PROVMANAGER: UMA ABORDAGEM PARA GERENCIAMENTO DE PROVENIÊNCIA DE EXPERIMENTOS CIENTÍFICOS

Anderson Souza Marinho

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Ciências em Engenharia de Sistemas e Computação.

Orientadores: Cláudia Maria Lima Werner
Leonardo Gresta Paulino Murta

Rio de Janeiro
Fevereiro de 2011

PROVMANAGER: UMA ABORDAGEM PARA GERENCIAMENTO DE PROVENIÊNCIA DE EXPERIMENTOS CIENTÍFICOS

Anderson Souza Marinho

DISSERTAÇÂO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

_____

Prof.ª Cláudia Maria Lima Werner, D.Sc.

_____

Prof. Leonardo Gresta Paulino Murta, D.Sc.

_____

Prof. Marta Lima de Queirós Mattoso, D.Sc.

_____

Prof. Fábio André Machado Porto, D.Sc.

RIO DE JANEIRO, RJ – BRASIL
FEVEREIRO DE 2011

Marinho, Anderson Souza

ProvManager: Uma Abordagem para Gerenciamento de Proveniência de Experimentos Científicos / Anderson Souza Marinho – Rio de Janeiro: UFRJ/COPPE, 2011.

XII, 133 p.: il.; 29,7 cm.

Orientadores: Cláudia Maria Lima Werner.

Leonardo Gresta Paulino Murta.

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2011.

Referências Bibliográficas: p. 96-100.

1. Gerenciamento de Proveniência de Dados. 2. Experimentação Científica. I. Werner, Cláudia Maria Lima *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

A Deus, por tudo que eu sou e tenho. Nada aconteceu ou vai acontecer na minha vida sem a Sua permissão. Independente da minha situação, na alegria ou na tristeza, eu sempre estarei agradecido, pois, sem merecer, grandes coisas já fizestes por mim. Por isso, sempre Te louvarei.

Aos meus pais Lydia Maria e Luiz Antônio, por estarem sempre presentes na minha formação com a maior atenção do mundo e amor. Cada um teve um papel fundamental para eu estar aqui hoje. Esta dissertação é dedicada a vocês dois.

A toda minha família e entes queridos que me ajudaram direta ou indiretamente para eu estar presente aqui.

Aos velhos amigos de graduação (Eldanae, Rafael, Hua Lin e Cláudio) e aos novos amigos que eu adquiri no período do mestrado (Rodrigo, Wallace, Marcelo, Eduardo, Daniel), fico muito feliz de ter conhecido essas pessoas. Eu aprendi muito, ajudei e fui extremamente ajudado por eles.

Aos meus amigos em geral que tiveram contribuição neste trabalho.

A minha orientadora Claudia Werner, por estar comigo desde a minha graduação dando suporte e contribuindo no meu amadurecimento profissional. Agradeço pela oportunidade, por ter acreditado no meu potencial e pela paciência durante essa jornada.

Ao meu orientador Leonardo Murta, que esteve sempre de perto para me orientar e dar bons conselhos desde a minha graduação. Durante este longo período, eu não posso dizer apenas que tive um grande orientador, mas também que obtive um grande amigo.

Aos integrantes da minha banca, Fábio Porto e Marta Mattoso, por terem gentilmente aceitado o convite e contribuído neste trabalho.

A Vanessa Braganholo e a Marta Mattoso, pelas contribuições que fizeram durante a minha pesquisa de dissertação, podendo ser consideradas em certos momentos como minhas co-orientadoras.

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

PROVMANAGER: UMA ABORDAGEM PARA GERENCIAMENTO DE PROVENIÊNCIA DE EXPERIMENTOS CIENTÍFICOS

Anderson Souza Marinho

Fevereiro/2011

Orientadores: Cláudia Maria Lima Werner
Leonardo Gresta Paulino Murta

Programa: Engenharia de Sistemas e Computação

A execução de *workflows* científicos em ambientes distribuídos e heterogêneos vem motivando a procura por abordagens de gerência de proveniência que são fracamente acopladas a sistemas de *workflow*. Este tipo de abordagem é essencial, pois permite o armazenamento e acesso aos dados de proveniência de uma forma integrada, mesmo em um ambiente onde diferentes sistemas gerenciadores de *workflow* científico trabalham em conjunto. A fim de proporcionar recursos de proveniência nestes cenários, as abordagens existentes sobrecarregam os cientistas com muitas tarefas computacionais manuais como, por exemplo, adaptação de código e implementação de funcionalidades extras. Entretanto, isso não é uma boa estratégia quando se trata de abordagens em larga escala para usuários que possuem outros interesses (*e.g.*, cientistas de áreas não tecnológicas). Portanto, esta dissertação propõe uma abordagem de gerência de dados de proveniência que facilita a captura, o armazenamento e a análise integrada de informações de proveniência em cenários de ambientes heterogêneos e distribuídos.

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

PROVMANAGER: AN APPROACH TO MANAGE PROVENANCE DATA FOR SCIENTIFIC EXPERIMENTS

Anderson Souza Marinho

February/2011

Advisors: Cláudia Maria Lima Werner
Leonardo Gresta Paulino Murta

Department: Computer Science and Systems Engineering

Running scientific workflows in distributed and heterogenous environments is motivating the definition of provenance gathering approaches that are loosely coupled to the workflow execution engine. This kind of approach is essencial because it allows both storage and access to provenance data in an integrated way, even in an environment where different workflow management systems work together. In order to provide provenance functionalities, the existing approaches overload scientists with many manually computing tasks, such as script adaptations and implementations of extra functionalities. However, this is not a good solution when it comes to large scale approaches for users who have other concerns (e.g., scientists from non technology fields). Hence, this paper purposes a provenance management approach that eases the gathering, storing, and analysis of provenance information in a distributed and heterogeneous environment scenario.

ÍNDICE

Índice de Figuras

Índice de Tabelas

Capítulo 1. Introdução

1.1 Experimentação científica

A experimentação científica é uma das formas usadas pelos cientistas para investigar fenômenos, tendo como objetivo adquirir novos conhecimentos ou corrigir e integrar conhecimentos previamente estabelecidos (Wilson 1991). Nas últimas décadas, com o avanço da computação, os experimentos científicos passaram a utilizar ferramentas computacionais para facilitar a sua execução. A utilização dessas ferramentas computacionais foi motivada, também, pelo próprio aumento da complexidade dos experimentos que, em alguns casos, eram inviáveis de serem executados sem algum tipo de apoio computacional. A partir dessa disponibilidade de recursos computacionais, alguns experimentos passaram a simular seus próprios ambientes de execução, deixando de ser executados em ambientes reais ou controlados, ou seja, experimentos conhecidos como *in vivo* e *in vitro*, respectivamente. Essa nova categoria de experimentos passou a ser chamada de experimentos *in virtuo*. Além disso, até mesmo os objetos e os participantes de um experimento passaram a ser simulados, surgindo a categoria de experimentos *in silico* (Travassos e Barros 2003) (Zhao et al. 2004).

A partir dessa transformação de cenário na experimentação, os experimentos passaram a usar uma grande quantidade de programas e a sua manipulação tornou-se mais complexa (Mattoso et al. 2008). O conceito de *workflow* (Hollingsworth 1995) passa a ser utilizado com o objetivo de facilitar a representação de um experimento, permitindo a composição estruturada de programas como uma sequência de atividades que visa um determinado resultado (Hollingsworth 1995). Um *workflow*, segundo (WfMC 2011), é qualificado como a automação de um processo de negócio, em sua totalidade ou não, na qual documentos, informação ou tarefas são passadas de um participante para outro para execução de uma ação, de acordo com um conjunto de regras. No contexto de experimentação científica, esses *workflows* passam a ser chamados de *workflows* científicos e são usados em diversas áreas da ciência como, por exemplo, química, física, biologia, geologia, etc. (Ludascher et al. 2006). Em seguida, com o propósito de automatizar a construção e execução desses *workflows*, foram

criados sistemas gerenciadores de *workflows* científicos (*SGWfC*). A partir desses sistemas, cientistas não precisam mais realizar tarefas maçantes como, por exemplo, disparar atividades individualmente ou transferir dados de um programa para outro (Stevens et al. 2003a). Esse tipo de tarefa fica a cargo do *SGWfC*, que passa a realizar a orquestração do *workflow*. Além disso, a utilização de *SGWfC* garante um melhor compartilhamento, reutilização e repetibilidade dos experimentos (Goderis et al. 2005).

1.2 A informação histórica em experimentos científicos

Com a utilização de recursos computacionais para a execução de experimentos e *SGWfC*, outros problemas surgiram (Zhao 2007). Um deles é a perda de conhecimento do cientista sobre o experimento, devido à delegação das tarefas para o computador que, geralmente, realiza ações que não são documentadas adequadamente. A maioria dos *SGWfC* foca mais na etapa de execução e não possui muitas funcionalidades para a documentação do *workflow*. Somado a isso, a complexidade dos novos experimentos faz com que o cientista, ao final de uma execução de experimento, se depare com uma grande quantidade de dados que ele não sabe de onde vieram ou como foram gerados. Em outras palavras, o cientista não tem conhecimento por completo da **proveniência** desses dados.

De forma geral, proveniência é todo tipo de informação que descreve a origem de um objeto (Freire et al. 2008). Uma pintura, por exemplo, possui informações de proveniência, tais como: o nome do pintor que realizou a pintura, a data de criação, o tipo de tinta ou papel que foi utilizado, etc. Tratando-se de *workflows* científicos, proveniência fornece informação histórica acerca dos dados manipulados em um *workflow*. Essa informação histórica descreve os dados que foram gerados, apresentando os seus processos de transformação a partir de dados primários e intermediários. O controle deste tipo de informação é extremamente importante, uma vez que proporciona aos cientistas uma variedade de aplicações. Por exemplo, a partir de proveniência, é possível garantir a qualidade dos dados gerados, já que é possível observar os dados que foram utilizados (seus ancestrais) e determinar se estes são confiáveis ou não. Outros exemplos são a possibilidade de realização de auditorias para verificar quais recursos estão sendo realmente utilizados; repetição da derivação dos dados sem a necessidade da reexecução completa do *workflow*; definição explícita de

pessoas ou entidades responsáveis pelos dados utilizados no experimento; entre outras aplicações (Simmhan et al. 2005).

Fazendo um comparativo do cenário de experimentação científica com engenharia de software (*ES*), um *SGWfC* pode ser considerado equivalente a uma ferramenta *CASE* (Finlay e Mitchell 1994) que possibilita a programação do experimento em alto nível, através do encadeamento de programas (ou atividades) que seguem um determinado fluxo e lógica. O conceito de proveniência, por sua vez, é então comparável ao conceito de rastreabilidade em engenharia de *software* (*ES*) (Kowalczykiewicz e Weiss 2002), utilizado em controle de processos (Shinskey 1979), para vincular programas, insumos e dados produzidos, com o intuito de possibilitar uma análise posterior. Este conceito é utilizado também no processo de desenvolvimento de software para antecipar a identificação de decisões ruins de projeto e para prover uma visão de alto nível das dependências do sistema (Settimi et al. 2004).

1.3 Dificuldades na gerência de proveniência

Para se obter benefícios das informações de proveniência, é necessário que estas sejam capturadas, modeladas e armazenadas de modo integrado para consulta posterior. Mesmo sendo tratado por vários *SGWfC* e diversos trabalhos na literatura (Cohen et al. 2006) (Simmhan et al. 2005), a gerência de proveniência possui várias questões em aberto. Um dos problemas ainda em aberto é a falta de concordância do que deve ser capturado no que diz respeito ao tipo de informação de proveniência, além de uma definição clara de como essa captura deve ser feita. A captura de dados de proveniência se torna mais complexa quando o *workflow* é executado em diferentes ambientes de execução, ou seja, em um ambiente distribuído e heterogêneo. Neste cenário, as informações de proveniência precisam ser coletadas, armazenadas e integradas a partir de fontes distintas.

A utilização de ambientes de execução distribuída vem se tornando comum devido a diversos fatores (Mattoso et al. 2008). O primeiro deles é a obtenção de recursos necessários para a execução do experimento. Por exemplo, as aplicações necessárias para compor a execução de um experimento podem estar disponíveis em ambientes específicos e não necessariamente podem ser movidas ou executadas localmente. De maneira análoga, dados utilizados em um experimento podem estar

distribuídos em diversas máquinas. Em certos casos (e.g., dados de larga escala), é mais vantajoso que sejam acessados e processados remotamente (em suas máquinas locais) do que transportados pela rede. Outro fator motivador é a possibilidade de otimização de recursos na execução de *workflows* de larga escala (Mattoso et al. 2008) com a utilização de paralelismo e distribuição de processamento usando clusters (Buyya 1999), grades computacionais (Yu e Buyya 2005), computação em nuvem (Weiss 2007), dentre outros.

