFillerActivity	5
org.purl.wf4ever.astrotaverna.tpipe.ResolveCoordsActivity	5
net.sf.taverna.t2.activities.xmpp.XMPPActivity	5
...wf4ever.astrotaverna.voutils.AddCommonRowToVOTableActivity	4
at.ac.tuwien.RAWverna.JavaImageEvaluatorActivity	4
org.openscience.cdk.applications.taverna.io.MDLSDFileReaderActivity	4
org.vamdc.taverna.vamdc_taverna_suite.ConsumerServiceActivity	4
org.vamdc.taverna.vamdc_taverna_suite.CEAActivity	4
org.openscience.cdk.applications.taverna.io.CSVFileReaderActivity	3
net.sf.taverna.cagrid.activity.xmlsplitter.XMLOutputSplitterActivity	3
...cdk.applications.taverna.iterativeio.IterativeSDFileReaderActivity	3
org.cagrid.transfer.CaGridTransferActivity	3
...cdk.applications.taverna.qsar.QSARVectorToCSVActivity	3
...taverna.weka.regression.SplitRegressionTrainTestsetActivity	2
...taverna.classification.art2a.ART2aClassificationActivity	2
...taverna.qsar.CreateFingerprintItemListFromQSARVectorActivity	2
...cdk.applications.taverna.io.MDLRXNFileReaderActivity	2
org.cagrid.cql.CQLActivity	2
...taverna.weka.regression.WekaRegressionActivity	2
org.openscience.cdk.applications.taverna.io.TextFileWriterActivity	2
org.vamdc.taverna.vamdc_taverna_suite.SpectColActivity	2
...reactionenumerator.ReactionEnumeratorSubgraphFilterActivity	2
...cdk.applications.taverna.qsar.MergeQSARVectorsActivity	2
org.openscience.cdk.applications.taverna.io.XRFFFileReaderActivity	2
...applications.taverna.curation.MoleculeConnectivityCheckerActivity	2
...taverna.weka.CreateWekaDatasetFromQSARVectorActivity	2
uk.ac.soton.mib104.t2.activities.oauth.OAuth10aRESTActivity	2

...applications.taverna.miscellaneous.TagMoleculesWithUUIDActivity	2
...taverna.weka.regression.EvaluateRegressionResultsAsPDFActivity	2
org.openscience.cdk.applications.taverna.weka.WekaClusteringActivity	2
org.purl.wf4ever.astrotaverna.tpipe.CoordTransformationActivity	2
..cdk.applications.taverna.renderer.WriteMoleculeAsPDFActivity	2
org.helio.taverna.helio_taverna_suite.TapHelioActivity	1
...taverna.jchempaint.JChemPaintActivity	1
net.sf.taverna.t2.activities.usecase.UseCaseActivity	1
...taverna.signaturescoring.PlotDistributionAsPDFActivity	1
...taverna.isomorphism.SubgraphIsomorphismFilterActivity	1
org.helio.taverna.helio_taverna_suite.TapHelioQueryHelperActivity	1
org.purl.wf4ever.astrotaverna.tcat.TcatActivity	1
...weka.GenerateSilhouettePlotFromClusteringResultAsPDFActivity	1
...taverna.miscellaneous.AddExplicitHydrogensActivity	1
at.ac.tuwien.photohawk.taverna.SimpleSSIMActivity	1
...taverna.weka.SplitMoleculesIntoClustersActivity	1
at.ac.tuwien.photohawk.taverna.MSEActivity	1
at.ac.tuwien.photohawk.taverna.AEActivity	1
...taverna.miscellaneous.AromaticityDetectorActivity	1
...cdk.applications.taverna.io.MDLRXNFileWriterActivity	1
...qsar.GetMolecularWeightDistributionFromQSARVectorActivity	1
...art2a.tools.ART2aResultConsideringDifferentOriginsAsPDF	1
...taverna.weka.regression.CreateWekaRegressionSetActivity	1
org.purl.wf4ever.astrotaverna.tpipe.AddSkyCoordsActivity	1
at.ac.tuwien.photohawk.taverna.MAEActivity	1
...taverna.qsar.MergeCSVsToQSARVectorActivity	1
...taverna.weka.ClusteringResultConsideringDifferentOriginsAsPDF	1
...taverna.curation.SugarGroupRemoverActivity	1
...taverna.signaturescoring.AtomSignatureActivity	1
...taverna.qsar.QSARDescriptorThreadedActivity	1
...taverna.signaturescoring.NaturalProductLikenessCalculatorActivity	1
org.openscience.cdk.applications.taverna.io.SMILESFileWriterActivity	1