A forma de execução de um experimento em um ambiente distribuído e heterogêneo pode ser classificada em diversos cenários. Cada cenário possui suas características que facilitam ou dificultam o gerenciamento de proveniência. Traçando dois eixos em um plano representando a complexidade de distribuição (eixo horizontal) e a complexidade de heterogeneidade (eixo vertical) é possível delinear cenários de complexidade de gerenciamento de proveniência em cada um dos quadrantes, de acordo com a Figura 1.1.

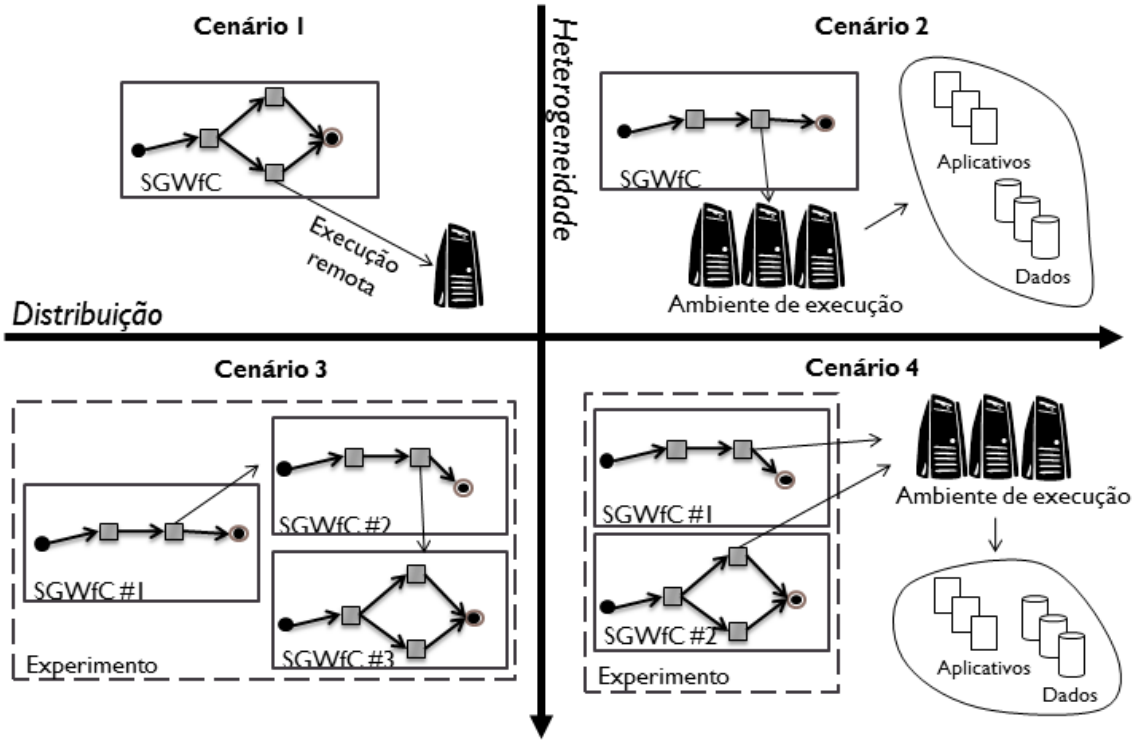

Figura 1.1 – Tipos de cenários em ambientes distribuídos e heterogêneos

Quando o experimento tem baixa complexidade de distribuição (cenários 1 e 3), as atividades do *workflow* que são executadas remotamente são implementadas por mecanismos de distribuição simples (*e.g.*, chamadas de procedimentos remotos, serviços web, grades computacionais, etc.). Nestes casos, um *SGWfC* que provê um

mecanismo de proveniência básico é capaz de capturar os dados de proveniência das atividades do *workflow*. Por outro lado, atividades de *workflow* que são executadas em cluster ou nuvem computacional, trata-se de experimentos que possuem alta complexidade de distribuição (cenários 2 e 4). Se o *SGWfC* não estiver preparado para esta situação, informações de proveniência importantes sobre a execução das atividades são perdidas (*e.g.*, tempo de início e fim de execução, mensagens de erro, dados gerados e consumidos).

Analisando a questão da heterogeneidade, uma baixa complexidade significa experimentos que são executados em um único *SGWfC* (cenários 1 e 2). Por outro lado, uma alta complexidade de heterogeneidade é verificada em experimentos que possuem seus *workflows* fragmentados em vários *workflows* menores, a fim de serem executados em *SGWfC* diferentes (cenários 3 e 4). Nesses cenários, cada *SGWfC* lida com o gerenciamento de proveniência de forma descentralizada, ou seja, cada sistema trata proveniência em uma determinada granularidade, armazena as informações em uma linguagem específica e, em piores casos, não provê suporte algum à proveniência.

A fragmentação do *workflow* de um experrimento pode ser explicada devido aos *SGWfC* possuírem propriedades específicas e a sua aplicação em apenas determinadas regiões do *workflow* possa ser mais vantajosa do que a sua aplicação como um todo. Por exemplo, um experimento pode possuir características que sejam necessárias a execução de parte do *workflow* em um *SGWfC* que tenha mecanismos avançados de paralelismo, e a execução de uma outra parte em um *SGWfC* que disponibilize recursos avançados de visualização de dados. Outros fatores motivadores dessa fragmentação podem ser devido a: questões organizacionais que implicam que um *workflow* seja executado separadamente em diversos setores de uma organização; *workflows* que possuem atividade manual e, com isso, precisam ser fragmentados em duas partes (antes e depois da atividade manual) para poderem ser executados no *SGWfC*.

Uma possível estratégia para resolver o problema de gerenciamento de proveniência em ambientes distribuídos e heterogêneos consiste em transferir essa responsabilidade para um único sistema específico. Este sistema seria responsável pela captura, armazenamento e fornecimento de informações de proveniência de um *workflow*. A ideia é promover o gerenciamento de proveniência, passando do gerenciamento no nível de *workflow*, ou seja, centralizado nos mecanismos de proveniência dos *SGWfC*, para um gerenciamento de proveniência mais abrangente no

nível de experimento, a partir de um mecanismo de proveniência único e integrado que facilite a captura e o acesso de todos os dados de proveniência de um experimento.

Com isso, de maneira mais formal, a questão de pesquisa ou hipótese geral investigada nesse trabalho é:

A promoção do gerenciamento de proveniência do nível de workflow para o nível do experimento torna mais produtiva a análise de proveniência por parte dos cientistas em ambientes distribuídos e heterogêneos?

Na literatura, alguns trabalhos (Da Cruz et al. 2008) (Simmhan et al. 2006) (Groth et al. 2006) (Groth et al. 2008) possuem as mesmas preocupações e defendem uma abordagem de proveniência que seja independente de *SGWfC*. Além disso, existe uma iniciativa de Moreau *et al.* (2007), que estabelece um modelo de proveniência padrão, chamado *Open Provenance Model* (OPM), que define uma representação genérica de proveniência para qualquer entidade, não somente *workflows* científicos. No entanto, o problema desta estratégia é que os *SGWfC* e o sistema gerenciador de proveniência precisam se comunicar para trocar informações.

Alguns trabalhos (Simmhan et al. 2006) (Munroe et al. 2006) (Lin et al. 2008) propõem uma série de adaptações manuais nas atividades ao longo do *workflow* para viabilizar essa comunicação. No entanto, esses trabalhos não foram concebidos para gerenciar a proveniência no nível do experimento. Além disso, analisando pela perspectiva do cientista, as soluções propostas apresentam deficiências, pois, em primeiro lugar, a maioria dos cientistas não possui conhecimentos profundos em computação. Em segundo lugar, as atividades utilizadas em um *workflow*, na maioria das vezes, são de terceiros ou proprietárias, de código fechado, o que dificulta ainda mais a adaptação. Esses trabalhos não foram concebidos para gerenciar a proveniência no nível do experimento, estando focados na gerência no nível de *workflow*. Portanto, é necessário que o *workflow* seja adaptado de uma maneira menos intrusiva no código das aplicações e mais automatizada possível, sem sobrecarregar o cientista com tarefas computacionais que não estão ao seu alcance.

1.4 Objetivo

O objetivo deste trabalho é viabilizar a avaliação da hipótese de pesquisa levantada através da concepção e elaboração de uma abordagem, intitulada

ProvManager, que torna possível o gerenciamento de proveniência de experimentos em ambientes distribuídos e heterogêneos. A abordagem *ProvManager* tem como propósito promover o gerenciamento de proveniência do nível de *workflow* para o nível de experimento através da modelagem, captura e armazenamento das informações de proveniência que estão localizadas de maneira descentralizada e não integrada em um ambiente distribuído e heterogêneo.

Além desse objetivo principal, *ProvManager* possui os seguintes objetivos secundários:

a) Ser independente de tecnologia de *SGWfC*;
b) Não sobrecarregar o cientista com tarefas de configuração do sistema;
c) Provimento de mecanismo de visualização em tempo real (monitoramento) dos dados de proveniência capturados; e
d) Viabilizar o acesso integrado aos dados distribuídos de proveniência do experimento.

1.5 Organização

Este trabalho está organizado em outros cinco capítulos, além deste capítulo de introdução.

O Capitulo 2 realiza uma revisão da literatura sobre proveniência, discutindo suas origens e detalhando no contexto de experimentação científica. Além disso, uma análise sobre trabalhos de gerência de proveniência, incluindo abordagens e ferramentas, é conduzida.

O Capítulo 3 apresenta a abordagem para o gerenciamento de proveniência de experimentos científicos proposta por este trabalho, denominada ProvManager, detalhando as soluções adotadas para solucionar os problemas de gerenciamento de proveniência de experimentos que são executados em ambientes heterogêneos e distribuídos.

O Capítulo 4 discute os detalhes de implementação do protótipo (apresentando sua arquitetura, as principais tecnologias utilizadas, padrões adotados, etc.) que foi desenvolvido como prova de conceito da abordagem ProvManager. Além disso, um experimento real do domínio de cristalografia é apresentado e utilizado para ilustrar as funcionalidades do protótipo.

O Capítulo 5 descreve o estudo de observação que foi conduzido na abordagem ProvManager utilizando o protótipo desenvolvido para avaliar a questão de pesquisa deste trabalho. O objetivo deste estudo foi analisar se o conjunto de dados de proveniência coletados pela abordagem *ProvManager* de um experimento executado em um ambiente heterogêneo auxilia o cientista no processo de análise de proveniência.

O Capítulo 6 apresenta uma conclusão, listando as contribuições deste trabalho, mostrando as limitações existentes e vislumbrando as oportunidades de novos trabalhos que podem ser concebidos a partir deste.

Finalmente, os Anexos apresentam algumas documentações complementares utilizadas nos capítulos. Os Anexos A e B descrevem as operações da API de publicação de proveniência da abordagem ProvManager, referenciada no Capítulo 3. Os Anexos C, D, E, F, G, H, I, J incluem os documentos utilizados no estudo de observação descrito no Capítulo 5.

Capítulo 2. Dados de Proveniência

2.1 *Introdução*

Segundo o dicionário *Oxford* (Simpson e Weiner 1989), proveniência é definida como:

(i) *A fonte ou origem de um objeto, sua história ou pedigree;*

(ii) *Um registro das últimas variações e passagens de um item através de seus vários donos.*

O conceito de proveniência teve maior destaque primeiramente no estudo das belas artes, arqueologia, paleontologia, dentre outras disciplinas históricas, onde existe a necessidade de se definir a história bem documentada de um achado ou de um objeto de valor (Moreau et al. 2008). Objetos que não possuem uma história confiável e comprovada podem ser tratados com ceticismo por aqueles que os estudam e analisam (Moreau et al. 2008). Por exemplo, uma obra de arte, que não tem uma descrição bem definida da passagem de todos os seus donos, pode perder a sua credibilidade. Um vinho, para ser considerado bom, deve possui uma história bem documentada de condições de armazenamento ao longo dos anos. Da mesma forma, no comércio de gado, vendedores precisam realizar um maior controle do histórico de vida dos animais comercializados para garantir que os mesmos são de boa procedência. No caso de livros, o estudo de proveniência refere-se à análise de propriedade intelectual de cópias de livros, dentre outros exemplos.