org.openscience.cdk.applications.taverna.filter.AtomTypeFilterActivity	1
...taverna.curation.CurateStrangeElementsActivity	1
at.ac.tuwien.photohawk.taverna.PAEActivity	1
org.vamdc.taverna.vamdc_taverna_suite.TapXSamsOperatorActivity	1
...taverna.weka.CreateWekaDatasetFromCSVActivity	1
...taverna.reactionenumerator.ReactionEnumeratorActivity	1
...taverna.qsar.QSARVectorGeneratorActivity	1
icar.cnr.it.FLSOM.FLSOMActivity	1
...taverna.qsar.CurateQSARVectorActivity	1
...taverna.weka.regression.GARAttributeSelectionActivity	1
org.purl.wf4ever.astrotaverna.aladin.AladinMacroActivity	1
org.cagrid.cds.CDSActivity	1
uk.ac.manchester.cs.img.esc.EscActivity	1
...taverna.miscellaneous.ReactionReactantSplitterActivity	1
uk.ac.soton.mib104.t2.activities.oauth.OAuthActivity	1
...taverna.classification.art2a.tools.ART2aResultAsPDF	1
Total	**16.492**

Tabela B.2: Tarefas do SGWfC Taverna no Repositório myExperiment

B.3 CÓDIGO DO PADRÃO *EXCLUSIVE CHOICE* NOS SGWFCS TAVERNA E KEPLER

```
<processors>
<processor>
  <name>Exclusive_Choice</name>
  <inputPorts>
    <port>
      <name>condition</name>
      <depth>0</depth>
    </port>
  </inputPorts>
  <outputPorts>
```

```
    <port>
      <name>failed</name>
      <depth>1</depth>
      <granularDepth>1</granularDepth>
    </port>
    <port>
      <name>passed</name>
      <depth>1</depth>
      <granularDepth>1</granularDepth>
    </port>
  </outputPorts>
  <annotations />
  <activities>
    <activity>
      <raven>
        <group>net.sf.taverna.t2.activities</group>
        <artifact>beanshell-activity</artifact>
        <version>1.4</version>
      </raven>
      <class>net.sf.taverna.t2.activities.beanshell.BeanshellActivity
      </class>
      <inputMap>
        <map from="condition" to="condition" />
      </inputMap>
      <outputMap>
        <map from="failed" to="failed" />
        <map from="passed" to="passed" />
      </outputMap>
      <configBean encoding="xstream">
        <net.sf.taverna.t2.activities.beanshell.
           BeanshellActivityConfigurationBean
          xmlns="">
          <inputs>
            <net.sf.taverna.t2.workflowmodel.processor.activity.config.
               ActivityInputPortDefinitionBean>
              <name>condition</name>
              <depth>0</depth>
              <mimeTypes>
                <string>text/plain</string>
```

```
      </mimeTypes>
      <handledReferenceSchemes />
      <translatedElementType>java.lang.String
      </translatedElementType>
      <allowsLiteralValues>true</allowsLiteralValues>
    </net.sf.taverna.t2.workflowmodel.processor.activity.config.
       ActivityInputPortDefinitionBean>
  </inputs>
  <outputs>
    <net.sf.taverna.t2.workflowmodel.processor.activity.config.
       ActivityOutputPortDefinitionBean>
      <name>passed</name>
      <depth>1</depth>
      <mimeTypes />
      <granularDepth>1</granularDepth>
    </net.sf.taverna.t2.workflowmodel.processor.activity.config.
       ActivityOutputPortDefinitionBean>
    <net.sf.taverna.t2.workflowmodel.processor.activity.config.
       ActivityOutputPortDefinitionBean>
      <name>failed</name>
      <depth>1</depth>
      <mimeTypes />
      <granularDepth>1</granularDepth>