Em um contexto computacional, proveniência é a informação que auxilia a determinar o processo de derivação dos dados produzidos a partir de suas fontes originais (Simmhan et al. 2005). Um dos primeiros indícios de estudo de proveniência em ciência da computação se dá na área de banco de dados na década de 70 (Zhao 2007), com trabalhos de atualização de visões (Stonebraker 1975). Mais tarde, o conceito de proveniência aparece também em engenharia de software, ganhando o nome de rastreabilidade, sendo utilizado no processo de desenvolvimento de software para

facilitar a identificação prévia de decisões ruins de projeto e para prover uma visão de alto nível das dependências do sistema (Settimi et al. 2004).

Este capítulo apresenta os principais conceitos de proveniência em experimentação científica que foram pesquisados através de uma revisão da literatura. Além disso, algumas abordagens de gerenciamento de proveniência são apresentadas e, no final, um comparativo entre as mesmas é realizado. A Seção 2.2 discute sobre proveniência no contexto de experimentação científica. A Seção 2.3 descreve os principais componentes da arquitetura de uma abordagem de gerenciamento de proveniência. A Seção 2.4 apresenta algumas abordagens de gerenciamento de proveniência presentes na literatura. A Seção 2.5 faz um comparativo das abordagens apresentadas, utilizando um conjunto de critérios que foram estabelecidos através de uma revisão da literatura.

2.2 Proveniência em experimentação científica

Especificamente em experimentação científica, a proveniência tem seu papel fundamental, pois auxilia não somente à interpretação e compreensão de resultados a partir da análise da sequência de passos que foram conduzidos, como também viabiliza a repetibilidade de um experimento científico (Freire et al. 2008). Proveniência passa a ganhar maior destaque em experimentação científica devido ao aumento da complexidade dos experimentos executados. Resultados de experimentos mais complexos e importantes passaram a ser publicados, porém a história de derivação dos dados de um experimento não era apresentada de maneira clara e concisa, o que causava descrença por outros cientistas que estudavam o experimento, uma vez que não conseguiam repetir os mesmos resultados (Bose e Frew 2005).

Estes experimentos mais complexos são denominados experimentos *in-silico*. Neste tipo de experimentos, tanto o ambiente, os objetos, quanto os participantes são simulados computacionalmente (Travassos e Barros 2003)(Zhao et al. 2004). Com isso, para dar suporte a esta nova categoria de experimentos que utilizam fortemente ferramentas computacionais, o conceito de *workflow* científico passa a ser utilizado. O objetivo era melhorar a abstração dos experimentos a partir da representação de sequências de atividades que manipulam dados de modo a atingir um determinado resultado. Além disso, sistemas de gerência de *workflows* científicos (*SGWfC*) (Taylor et al. 2006) são concebidos para prover um ambiente integrado para simplificar o

trabalho de programação realizado pelos cientistas para orquestrar *workflows* científicos durante a elaboração de um experimento *in silico*.

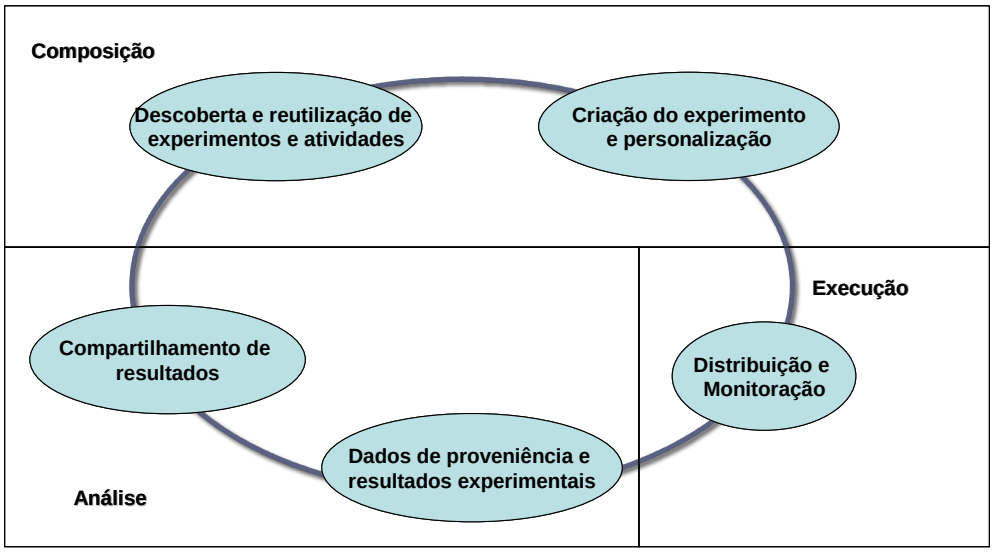

Figura 2.1 - Ciclo de vida de um experimento *in silico*

Segundo a visão de Oinn et al. (2007), o ciclo de vida de um experimento *in silico* é representado de acordo com a Figura 2.1. A interação dos *workflows* durante o ciclo de vida de um experimento *in silico* pode ser classificada em três principais fases (Mattoso et al. 2009):

(i) **Composição:** Para cada configuração de *workflow*, a composição trata da conceituação do escopo de uma atividade, da seleção de um programa ou componente adequado para atuar como atividade e também a configuração do fluxo de atividade. De acordo com Gil (2007), nesta fase, o foco é na construção do *workflow* em diferentes níveis de abstração, usando diferentes modos de representação como *script* textual ou modelagem gráfica.

(ii) **Execução:** Esta fase é focada na execução dos *workflows* propriamente ditos, incluindo a distribuição de dados e de programas (Couvares et al. 2007) e o monitoramento das execuções (Oinn et al. 2004).

(iii) **Análise:** Nesta fase, o foco é na avaliação dos resultados experimentais obtidos nas execuções dos *workflows*, que incluem atividades como visualização de dados, consultas e proveniência. Além disso, todos os resultados do experimento são documentados, para que possam ser

compartilhados com outros cientistas que queiram reexecutar o experimento (Freire et al. 2008).

De acordo com as fases descritas acima, a funcionalidade de consulta de proveniência está localizada na fase de análise, já que é nesta fase que o cientista realiza a avaliação da execução do *workflow* e necessita de informações sobre a origem dos dados produzidos. No entanto, para tornar os dados de proveniência disponíveis na fase de análise, é necessário que um mecanismo de proveniência participe também nas etapas de composição e execução. Na fase de composição, por exemplo, o cientista precisa definir que tipos de informações de proveniência são importantes para o experimento. Essa definição pode evoluir de acordo com cada interação do ciclo de vida, ou seja, quando novas atividades são adicionadas ao *workflow*, ou a partir de novas características ou restrições incorporadas ao experimento. Por outro lado, durante a fase de execução, o mecanismo de proveniência precisa estar em funcionamento, coletando, armazenando e fazendo as associações dos dados gerados pelo *workflow*.

Segundo Freire et al. (Freire et al. 2008), os dados de proveniência podem ser classificados em duas categorias: prospectiva e retrospectiva. A proveniência prospectiva representa os dados relacionados à definição do experimento, ou seja, informações sobre a especificação do *workflow* (e.g., parâmetros de entrada e saída de cada atividade do *workflow* e ordem de execução das atividades) e anotações feitas pelo cientista, entre outros. A proveniência retrospectiva mantém informações sobre as atividades executadas e os dados produzidos. Em suma, a proveniência prospectiva se preocupa com os dados de proveniência do experimento no momento de desenvolvimento, enquanto a proveniência retrospectiva se preocupa com os dados de proveniência do experimento no momento de execução.

Proveniências prospectivas e retrospectivas podem ser consideradas complementares, pois um mecanismo de proveniência precisa trabalhar com as duas formas, a fim de obter todas as informações necessárias sobre proveniência. A Figura 2.2 e a Figura 2.3 apresentam dois cenários onde informações de proveniência prospectiva e retrospectiva precisam ser analisadas em conjunto, para se obter uma boa interpretação dos resultados de um experimento. Nos dois cenários, a proveniência prospectiva é representada pela especificação do *workflow*, utilizando a notação de diagramas de atividades da UML (OMG 2010), enquanto que a proveniência

retrospectiva é representada, por questões de simplicidade, por um *log* de execução. O primeiro cenário, descrito na Figura 2.2, mostra que uma má interpretação pode ser feita quando apenas a proveniência retrospectiva é analisada. A partir dela, é possível imaginar que a atividade B precede a atividade A, porém isso não é verdade, já que, se analisarmos a proveniência prospectiva, é possível notar que as duas atividades podem ser iniciadas em paralelo. Na Figura 2.3, a situação oposta é apresentada. Analisando apenas a proveniência prospectiva, não é possível definir qual atividade será executada após o ponto de decisão (atividade B ou C?). Esta informação só é possível de ser descoberta analisando a proveniência retrospectiva, que indica a execução da atividade B.

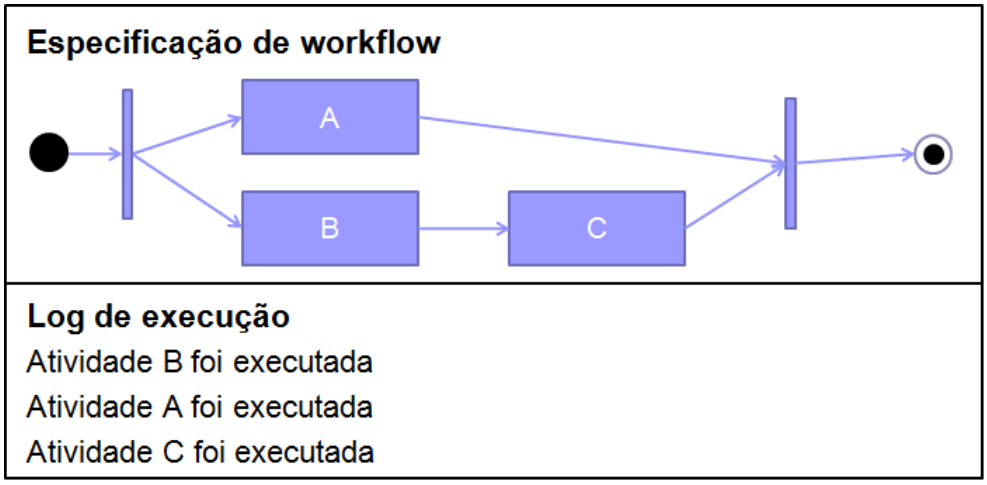

Figura 2.2 - Primeiro cenário de proveniência prospectiva e retrospectiva

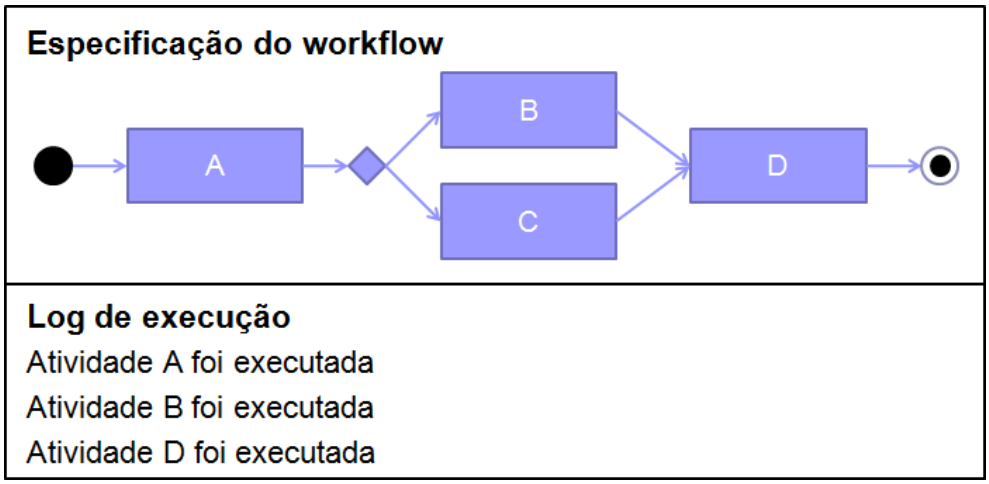

Figura 2.3 - Segundo cenário de proveniência prospectiva e retrospectiva

2.3 Abordagens para gerência de proveniência

Nesta seção, é feita uma discussão sobre as características gerais de abordagens de gerenciamento de proveniência. Segundo Freire *et al.* (2008), o gerenciamento de proveniência pode ser dividida em três componentes principais:

(i) **Mecanismo de captura:** responsável pela coleta de informações de proveniência do experimento;

(ii) **Modelo de representação dos dados de proveniência (modelo de proveniência):** define como as informações de proveniência da abordagem são representadas ou modeladas; e

(iii) **Infraestrutura de armazenamento, acesso e consulta:** define as questões diversas relacionadas a base da estrutura da abordagem, tais como estratégias de armazenamento, definição de formas de acesso, mecanismo de consultas aos dados de proveniência, dentre outras.

As seções a seguir discutem sobre estes três componentes.

2.3.1 Mecanismo de captura

O mecanismo de captura é responsável pela coleta de informações de proveniência do experimento. Estas informações podem ser tanto do tipo prospectiva quanto retrospectiva. No caso da proveniência prospectiva, este mecanismo pode capturar informações sobre a especificação do *workflow*, anotações do usuário, etc. Já no caso da proveniência retrospectiva, o mecanismo de captura, durante a execução do *workflow*, pode coletar os dados produzidos, as intervenções feitas pelo usuário, etc.

Existem diversos tipos de mecanismos de captura de proveniência na literatura. Estes mecanismos apresentam diversas estratégias que os tornam mais apropriados em determinados cenários. De forma a facilitar a análise desses mecanismos, podemos usar a classificação de Freire *et al.* (2008) que determina três níveis de funcionamento de um mecanismo de captura de proveniência:

- Nível de sistema operacional (*SO*);
- Nível de *workflow;*

- Nível de atividade.

Mecanismos de captura que atuam no nível de *SO* utilizam as funcionalidades do sistema operacional para capturar informações de proveniência. Mecanismos deste nível, por exemplo, podem utilizar a *API* de sistema de arquivos do *SO* para capturar os dados produzidos pelas atividades do *workflow* e descobrir as suas dependências. Outro exemplo é a utilização do registro de chamadas do *SO* (*system call tracer*) para saber quais usuários estão executando quais processos. A principal vantagem desta abordagem é a independência do *SGWfC*, o que viabiliza a sua aplicação em qualquer tipo de cenário de ambiente distribuído. No entanto, esta característica pode ser considerada uma desvantagem também, uma vez que os dados recolhidos por este mecanismo estão em um grão muito fino, tornando necessário um pré-processamento desses dados para tornarem-se informações "úteis". Por exemplo, é necessário realizar um pré-processamento nos dados do registro de chamadas do *SO* para poder definir a ordem de execução das atividades do *workflow* e os seus dados gerados.

A captura no nível de *workflow* é uma das estratégias mais utilizadas pelas abordagens de proveniência. Nesta abordagem, o *SGWfC* é responsável por coletar todas as informações de proveniência. A vantagem deste nível é a facilidade de aplicação, uma vez que o mecanismo está acoplado diretamente ao *SGWfC*, o que torna mais simples a captura dos dados trafegados no *workflow*. Entretanto, dados que são gerados internamente pelas atividades e que não são explicitados na descrição do *workflow* não são capturados pelo mecanismo. Outra desvantagem deste nível é a dependência do mecanismo de captura do *SGWfC*, uma vez que foi construído exclusivamente para utilizar as funcionalidades de um *SGWfC* específico. Para utilizar o mesmo mecanismo de captura em outro *SGWfC*, será necessário ter que implementá-lo novamente. Com isso, a aplicação de mecanismos deste nível é aconselhável apenas em cenários onde somente um tipo de *SGWfC* é utilizado e as informações de proveniência disponíveis no *SGWfC* são suficientes para a análise do experimento.

Os mecanismos de captura que trabalham no nível de atividade estabelecem que cada atividade do *workflow* é responsável por coletar as suas próprias informações de proveniência. Os mecanismos que atuam neste nível possuem a vantagem de serem independentes de *SGWfC*, da mesma forma daqueles que trabalham no nível de *SO*. Entretanto, a informação capturada é mais precisa, já que o mecanismo está acoplado

diretamente à atividade. Além disso, esse mecanismo possibilita que qualquer tipo de informação gerada por qualquer atividade do *workflow* seja capturado, diferentemente dos mecanismos que trabalham no nível de *SGWfC*, que conseguem capturar somente as informações que são explicitadas na descrição do *workflow*. O principal problema deste nível é que atividades do *workflow* precisam ser adaptadas para suportar as funcionalidades do mecanismo de captura. O que pode ser prejudicial se esta tarefa de adaptação for delegada ao cientista.

2.3.2 Modelo de Proveniência

O modelo conceitual de proveniência define como as informações de proveniência são representadas ou modeladas. Um modelo de proveniência pode representar informações de proveniência tanto prospectiva quanto retrospectiva. Além disso, esses modelos podem incluir anotações para prover mais semântica aos dados de proveniência.

Geralmente, cada sistema de proveniência define o seu próprio modelo com o objetivo de focar características específicas ou, simplesmente, por conveniência. O resultado disso é uma grande quantidade de modelos de proveniência na literatura que diferem entre si, embora possuam um conjunto comum de informações. Essa diversificação de modelos pode prejudicar a tarefa de cientistas que manipulam vários sistemas de gerenciamento de proveniência. Em cada sistema, o cientista tem que saber quais informações de proveniência são suportadas e o modo como estas informações são acessadas. Além disso, essa pluralidade de modelos prejudica também a comunicação de sistemas que queiram trocar informações de proveniência. Uma possível solução é a construção de tradutores para viabilizar a comunicação entre os sistemas. Porém, sem um modelo padrão, um sistema de proveniência deve prover tradutores específicos para cada sistema que for se comunicar.

Tendo em vista esse problema, Moreau *et al.* (2007) definiram um novo modelo de proveniência, chamado *Open Provenance Model* (*OPM*), que possui os seguintes objetivos:

- Ser capaz de representar digitalmente qualquer tipo de aplicação de proveniência;
- Ser um modelo independente de tecnologia; e

- Permitir a troca de informações de proveniência entre sistemas.

A ideia principal do *OPM* é ser o modelo referência para qualquer sistema de proveniência. Além disso, a proposta é utilizar o *OPM* como linguagem de comunicação de troca de dados de proveniência entre sistemas. Alguns w*orkshops* especializados de proveniência vêm sendo realizados para testar a capacidade dos sistemas de proveniência em exportar e importar dados de proveniência utilizando a especificação do *OPM*. Um exemplo desses *workshops* é o *Provenance Challenge* (ProvChallenge 2011).

O *OPM* representa os dados de proveniência através de um grafo causal, que registra o histórico do fluxo de execução do *workflow*. O *OPM* é baseado em três entidades principais: Artefato, representação imutável de um dado; Processo, ação ou conjunto de ações realizadas em ou causadas por artefatos que resultam em novos artefatos; e Agente, entidade contextual que age sobre um processo, habilitando, facilitando, controlando e afetando sua execução. A Figura 2.4 ilustra essas três entidades e suas possíveis formas de relacionamento, chamadas também de dependências.

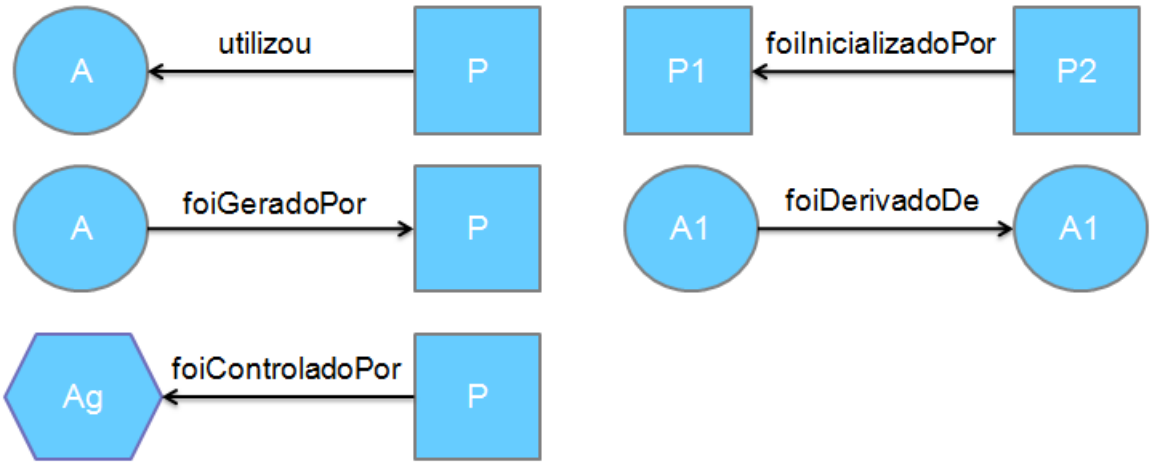

Figura 2.4 - Entidades e relacionamentos do *OPM* (adaptado de Moreau *et al.* (2007))

Na parte da esquerda da Figura 2.4, os dois primeiros relacionamentos expressam que um processo *usou* um artefato e que um artefato *foi gerado por* um processo. Essas duas dependências representam relacionamentos de derivação de dados. O terceiro relacionamento indica que um processo *foi controlado por* um agente. Diferente dos outros dois primeiros relacionamentos citados, este representa um relacionamento de controle. O quarto relacionamento é usado em situações onde não se

sabe com exatidão quais artefatos foram utilizados por um processo, contudo se sabe que este processo utilizou algum artefato gerado por outro processo. Com isso, pode-se dizer que o processo *foi inicializado por* outro processo. De maneira análoga, o quinto relacionamento é utilizado em situações que não se sabe qual processo gerou um determinado artefato, porém se sabe que esse artefato *foi derivado de* outro artefato. Esses dois últimos relacionamentos são recursivos e podem ser implementados a partir de regras de inferência. Portanto, a partir deles, é possível determinar a sequência de execução dos processos ou o histórico de derivação que originou um dado.

O *OPM* é uma contribuição de vários autores da área de proveniência e promete ser o modelo de referência para qualquer solução de proveniência. Todavia, este modelo possui algumas características divergentes com alguns modelos definidos anteriormente, que podem causar certa resistência na sua aceitação. Alguns modelos de proveniência (Groth et al. 2006, Simmhan et al. 2006) consideram de suma importância que informações internas de uma atividade sejam armazenadas, a fim de que um experimento seja analisado adequadamente. Essas informações internas das atividades são, por exemplo, estados de execução, dados intermediários gerados, etc. O *OPM* não provê suporte para esse tipo de informação. Além disso, o *OPM* não provê suporte à representação de informações sobre a especificação do *workflow* e, por isso, pode ser classificado como apenas um modelo de proveniência retrospectiva.

2.3.3 Infraestrutura de armazenamento, acesso e consulta

Durante o estabelecimento de um mecanismo de proveniência, algumas questões merecem um tratamento especial. Por exemplo, questões que envolvem estratégias de armazenamento, definição de formas de acesso e mecanismo de consultas aos dados de proveniência. Estas questões, dentre outras secundárias, formam a infraestrutura de um mecanismo de proveniência.

Com relação à questão de armazenamento, várias soluções são empregadas. Estas soluções são afetadas por requisitos não funcionais, tais como facilidade de utilização, restrições de ambiente, etc. Por exemplo, um sistema de arquivos é uma boa solução para usuários, pois é um mecanismo nativo do sistema operacional e, com isso, não necessita de qualquer instalação de programas adicionais para utilizá-lo. A mesma situação não acontece com uma solução que utilize banco de dados, uma vez que é necessário que o usuário faça a instalação de um sistema gerenciador de banco de dados

(*SGBD*), além de realizar configurações adicionais como, por exemplo, a instanciação do esquema no *SGBD*. Em alguns casos, uma solução que utilize os dois mecanismos é a melhor alternativa. Por exemplo, o trabalho de Groth *et al.* (2006) sugere a utilização de mecanismos de referência (ponteiros) para armazenar os dados de proveniência. Utilizando um *SGBD* para armazenar as referências, os arquivos gerados durante a execução de um experimento não seriam replicados em um repositório central, e sim referenciados para o local onde foram gerados. Essa estratégia é interessante em experimentos que trabalham com grande quantidade de dados, tornando a sua replicação inviável por motivos de desempenho, devido à transferência de grande quantidade de informação pela rede, ou simplesmente por motivos de redução de custos através da diminuição do número de unidades de armazenamento.