    </net.sf.taverna.t2.workflowmodel.processor.activity.config.
       ActivityOutputPortDefinitionBean>
  </outputs>
  <classLoaderSharing>workflow</classLoaderSharing>
  <localDependencies />
  <artifactDependencies />
  <script>
    passed = new java.util.ArrayList();
    failed = new java.util.ArrayList();

    if(condition.equals("true")) {
    passed.add("Passed");
    } else {
    failed.add("Failed");
    }
  </script>
```

```
        <dependencies />
      </net.sf.taverna.t2.activities.beanshell.
         BeanshellActivityConfigurationBean>
    </configBean>
    <annotations />
  </activity>
</activities>
<dispatchStack>
  <dispatchLayer>
    <raven>
      <group>net.sf.taverna.t2.core</group>
      <artifact>workflowmodel-impl</artifact>
      <version>1.4</version>
    </raven>
    <class>net.sf.taverna.t2.workflowmodel.processor.dispatch.layers.
       Parallelize
    </class>
    <configBean encoding="xstream">
      <net.sf.taverna.t2.workflowmodel.processor.dispatch.layers.
         ParallelizeConfig
        xmlns="">
        <maxJobs>1</maxJobs>
      </net.sf.taverna.t2.workflowmodel.processor.dispatch.layers.
         ParallelizeConfig>
    </configBean>
  </dispatchLayer>
  <dispatchLayer>
    <raven>
      <group>net.sf.taverna.t2.core</group>
      <artifact>workflowmodel-impl</artifact>
      <version>1.4</version>
    </raven>
    <class>net.sf.taverna.t2.workflowmodel.processor.dispatch.layers.
       ErrorBounce
    </class>
    <configBean encoding="xstream">
      <null xmlns="" />
    </configBean>
  </dispatchLayer>
```

```
<dispatchLayer>
  <raven>
    <group>net.sf.taverna.t2.core</group>
    <artifact>workflowmodel-impl</artifact>
    <version>1.4</version>
  </raven>
  <class>net.sf.taverna.t2.workflowmodel.processor.dispatch.layers.
     Failover
  </class>
  <configBean encoding="xstream">
    <null xmlns="" />
  </configBean>
</dispatchLayer>
<dispatchLayer>
  <raven>
    <group>net.sf.taverna.t2.core</group>
    <artifact>workflowmodel-impl</artifact>
    <version>1.4</version>
  </raven>
  <class>net.sf.taverna.t2.workflowmodel.processor.dispatch.layers.
     Retry
  </class>
  <configBean encoding="xstream">
    <net.sf.taverna.t2.workflowmodel.processor.dispatch.layers.
       RetryConfig
      xmlns="">
      <backoffFactor>1.0</backoffFactor>
      <initialDelay>1000</initialDelay>
      <maxDelay>5000</maxDelay>
      <maxRetries>0</maxRetries>
    </net.sf.taverna.t2.workflowmodel.processor.dispatch.layers.
       RetryConfig>
  </configBean>
</dispatchLayer>
<dispatchLayer>
  <raven>
    <group>net.sf.taverna.t2.core</group>
    <artifact>workflowmodel-impl</artifact>
    <version>1.4</version>
```

```
      </raven>
      <class>net.sf.taverna.t2.workflowmodel.processor.dispatch.layers.
         Invoke
      </class>
      <configBean encoding="xstream">
        <null xmlns="" />
      </configBean>
    </dispatchLayer>
  </dispatchStack>
  <iterationStrategyStack>
    <iteration>
      <strategy>
        <cross>
          <port name="condition" depth="0" />
        </cross>
      </strategy>
    </iteration>
  </iterationStrategyStack>
</processor>
```