Quanto à questão de acesso, é importante se discutir que tipos de interfaces serão disponibilizados. Uma interface gráfica, por exemplo, ao contrário do uso de uma interface de linha de comando, é essencial para os usuários que não têm muita experiência em computação. O uso de uma interface *web* é interessante pra aumentar a portabilidade do sistema, já que existem casos em que os dados dos experimentos são compartilhados por diversos usuários que podem estar localizados geograficamente distantes e/ou utilizarem ambientes diferentes. Algumas soluções definem *API* para permitir a comunicação com outros sistemas. Como foi citado anteriormente, um exemplo é o *OPM* que está sendo utilizado como a linguagem de comunicação entre sistemas de proveniência. Além disso, o acesso tem que ser feito de maneira controlada e segura em situações onde informações privadas são extraídas do experimento. Com isso, mecanismos de controle de acesso e políticas de segurança devem ser aplicados.

Com relação às formas de consulta, diferentes soluções de proveniência definem linguagens de consulta estreitamente ligadas aos modelos de armazenamentos utilizados. No entanto, isto é prejudicial para usuários inexperientes em computação (*e.g.*, cientistas de áreas não tecnológicas), que têm que criar, por exemplo, consultas complexas em *SQL* ou em *Prolog*. Com isso, é aconselhável que linguagens de alto nível de abstração ou mecanismos sejam desenvolvidos para facilitar o processo de consulta de dados de proveniência. Segundo Anand *et al.* (2010), existem duas formas importantes para acessar e explorar informações de proveniência: mecanismos de navegação (*browsing*), visualizando e navegando grafos de dependências de dados e de processos, e mecanismos de consulta (*querying*), para selecionar porções de grafos de

proveniência ou para determinar se certos caminhos existem. Além disso, esses dois mecanismos podem ser usados em conjunto, como é o caso do trabalho de Anand *et al.* (2010), que disponibiliza um navegador de proveniência onde o usuário pode visualizar um grafo de proveniência em três visões diferentes (*dependency history view*, *collection history view*, *invocation dependency view*), podendo refinar os resultados através de uma linguagem de consulta similar a *XPath* (W3C 1999). Outro exemplo de refinamento de resultados é o trabalho de Cohen *et al.* (2006), que propõe simplificações na visualização dos dados de proveniência através do uso de filtros, possibilitando ao usuário visualizar os dados de entrada e saída de uma única atividade ou de regiões complexas compostas de mais de uma atividade (i.e., *sub-workflows)*.

2.4 Abordagens de gerenciamento de proveniência

Nesta seção são apresentadas algumas abordagens de sistemas de gerenciamento de proveniência. Dentre estas abordagens, algumas foram implementadas em protótipos e outras foram implementadas e integradas em *SGWfC* consolidados. Vale destacar que são analisadas apenas abordagens que contemplam ao menos um conjunto mínimo de funcionalidades básicas de sistema de gerenciamento de proveniência, que são: capturar, armazenar e prover mecanismo de consulta aos dados de proveniência. Além deste critério, as abordagens foram selecionadas pelo grau de reconhecimento da comunidade e/ou pela semelhança com o problema tratado por este trabalho. As abordagens selecionadas são: *Karma*, *PASOA*, *Wings/Pegasus*, *VisTrails*, *myGrid/Taverna*, *Kepler* e *Swift*.

Para guiar a discussão sobre as abordagens, alguns critérios gerais são utilizados. Estes critérios foram definidos a partir da revisão da literatura e através de uma análise das principais contribuições e limitações das abordagens descritas nesse capítulo. Além disso, alguns destes critérios foram estipulados a partir das principais características do problema de gerenciamento de proveniência que este trabalho tenta solucionar. Estes critérios gerais são:

- **Suporte a experimentos executados em ambiente distribuídos e heterogêneos:** indica se a abordagem é capaz de gerenciar os dados de proveniência de experimentos executados em ambientes distribuídos e heterogêneos. Como foi abordado na Seção **Error! Reference source not**

found., quatro cenários foram definidos de acordo com a complexidade de distribuição e heterogeneidade de execução de um experimento. Com isso, uma característica importante é verificar se a abordagem é capaz de trabalhar de maneira independente de um *SGWfC*.

- **Modelo de proveniência:** analisa os tipos de dados de proveniência suportados pela abordagem. De forma geral, analisa se a abordagem suporta tanto proveniência prospectiva quanto retrospectiva. De maneira mais específica, é analisado se o modelo trabalha com ontologias, anotações, especificação de *workflow*, controle de evolução de histórico de *workflow*, informações de execução, etc.

- **Característica do mecanismo de captura de proveniência:** discute sobre o nível de funcionamento do mecanismo de captura. Como foi citado anteriormente na Seção 2.3.1, um mecanismo de captura de proveniência pode ser classificado em três níveis de funcionamento: nível de *workflow*, nível de *SO* e nível de atividade.

- **Análise de dados de proveniência:** indica se a abordagem disponibiliza mecanismos de acesso e ferramentas gráficas que facilitem a exploração dos dados de proveniência coletados, tais como mecanismos de navegação em grafos de proveniência, mecanismos de consultas, mecanismos de monitoramento dos dados de proveniência em tempo real, etc.

- **Encargos de configuração da abordagem para seu funcionamento:** verifica o grau de complexidade das tarefas de configuração delegadas ao usuário que são necessárias para o funcionamento da abordagem. Essa característica é importante, pois uma abordagem de gerenciamento de proveniência pode ser utilizada por usuários (na maioria das vezes, cientistas) que não possuem muita experiência em computação, além de ser uma tarefa repetitiva e maçante. Geralmente, essa característica varia de acordo com os valores das características previamente citados. Por exemplo, um mecanismo de captura de proveniência trabalhando no nível de atividade implica que as atividades do *workflow* sejam adaptadas ao mecanismo. Essa tarefa de adaptação das atividades, porém, pode ser delegada ao usuário.

2.4.1 Karma

Karma (Simmhan et al. 2006) é um *framework* para captura de proveniência de experimentos científicos focados em *workflows* que são executados em ambientes distribuídos de grade computacional. Neste tipo de ambiente, uma máquina de execução orquestra as chamadas das atividades do *workflow* e estas últimas realizam a execução remota das aplicações distribuídas na grade computacional através da utilização de serviços *web*.

O modelo de proveniência do framework *Karma* define quatro dimensões de informações: **nível de execução**, definindo o nível de execução de uma atividade (e.g., nível do serviço, nível da aplicação); **localização**, definindo a localização dos elementos do *workflow* (i.e., a especificação do *workflow*); **tempo**, definindo o tempo de execução das atividades do *workflow* nos seus níveis de execução; e **fluxo de dados**, definindo a sequência de dados produzidos e consumidos no *workflow*.

O mecanismo de captura de proveniência do *framework Karma* funciona no nível de atividade. A partir de um mecanismo de *publish/subscribe* (Eugster et al. 2003), as informações são coletadas por cada atividade do *workflow* e publicadas em um repositório central através de um serviço *web* específico, chamado *Karma Provenance Service*. Entretanto, as atividades do *workflow* precisam ser adaptadas para realizarem a comunicação e a captura dos dados de proveniência. Com relação à comunicação, a tecnologia *WS-Messenger* (Huang et al. 2006) é utilizada para implementar o mecanismo *publish/subscribe*. Quanto à captura dos dados de proveniência, o *framework Karma* disponibiliza algumas ferramentas que auxiliam na criação das mensagens que serão publicadas no repositório. Por exemplo, recursos de preenchimento semiautomático das mensagens com dados de proveniência.

Com relação à visualização dos dados de proveniência, *Karma* disponibiliza um mecanismo de construção de grafos de proveniência que ilustra as dependências dos dados manipulados e dos artefatos produzidos dos *workflows*. O grafo é apresentado textualmente em formato *XML* (W3C 2008a) e não de maneira gráfica, o que facilitaria a visualização do usuário. Esse mecanismo ainda possui uma interface de consulta que possibilita filtrar esses grafos a partir de alguns parâmetros fixos. Por exemplo, é possível filtrar grafos informando o identificador de um *workflow* específico ou de um

determinado artefato produzido. Além disso, *Karma* disponibiliza uma ferramenta de monitoramento, chamada *XBaya Monitor GUI*, que permite acompanhar através de uma interface gráfica o estágio de execução do *workflow*.

Como a abordagem é uma solução específica para o cenário de grades computacionais, não é possível avaliar o seu comportamento em outros tipos de cenários. Além disso, essa ferramenta engloba outras funcionalidades que vão além de mecanismo de gerenciamento de proveniência. O *Karma* pode ser considerado um *SGWfC*, pois as etapas de composição e execução do *workflow* são feitas na própria ferramenta. Isso é prejudicial, pois as soluções fornecidas para essas funcionalidades são de nível inferior a de outros *SGWfC* existentes, desenvolvidos a mais tempo. Contudo, o trabalho não apresenta uma solução alternativa onde possa ser utilizado o *Karma* em conjunto com o *SGWfC* de preferência do usuário. Além disso, outro grande problema do *Karma* é o encargo imposto ao usuário na configuração do mecanismo de captura de proveniência nas atividades pertencentes ao *workflow*. Embora o *Karma* ainda forneça algumas funcionalidades que facilitem a geração das mensagens de publicação de dados proveniência, todo o mecanismo de comunicação do serviço *web* com o *Karma* tem que ser implementado pelo usuário.

2.4.2 PASOA

PASOA (*Provenance-Aware Service-Oriented Architecture*) (Groth et al. 2006) é um dos primeiros trabalhos a se preocupar com a definição de um mecanismo de proveniência que fosse independente de *SGWfC*. Com isso, este trabalho define uma arquitetura de proveniência agnóstica de *SGWfC*. Esta arquitetura é focada em experimentos que são executados em grades computacionais, utilizando tecnologia de serviços *web* para realizar as chamadas das aplicações na grade. Além disso, esta arquitetura trata de questões de segurança e escalabilidade de dados de proveniência e foi implementada no contexto de bioinformática (Groth et al. 2008).

PASOA possui o seu próprio modelo de proveniência, que define uma informação de proveniência como *p-assertion* (declaração-p). Um *p-assertion* é classificado de três formas: **interação**, que representa as interações entre as atividades de um *workflow*; **relação**, que representa o fluxo de dados dentro de uma atividade do *workflow*; e **estado do ator**, que representa os possíveis estados de uma atividade do

workflow. De acordo com este modelo, algumas informações são exclusivas de cada atividade do *workflow* como, por exemplo, o fluxo interno de dados e os estados de uma atividade.

Devido às particularidades do modelo de proveniência, foi estabelecido que no *PASOA* o mecanismo de captura de proveniência deve ser aplicado no nível de atividade, ou seja, cada atividade do *workflow* (i.e., a aplicação invocada pela atividade) deve ser capaz de capturar suas próprias informações de proveniência. Essas aplicações capazes de realizar tal tarefa são denominadas pelos autores de *provenance-aware applications*. Para facilitar o desenvolvimento de aplicações que sejam "*provenance-aware"*, os autores definiram uma metodologia chamada *PrIMe* (Munroe et al. 2006). No entanto, no caso de aplicações preexistentes utilizadas no experimento que não possuem as funcionalidades de proveniência previstas, esta metodologia não é útil e o usuário terá que adaptá-las manualmente para proverem tais funcionalidades.

As informações de proveniência capturadas pelas atividades são publicadas em um repositório denominado *provenance store*. Esse repositório pode ser distribuído em diversas máquinas, evitando a centralização da informação em um ponto único. Isso é vantajoso para prevenir paralisações do sistema ou para criar opções de acesso para, por exemplo, contornar problemas de *firewall*. Além disso, esse repositório possui uma interface de consulta que retorna, em formato *XML*, o grafo de proveniência de um determinado artefato manipulado no *workflow*.

Como este trabalho está no nível arquitetural, existem mais especificações de requisitos e soluções de alto nível do que soluções concretas. Embora um protótipo tenha sido construído, algumas questões ficaram em aberto. Isso pode ser observado na definição do mecanismo de captura de proveniência. Existe toda uma solução de alto nível de como realizar a captura de dados, através de definição de protocolos de comunicação e formas de armazenamento, porém existem várias questões em aberto. Por exemplo, falta um esclarecimento sobre a granularidade das informações internas coletadas pelas atividades, discutindo até que ponto coletar essas informações é importante. Outro exemplo é a falta de um ferramental que auxilie o usuário a adaptar as aplicações utilizadas no *workflow* para se tornarem aptas a capturar informação de proveniência. Existe apenas a metodologia *PrIMe* que auxilia a construção de aplicações com recursos de proveniência. No entanto, um experimento, geralmente, é composto por aplicações de terceiros que provavelmente não possuem tais recursos de

captura de proveniência definidos pela arquitetura e que, em alguns casos, nem podem ser adaptadas. Com isso, nesses casos, a captura de proveniência do experimento fica comprometida.