Listagem B.1: Código do Padrão *Exclusive Choice* na Especificação do SGWfC Taverna

```
<entity name="Boolean␣Switch" class="ptolemy.actor.lib.BooleanSwitch">
  <property name="KeplerDocumentation"
    class="ptolemy.vergil.basic.KeplerDocumentationAttribute">
    <property name="description" class="ptolemy.kernel.util.
       ConfigurableAttribute">
      <configure>null</configure>
    </property>
    <property name="author" class="ptolemy.kernel.util.
       ConfigurableAttribute">
      <configure>Steve Neuendorffer</configure>
    </property>
    <property name="version" class="ptolemy.kernel.util.
       ConfigurableAttribute">
      <configure>null</configure>
    </property>
    <property name="port:input" class="ptolemy.kernel.util.
       ConfigurableAttribute">
      <configure>An input port that accepts tokens of any type.</
```

```
        configure>
    </property>
    <property name="port:falseOutput" class="ptolemy.kernel.util.
        ConfigurableAttribute">
      <configure>An output port that broadcasts the input token when the
        control is false.</configure>
    </property>
    <property name="port:trueOutput" class="ptolemy.kernel.util.
        ConfigurableAttribute">
      <configure>An output port that broadcasts the input token when the
        control is true.</configure>
    </property>
    <property name="port:control" class="ptolemy.kernel.util.
        ConfigurableAttribute">
      <configure>An input port that accepts a Boolean token used to
          select
        which output port (trueOutput or falseOutput) to broadcast.</
            configure>
    </property>
  </property>
  <property name="entityId" class="org.kepler.moml.NamedObjId"
    value="urn:lsid:kepler-project.org:actor:54:1">
  </property>
  <property name="class" class="ptolemy.kernel.util.StringAttribute"
    value="ptolemy.actor.lib.BooleanSwitch">
    <property name="id" class="ptolemy.kernel.util.StringAttribute"
      value="urn:lsid:kepler-project.org:class:930:1">
    </property>
  </property>
  <property name="semanticType00" class="org.kepler.sms.SemanticType"
    value="urn:lsid:localhost:onto:1:1#BooleanControlActor">
  </property>
  <property name="semanticType11" class="org.kepler.sms.SemanticType"
    value="urn:lsid:localhost:onto:2:1#BooleanControl">
  </property>
  <property name="_location" class="ptolemy.kernel.util.Location"
    value="{160.0, 120.0}">
  </property>
  <port name="control" class="ptolemy.actor.TypedIOPort">
```

```
    <property name="input" />
    <property name="_cardinal" class="ptolemy.kernel.util.
        StringAttribute"
      value="SOUTH">
    </property>
  </port>
</entity>
```

Listagem B.2: Código do Padrão *Exclusive Choice* na Especificação do SGWfC Kepler

B.4 WORKFLOWS CONVERTIDOS UTILIZANDO O MECANISMO DE TRANSFORMAÇÕES

O workflow ilustrado na figura B.1 foi obtido através do repositório myExperiment que está disponibilizado no endereço `http://www.myexperiment.org/workflows/991.html`. Suas representações após sua transformação para os SGWfCs Taverna, Kepler e VisTrails (após passar pela linguagem WISP), são ilustradas nas figuras B.2, B.3 e B.4, respectivamente.

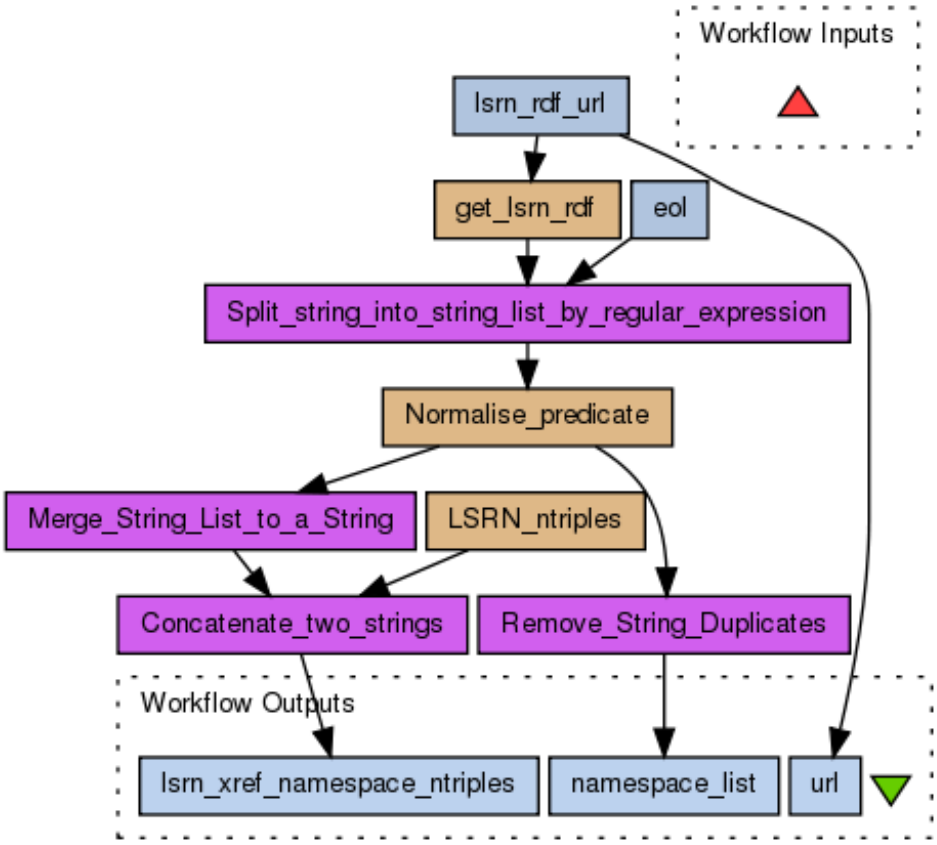


Figura B.1: Workflow Original Taverna

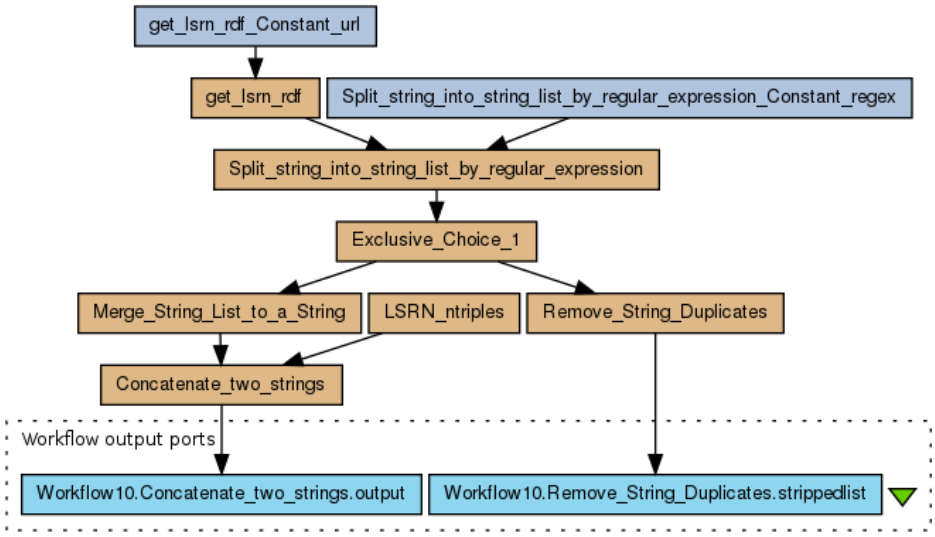


Figura B.2: Workflow Taverna após ser Convertido para WISP e Voltar para o Taverna