2.4.3 Wings/Pegasus

O *SGWfC Wings/Pegasus* é a junção de duas ferramentas (*Wings* e *Pegasus*) que fornece um ferramental para a construção e gerenciamento de experimentos científicos de larga-escala que são executados em ambientes distribuídos, mais especificamente em ambientes de grade computacionais. Um dos pontos fortes do *Wings/Pegasus* é a capacidade de trabalhar com níveis de abstração de especificação do *workflow*. Segundo os autores, esses níveis de abstração são importantes para o mecanismo de proveniência, pois são a partir deles que grande parte das informações de proveniência é capturada.

Wings/Pegasus trabalha com três níveis de abstração na composição de um *workflow*. O primeiro nível, o mais abstrato, define *templates* de *workflow* que são agnósticos de definição de dados e detalhes computacionais. O segundo nível utiliza os *templates* de *workflow* definidos no primeiro nível e gera instâncias de *workflows* que possuem informações sobre os dados a serem utilizados no *workflow* (dados de entrada e saída de cada atividade, fluxo dos dados, etc.), porém essas instâncias ainda são agnósticas de informações computacionais. Por fim, o terceiro nível faz o mapeamento das instâncias de *workflow* geradas no segundo nível em *workflows* executáveis. É nesse momento que são definidas as informações computacionais como, por exemplo, as aplicações a serem utilizadas que estão disponíveis no ambiente de execução. Os dois primeiros níveis são implementados pelo *Wings*, enquanto que o terceiro nível é implementado pelo *Pegasus*.

O mecanismo de proveniência *Wings/Pegasus* é capaz de capturar dois tipos de informação de proveniência: **proveniência de nível de aplicação**, através das representações semânticas utilizadas nas descrições do *workflow* nos três níveis de abstração (i.e., proveniência prospectiva), e **proveniência de execução**, durante a execução do *workflow* (i.e., proveniência retrospectiva). Quanto ao primeiro tipo, as informações armazenadas no *Wings* estão no formato de ontologias, usando a linguagem *OWL* (W3C 2007) e podem ser consultadas a partir da linguagem *SPARQL* (W3C 2008b). Já o segundo tipo, as informações são capturadas durante a execução do

workflow pelo *Pegasus* e armazenadas em um banco de dados relacional que permite consultas em *SQL*.

Sobre a classificação do mecanismo de captura do *Wings/Pegasus*, este pode ser classificado em dois tipos, de acordo com o tipo de proveniência coletada. No caso da coleta de proveniência prospectiva, o mecanismo de captura de proveniência pode ser classificado como de nível de *workflow*, pois o mecanismo está associado diretamente à ferramenta *Wings*, coletando as informações durante a composição do *workflow*. No caso da proveniência retrospectiva, o mecanismo de captura de proveniência do *Pegasus* trabalha no nível de atividade, pois coleta as informações de execução das atividades executadas na máquina de execução *Condor-G* (Frey et al. 2002), através de um *script* que é associado em cada tarefa submetida nessa máquina. As informações coletadas contemplam o nome do executável utilizado na tarefa, os argumentos utilizados, tempos de início e término de execução e mensagens de erro, caso haja alguma exceção na execução da tarefa.

O grande problema do *Wings/Pegasus* é que a ferramenta não consegue associar as informações de proveniência prospectiva coletadas pelo *Wings* com as informações de proveniência retrospectiva coletadas pelo *Pegasus*. Com isso, não é possível, por exemplo, mapear uma aplicação executada no ambiente distribuído com uma atividade do *workflow* abstrato, e vice-versa. O mesmo se aplica com os dados. No estágio atual, o que transparece é que as ferramentas *Wings* e *Pegasus* estão apenas sendo utilizadas em conjunto, porém é importante que estas ferramentas sejam efetivamente integradas. Tendo em vista esse problema, os autores da arquitetura *PASOA* propuseram uma solução para capturar e mapear essas transformações feitas no *Pegasus* de *workflow* abstrato para concreto utilizando a abordagem do *PASOA* (Miles et al. 2007a).

2.4.4 VisTrails

O *SGWfC VisTrails* (Callahan et al. 2006) é bastante conhecido na comunidade científica. Este sistema foi desenvolvido com o objetivo de gerenciar principalmente *workflows* que explorem recursos de visualização e exploração de dados. Nestes tipos de *workflow*s, diversas ferramentas de visualização e uma grande quantidade de dados são utilizadas, fazendo com que o cientista tenha que realizar uma série de ajustes no *workflow* para criar uma representação visual apropriada. Devido a isso, um *workflow* passa por diversas versões até chegar a uma versão definitiva. *VisTrails* ajuda

justamente neste ponto, guardando o histórico da composição do *workflow* e possibilitando que o cientista consiga voltar a um determinado ponto do passado da definição do *workflow* para poder re-executá-lo ou modificá-lo. Além disso, por ser uma ferramenta com diversos recursos, *VisTrails* é utilizado por experimentos de diversos propósitos.

O modelo de proveniência do *VisTrails* contempla apenas a parte de proveniência prospectiva, armazenando informações relacionadas à especificação do *workflow*. Por exemplo, informações sobre as atividades, conexões entre as atividades, definição de dados, parâmetros utilizados, etc. O grande diferencial do modelo é o fato de trabalhar com versionamento (Estublier 2000), permitindo armazenar o histórico de modificações da especificação do *workflow*. O modelo armazena as informações em forma de passos que indicam as ações (adicionar, remover, modificar) que o cientista tomou para transformar um *workflow* vazio até uma determinada configuração. Com relação às informações de proveniência retrospectiva, estas ainda são capturadas e armazenadas através de um mecanismo de *log*, mas as mesmas não são associadas com a proveniência prospectiva.

Todas as informações de proveniência são coletadas diretamente pelo *VisTrails*, ou seja, o mecanismo de captura de proveniência trabalha no nível de *workflow*. Essas informações podem ser salvas em arquivos ou utilizando banco de dados. Com relação à consulta, *VisTrails* disponibiliza algumas funcionalidades interessantes que auxiliam o cientista a realizar consultas sobre os dados de proveniência. Uma delas é uma ferramenta gráfica de visualização do histórico das modificações do *workflow* através de um grafo. Outra funcionalidade é denominada de *query-by-example*, que possibilita o cientista descobrir um *workflow* a partir da composição de exemplos de trechos de *workflow*s que são semelhantes ao *workflow* procurado.

VisTrails apresenta recursos interessantes no que tange à proveniência, como é o caso do mecanismo de versionamento do *workflow*. Este mecanismo, porém, possui algumas deficiências como, por exemplo, a falta de uma separação clara entre a produção do *workflow* e o espaço de trabalho do cientista (Ogasawara et al. 2009). Além disso, como já foi mencionado, o *VisTrails* tem uma preocupação maior com a proveniência prospectiva e deixa de lado a parte retrospectiva. Isso é comprovado nas funcionalidades de proveniência que o sistema disponibiliza. O mecanismo de histórico de modificações de *workflows*, por exemplo, não permite visualizar informações sobre a

execução em uma determinada configuração do *workflow*. Outro exemplo é a funcionalidade *query-by-example* que foca a consulta apenas na proveniência prospectiva.

2.4.5 myGrid/Taverna

O projeto *myGrid* (Stevens et al. 2003b) provê um conjunto de *middlewares* para dar suporte na construção, gerenciamento e colaboração de experimentos de biologia *in silico* em um ambiente de grade computacional. Uma das principais ferramentas do *myGrid* é o *SGWfC Taverna*, utilizado para compor e executar *workflows* que são escritos na linguagem *XScufl*. Esta linguagem trabalha com dois níveis de abstração na definição do *workflow*: um nível abstrato, onde o *workflow* é definido sem instanciar os serviços que serão utilizados; e um nível concreto, onde os serviços utilizados no *workflow* são instanciados.

O mecanismo de proveniência do *myGrid* utiliza fortemente recursos de ontologia para permitir o mapeamento dos dados, para que sejam mais fáceis de serem descobertos, integrados e compartilhados entre os cientistas (Zhao et al. 2004). Como é voltado para experimentos de biologia, *myGrid* possui internamente um conjunto de ontologias relacionadas a biologia. Essas informações são utilizadas para inferir informações de proveniência nos *workflow*s desenvolvidos no *Taverna*. No entanto, para que isso aconteça, todos os serviços e *workflows* utilizados no ambiente precisam ser cadastrados com anotações semânticas pelo cientista. Além disso, complementando as informações de proveniência prospectiva, outros tipos de informação são contemplados: anotações gerais descrevendo os objetivos do experimento, informações sobre aplicações utilizadas, artigos relevantes relacionados ao experimento e páginas *web* de colaboradores importantes. A ideia é que essas informações sejam associadas ao experimento para que sirvam de informação adicional para compreensão dos resultados obtidos no experimento.

Com relação aos dados de proveniência retrospectiva, estes são capturados pelo *Taverna* através do *log* da máquina de execução. Este *log* de execução contém informações sobre os serviços executados, descrevendo os tempos de execução e os parâmetros utilizados. No entanto, estas informações de proveniência não são associadas com a proveniência prospectiva. É necessário que o cientista faça anotações

semânticas sobre esses dados para que haja um mapeamento com a descrição do *workflow* e a ontologia.

Todos os dados de proveniência dos experimentos são armazenados em um repositório central, denominado *mIR* (*myGrid information repository*). *myGrid* disponibiliza algumas funcionalidades de exploração dos dados armazenados no *mIR*. Por exemplo, informações semânticas escritas em *RDF* podem ser visualizadas na forma de grafos utilizando o navegador *web* semântico *Haystack* (Zhao et al. 2004). Além disso, outra ferramenta usada é o *COHSE (Conceptual Open Hypermedia Services Environment)* para construir uma *web* semântica de proveniência. A partir dessa ferramenta, *logs* de proveniência anotados com informações semânticas são interconectados a partir de um mecanismo de inferência de ontologia e são apresentadas para o cientista em formato de páginas web conectadas por *hyperlinks* (Stevens et al. 2003b).

myGrid é um projeto interessante com contribuições importantes para a área, porém possui algumas limitações devido a ser um projeto restrito ao domínio de biologia. O mecanismo de captura de proveniência, que trabalha no nível de *workflow*, peca no mapeamento das informações de proveniência prospectiva e retrospectiva. Essa tarefa é confiada ao cientista, que deve verificar o *log* de proveniência e adicionar informações semânticas. Além de ser uma tarefa cansativa, é passível de erro, já que é uma atividade manual e depende do conhecimento do cientista.

2.4.6 Kepler

O *SGWfC Kepler* (Ludascher et al. 2006) é uma evolução da ferramenta *Ptolemy II*, que visa construir modelos de simulação a partir da composição de componentes, denominados *atores*, e executar estes modelos usando diferentes comportamentos computacionais, que são implementados em componentes denominados *diretores*. *Kepler* é uma ferramenta multidisciplinar e provê um ambiente onde cientistas podem desenvolver e executar *workflows*.

Como o *Kepler* tem como proposta ser multidisciplinar, o seu mecanismo de proveniência pode ser customizado, permitindo que a captura de proveniência seja adaptado ao cenário do experimento executado (Altintas et al. 2006). Por exemplo, é possível configurar a granularidade dos dados coletados, definir um repositório específico onde os dados serão armazenados, ou então salvar apenas resultados

específicos. Além disso, dados gerados externamente pelas atividades do *workflow* podem ser configurados para ser coletados através de uma *API* específica.

O mecanismo de captura de proveniência, denominado *Provenance Recorder* (*PR*), trabalha em conjunto com o *Kepler* (captura no nível de *workflow*), coletando informações de proveniência prospectiva e retrospectiva. As informações de proveniência prospectiva são extraídas a partir da especificação do *workflow*. PR armazena cada versão do *workflow* gerada pelo cientista, permitindo, assim, o controle da evolução do *workflow*. As informações de proveniência retrospectiva são capturadas através de eventos lançados pelo *Kepler* durante a execução do *workflow* (Bowers et al. 2006). Esses eventos indicam o momento de leitura ou escrita de dados das atividades. Além disso, existe um evento especial que controla os estados de execução de atividades do *workflow* e permite a captura da proveniência dos dados de maneira consistente (Bowers et al. 2006).

Kepler apresenta uma solução de mecanismo de proveniência interessante, porém alguns pontos da solução possuem limitações. Por exemplo, a parte de controle de estado de execução de uma atividade não depende apenas do controle do *Kepler*. É necessário que a atividade comunique com o *Kepler* lançando esse evento. No entanto, nem todas as atividades utilizadas no *workflow* podem ser adaptadas para realizar essa comunicação (e.g., serviços *web*, aplicativos proprietários, etc.). Além disso, *Kepler* apresenta mecanismos pobres de exploração de dados de proveniência. Apenas um protótipo foi desenvolvido (Bowers et al. 2006), mostrando a possibilidade de consulta de proveniência com o uso de *datalog* (Ceri et al. 1989).

2.4.7 Swift

Swift (Zhao et al. 2007) é um *SGWfC* sucessor do *Virtual Data System* (*VDS*) (Foster et al. 2002) que permite a especificação, gerenciamento e execução de experimentos de grande escala em ambientes distribuídos, principalmente em ambientes de grade computacional. *Swift* tenta facilitar a tarefa do cientista ao executar um experimento em um ambiente distribuído através de uma definição de alto nível do *workflow*. O objetivo é que a definição do modo de execução das aplicações no ambiente e a seleção de cojuntos de dados sejam feitos de modo transparente para o cientista pelo *Swift*. Entretanto, essas informações precisam ser configuradas previamente por um administrador do ambiente.

O mecanismo de captura de proveniência do *Swift* trabalha no nível de *workflow*. As informações de proveniência prospectivas são oriundas da especificação do *workflow* desenvolvida pelo cientista. Com relação à proveniência retrospectiva, as informações contempladas são: status sobre as execuções das atividades, indicando seus respectivos tempos de execução, e os dados gerados. Entretanto, informações relacionadas às atividades de configuração do ambiente para a execução do *workflow* não são capturadas. Por exemplo, submissões de tarefas para a máquina de execução ou transferências de dados necessários para a execução de uma tarefa no ambiente. Essas informações seriam importantes, por exemplo, em situações onde é necessário fazer uma depuração da execução do *workflow*.

As informações de proveniência são armazenadas em uma base de dados relacional. Logo, consultas *SQL* podem ser feitas nessa base. No entanto, os autores reconhecem que consultas *SQL* são difíceis de representar transitividade devido a estrutura das tabelas da base. Com isso, fazer uma consulta simples, por exemplo, que retorne todos os artefatos utilizados (proveniência) para a geração de um determinado artefato, é complexo. Uma solução para este problema é proposta por Dong *et al.* (1999). Além disso, durante o seu desenvolvimento, *Swift* foi experimentado com outras formas de armazenamento como, por exemplo, *XML* e *Prolog.*

Embora o *Swift* tenha sido desenvolvido com o objetivo de facilitar a tarefa do cientista na configuração e execução do experimento em ambientes distribuídos, este sistema apresenta ferramentas bastante complexas de serem manipuladas por cientistas pouco experientes em computação. *Swift* não apresenta interface gráfica e a configuração é toda por linha de comando. Para definir a estrutura de um *workflow*, por exemplo, o cientista tem que escrever um *script* similar a um código de programa. Isso vale também para as funcionalidades de exploração dos dados de proveniência, que não apresentam recursos gráficos.

2.5 Comparação entre as abordagens

Para facilitar a comparação das abordagens, os critérios gerais utilizados como guia na discussão das abordagens de proveniência na seção anterior foram desmembrados em critérios mais específicos. Estes critérios são listados abaixo:

- **Complexidade de distribuição:** indica o grau de capacidade da abordagem em gerenciar dados de proveniência de experimentos executados em

ambiente distribuídos, discutido na Seção 1.3. Uma abordagem pode ser classficada como capaz de gerenciar proveniência de experimentos executados em ambientes distribuídos de *baixa complexidade* ou de *alta complexidade*.

- **Complexidade de heterogeneidade:** indica o grau de capacidade da abordagem em gerenciar dados de proveniência de experimentos executados em ambiente heterogêneos, discutido na Seção 1.3. Uma abordagem pode ser classficada como capaz de gerenciar proveniência de experimentos executados em ambientes heterogêneos de *baixa comlexidade* ou de *alta complexidade*.

- **Adaptação de aplicações:** indica se a abordagem de proveniência tem como estratégia realizar adaptações nas aplicações utilizadas no *workflow* para fazer a coleta de proveniência. Essa estratégia é restritiva e não se adéqua a *experimentos* que utilizam aplicativos e/ou serviços legados ou proprietários que não podem ser adaptados.

- **Independência de *SGWfC*:** indica se a abordagem de proveniência funciona como mecanismo desacoplado do *SGWfC*, podendo ser usado em qualquer outro sistema de *workflow*.

- **Mapeamento entre proveniência prospectiva e retrospectiva:** indica se os dados de proveniência prospectiva e retrospectiva capturados pela abordagem são mapeados ou se são simplesmente armazenados isoladamente. A partir desse mapeamento, por exemplo, é possível associar informações da execução (proveniência retrospectiva) do *workflow* com informações da especificação do *workflow* (proveniência prospectiva), ou vice-versa.

- **Versionamento:** indica se a abordagem apresenta algum mecanismo de gerência da evolução das modificações do *workflow*.

- **Mecanismo de captura:** indica o nível de funcionamento do mecanismo de captura de proveniência da abordagem (i.e., nível de *workflow*, nível de SO ou nível de atividade), como foi discutido na Seção 2.3.1,

- **Complexidade de utilização:** indica o grau de complexidade de utilização da abordagem na perspectiva do cientista que possui pouca experiência em computação. Essa complexidade é analisada tanto nas interfaces de uso das ferramentas quanto na quantidade de tarefas manuais de configuração da abordagem delegadas ao cientista.
- **Visualização de dados:** indica o grau de suporte da abordagem na disponibilização de ferramentas de visualização de dados de proveniência.
- **Modelo de dados:** indica qual modelo de dados é utilizado pela abordagem para armazenar os dados de proveniência.

Tabela 2.1 - Quadro comparativo entre as abordagens descritas neste capítulo

Critérios	*Karma*	*PASOA*	*Wings/ Pegasus*	*VisTrails*	*myGrid/ Taverna*	*Kepler*	*Swift*
Complexidade de distribuição	*Baixa*	*Baixa*	*Baixa*	*Baixa*	*Baixa*	*Baixa*	*Baixa*
Complexidade de heterogeneidade	*Baixa*	*Baixa*	*Baixa*	*Baixa*	*Baixa*	*Baixa*	*Baixa*
Suporte de uso de aplicações proprietárias	*Não*	*Não*	*Sim*	*Sim*	*Sim*	*Sim*	*Sim*
Independência de *SGWfC*	*Não*	*Sim*	*Não*	*Não*	*Não*	*Não*	*Não*
Mapeamento entre proveniência prospectiva e retrospectiva	*Sim*	*Sim*	*Não*	*Não*	*Sim*	*Sim*	*Sim*
Versionamento	*Não*	*Não*	*Não*	*Sim*	*Não*	*Sim*	*Não*
Mecanismo de captura	*Atv*	*Atv*	*Wf e Atv*	*Wf*	*Wf*	*Wf*	*Wf*
Complexidade de utilização	*Alta*	*Alta*	*Média*	*Baixa*	*Média*	*Baixa*	*Alta*
Visualização de dados	*Médio*	*Nulo*	*Nulo*	*Alto*	*Alto*	*Nulo*	*Nulo*
Modelo de dados	*Relacio-nal*	*XML*	*RDF*	*Relacio-nal*	*RDF*	*XML*	*Relacio-nal*

Vale destacar que nem todas as abordagens estudadas estão inseridas exatamente no mesmo contexto e, com isso, pode justificar algumas de suas deficiências. Portanto, esses critérios devem ser tratados como uma guia para a definição de uma abordagem de gerenciamento de proveniência que possua um conjunto mínimo de características,

disponíveis nas abordagens atuais, e que novas contribuições sejam acrescentadas. A Tabela 2.1 classifica as abordagens de acordo com os critérios definidos.

2.6 Considerações finais

Analisando o comparativo das abordagens apresentado, algumas considerações podem ser delineadas. Vale ressaltar que estas considerações não são verdades absolutas e devem ser consideradas como conjecturas para guiar o desenvolvimento de uma abordagem de proveniência mais abrangente:

- Nenhuma abordagem apresenta uma solução para gerenciamento de proveniência em cenários onde o experimento é segmentado em mais de um *workflow* executados em *SGWfC* distintos.
- A maioria das abordagens é dependente de *SGWfC*, com exceção do abordagem *PASOA*.
- As abordagens que usam mecanismo de captura de proveniência no nível de atividade usam estratégias de adaptação das atividades de *workflow*, que implicam em alterações nas funcionalidades das aplicações utilizadas. No entanto, essas estratégias não se aplicam a casos de experimentos que utilizam aplicações de terceiros, programas legados ou serviços remotos, onde o cientista é impossibilitado de fazer tais alterações.
- A maioria das abordagens não apresenta mecanismos de visualização de dados de proveniência. Além disso, as abordagens utilizam linguagens de consultas complexas para usuário com pouca experiência em computação.
- As abordagens que apresentam mais recursos para a captura de proveniência em ambiente distribuído não possuem mecanismos automatizados de captura e dependem do cientista na realização de algumas tarefas manuais de configuração do sistema. Além disso, em algumas destas abordagens, a coleta de proveniência tem que ser feita manualmente pelo cientista.

Além das abordagens descritas neste capítulo, existem outras abordagens na literatura como, por exemplo, *Zoom* (Cohen-Boulakia et al. 2008), *Redux* (Barga e Digiampietri 2008) e *Matriohska* (Cruz et al. 2008), dentre outras. Comparações

envolvendo estas e outras abordagens de proveniências podem ser encontradas em *surveys* na literatura (Simmhan et al. 2005) (Bose e Frew 2005) (Da Cruz et al. 2009).

A partir dessa análise, é possível perceber que nenhuma abordagem permite avaliar a questão de pesquisa proposta neste trabalho, por não atuar em cenários de ambiente de distribuição mais complexos, ou por não procurar dirimir os esforços do cientista tanto na etapa de configuração do mecanismo de proveniência quanto na etapa de utilização. Com isso, existe a necessidade do desenvolvimento de uma nova abordagem que trate essas características. Essa abordagem será apresentada no Capítulo 3.

Capítulo 3. ProvManager

3.1 Introdução

Como discutido no Capítulo 2, existe a necessidade de uma abordagem de gerenciamento de proveniência que consiga tratar experimentos que são executados em ambientes distribuídos mais complexos. Nestes cenários mais complexos, vários *SGWfC* ou máquinas de execução são utilizados, dificultando uma análise integrada dos dados de proveniência durante e após a execução do experimento. Este capítulo apresenta uma nova abordagem, intitulada *ProvManager*, que tenta mitigar esse problema através da definição de um mecanismo de proveniência independente de *SGWfC* ou máquinas de execução.