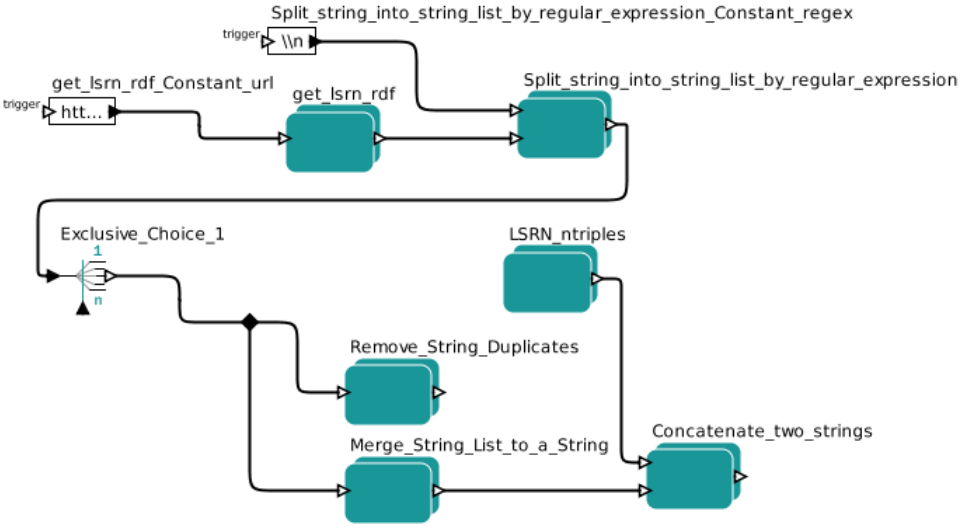


Figura B.3: Workflow Taverna após ser Convertido para WISP e Voltar para o Kepler

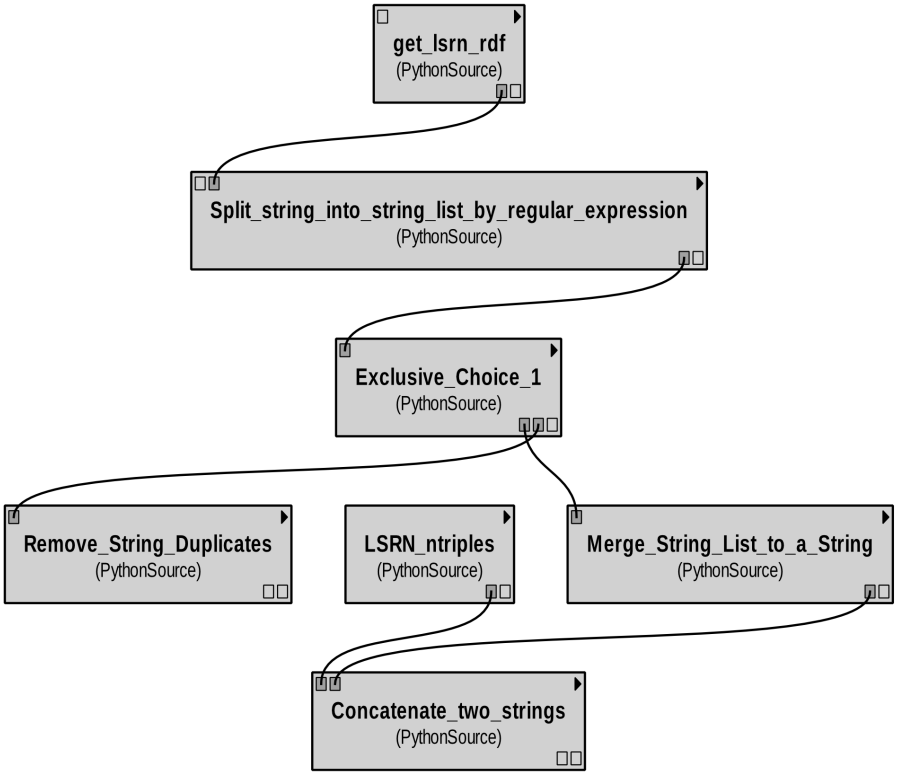


Figura B.4: Workflow Taverna após ser Convertido para WISP e Voltar para o VisTrails