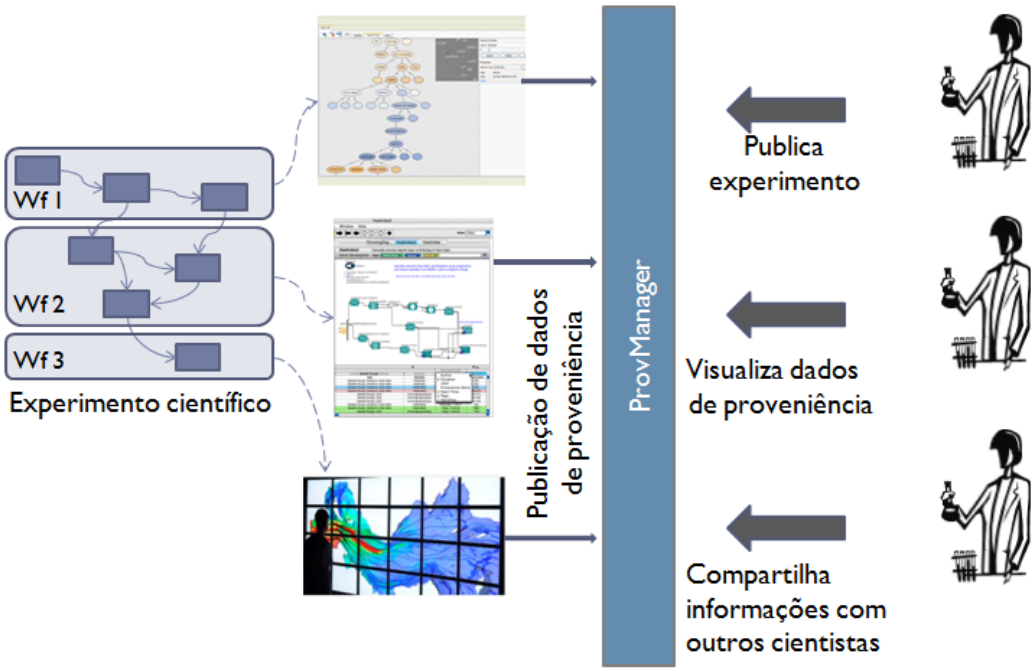

Figura 3.1 - Visão geral de funcionamento da abordagem

A Figura 3.1 apresenta uma visão geral de funcionamento da abordagem proposta. *ProvManager* fornece um registro integrado dos dados de proveniência do experimento, gerados de forma distribuída em um ambiente. Isto é possível, pois os dados de proveniência são coletados dos *SGWfC* utilizados na execução dos *workflows*, permitindo que o cientista tenha uma visão integrada dos dados de proveniência do

experimento sem ter que visitar individualmente cada um desses sistemas. Além disso, essa estratégia é interessante sob a perspectiva de usabilidade, pois o cientista passa a usar uma interface padronizada de acesso e um modelo único de dados de proveniência. A partir dessa interface, cientistas publicam seus experimentos, visualizam os dados de proveniência capturados e, além disso, podem compartilhar essas informações com outros cientistas.

Nas próximas seções deste capítulo, a abordagem proposta é detalhada, descrevendo suas principais características, sua arquitetura e seus principais componentes. A Seção 3.2 discute as possíveis estratégias de captura de dados de proveniência e apresenta a estratégia adotada na abordagem *ProvManager*. A Seção 3.3 fornece uma visão geral de funcionamento da abordagem *ProvManager* e apresenta as suas principais características. A Seção 3.4 descreve a arquitetura da abordagem proposta, destacando os principais componentes envolvidos. A Seção 3.5 detalha o modelo de dados de proveniência definido por esta abordagem. A Seção 3.6 apresenta a máquina de processos *Charon* que foi estendida e utilizada no *ProvManager*. A Seção 3.7 discute sobre os configuradores de proveniência. A Seção 3.8 define e discute sobre os agentes inteligentes no contexto de proveniência.

3.2 Estratégias de captura de proveniência

Ainda analisando a Figura 3.1, o processo de captura dos dados de proveniência dos *SGWfC* e publicação no *ProvManager* pode ser realizado de diversas maneiras. Uma estratégia consiste em delegar a coleta dos dados de proveniência para os próprios mecanismos de captura dos *SGWfC* utilizados no experimento. Os dados coletados por estes mecanismos seriam exportados e publicados no *ProvManager*. Essa estratégia é viável, uma vez que atualmente grande parte dos *SGWfC* passou a disponibilizar funcionalidades de exportação de dados de proveniência utilizando o formato OPM, como discutido na Seção 2.3.2. Utilizando a classificação de mecanismos de captura da Seção 2.3.1, esta estratégia seria equivalente à captura de proveniência no nível de *workflow*.

Esta estratégia é interessante devido à facilidade de implementação, porém apresenta algumas deficiências que vão de encontro aos objetivos da abordagem *ProvManager*. A primeira delas está relacionada à independência com o *SGWfC*.

Embora essa estratégia viabilize a independência dos *SGWfC* durante a consulta, uma vez que os dados de proveniência podem ser acessados diretamente no *ProvManager*, ainda existe a dependência dos *SGWfC* no momento da captura dos dados. As informações de proveniências providas pelo *ProvManager* seriam somente aquelas fornecidas pelos *SGWfC*. Com isso, se os sistemas utilizados em um experimento não possuírem bons mecanismos de proveniência, a abordagem *ProvManager* estaria comprometida. Além disso, esta estratégia faz com que o nível de detalhamento dos dados de proveniência seja heterogêneo ao longo do experimento, uma vez que os *SGWfC* possuem mecanismos de proveniência distintos. Isso resultaria em determinadas regiões do experimento com mais informações de proveniência do que outras. Por exemplo, em uma determinada região do experimento seria possível analisar os dados internos utilizados pelas atividades, enquanto que em outra região seria possível analisar apenas os dados de entrada e saída das atividades. Ou então, em situações mais graves, onde *workflows* estão conectados em série, uma análise sobre a sequência dos passos realizados para a geração de um determinado dado ficaria comprometida, se apenas um destes *workflows* estiver sendo executado em um *SGWfC* que não captura proveniência de maneira adequada. Outra característica negativa é que os mecanismos de exportação dos dados de proveniência dos *SGWfC* ainda possuem deficiência, apresentando falhas na conversão para a linguagem *OPM*.

Outras estratégias para viabilizar a independência desejada da abordagem dos sistemas de *workflow* são utilizar mecanismos de captura de proveniência que trabalhem no nível de *SO* ou no nível de atividade, como foi apresentado na Seção 2.3.1. No caso de utilizar um mecanismo no nível de *SO*, as informações de proveniência seriam extraídas utilizando as próprias informações geradas pelo sistema operacional. Por exemplo, informações sobre o início e término de execução de processos fornecido pelo registro de chamadas do sistema (*system call tracer*), ou observando o sistema de arquivos do sistema operacional para descobrir novos arquivos gerados. No entanto, estas informações possuem granularidade fina e precisam ser processadas para extrair informações de proveniência úteis. Além disso, como é uma estratégia que funciona completamente desvinculada ao *SGWfC*, nem todas as informações de proveniência são coletadas, capturando apenas informações de proveniência retrospectiva. Informações de proveniência prospectiva armazenadas nos *SGWfC* (e.g., especificação do *workflow*, anotações do cientista feitas no sistema, etc.) não são capturadas.

Por outro lado, a estratégia de utilizar um mecanismo de captura que trabalhe no nível de atividade funciona como um meio termo das duas estratégias anteriores. Trabalhando neste nível, a captura de proveniência é delegada para cada atividade do *workflow*, garantindo, assim, uma captura de proveniência mais precisa, equivalente a um mecanismo que trabalha no nível de *workflow*. Além disso, o mecanismo de captura torna-se independente de *SGWfC*, da mesma maneira que um mecanismo que trabalha no nível de *SO*. No entanto, esta estratégia apresenta também algumas deficiências. A principal delas é a necessidade de adaptação das atividades do *workflow* (mais especificamente, as aplicações utilizada no *workflow*) para viabilizar a captura dos dados de proveniência. No entanto, nem todas as aplicações são possíveis de serem adaptadas (e.g., serviços *web*, aplicações legadas, proprietárias, etc.). Com isso, a captura de proveniência fica comprometida. Além disso, na maioria das abordagens que trabalham com mecanismos neste nível, essa adaptação é delegada ao cientista (Simmhan et al. 2006) (Munroe et al. 2006) (Lin et al. 2008) que, em determinados cenários, não possui muita experiência em computação. Por fim, outro ponto negativo é que a parte de proveniência prospectiva não é capturada com este tipo de mecanismo.

Embora apresente deficiências como as outras, a última estratégia apresentada é a mais adequada de ser aplicada na abordagem *ProvManager*. Um dos motivos é a sua característica de ser independente de *SGWfC* e, ao mesmo tempo, possibilitar uma captura de proveniência precisa. Com isso, esta estratégia de captura de proveniência é adotada pela abordagem *ProvManager*. No entanto, *ProvManager* apresenta algumas soluções adicionais, discutidas no decorrer deste capítulo, que tentam amenizar os problemas encontrados nesta estratégia. A partir dessas soluções, a abordagem torna-se capaz de: (i) capturar informações de proveniência do tipo prospectiva, além da retrospectiva; (ii) possibilitar a adaptação de qualquer tipo de aplicação para suportar o mecanismo de captura; e (iii) viabilizar que essa adaptação seja feita de forma automática sem sobrecarregar o cientista.

3.3 Visão geral de funcionamento

A Figura 3.2 apresenta uma visão geral do funcionamento do *ProvManager*, utilizando um mecanismo de captura de proveniência que trabalha no nível de atividade com a adição de novas características que complementam as lacunas deixadas na

utilização desta estratégia. Depois de realizada a composição do experimento, o primeiro passo que o cientista precisa realizar é publicar o experimento no *ProvManager*, passando as especificações dos *workflows* definidas em cada *SGWfC*. A partir dessas especificações, *ProvManager* é capaz de extrair as informações de proveniência prospectiva do experimento que não são possíveis de serem coletadas pelos mecanismos de captura das atividades. Neste momento, também é feita a adaptação automática das especificações do *workflow* para permitirem que as atividades suportem o mecanismo de captura de proveniência retrospectiva. Esta adaptação é feita no nível de especificação e não diretamente no nível da aplicação, permitindo, assim, que qualquer aplicação seja adaptada, como será discutido em detalhes na Seção 3.7. Estas especificações adaptadas são retornadas ao cientista para que sejam carregadas de volta nos *SGWfC*. Após essa etapa, quando o cientista inicializa a execução do experimento, os dados de proveniência retrospectiva são transferidos de cada atividade do *workflow* para o *ProvManager* através de uma *API* de serviços de publicação de proveniência. Por fim, o cientista visualiza e analisa os dados de proveniência diretamente na interface do *ProvManager*. É interessante que a interface de análise seja do tipo *web*, para facilitar o acesso do cientista que pode estar localizado em qualquer máquina em um ambiente distribuído.

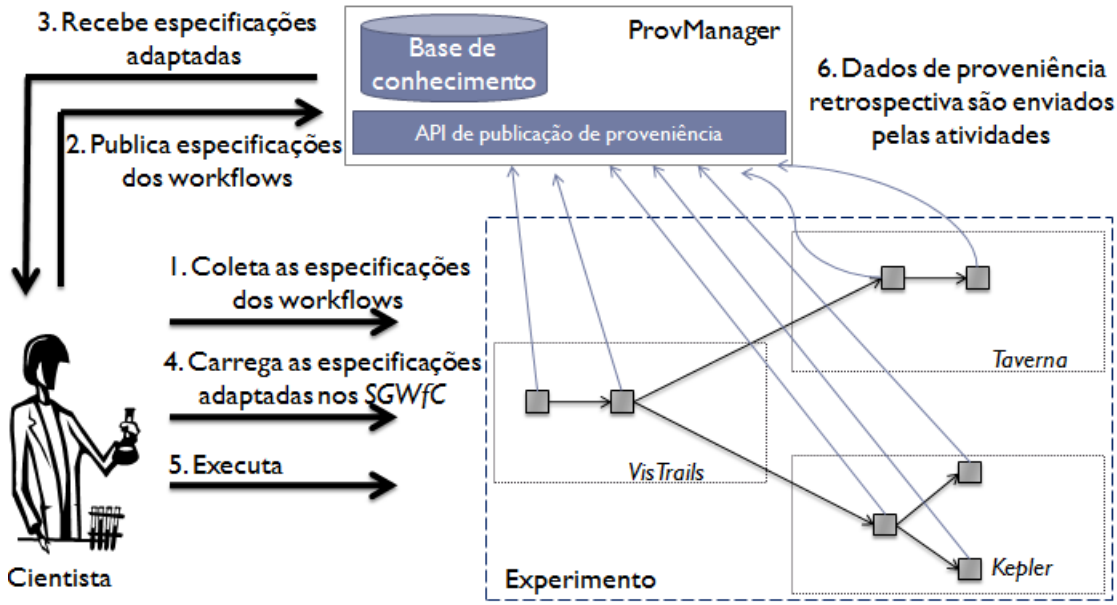

Figura 3.2 - Funcionamento do mecanismo de captura da abordagem

Para facilitar o processo de captura dos dados de proveniência, a abordagem *ProvManager* utiliza uma base de conhecimento (Gonzalez e Dankel 1993), que possui

um mecanismo de inferência para armazenar e derivar os dados de proveniência. Com esse mecanismo, novas informações de proveniência podem ser derivadas a partir das informações capturadas pelas atividades do *workflow*. Por exemplo, uma simples notificação de término de execução enviada por uma atividade do *workflow* funciona como gatilho para derivar que atividades dependentes a esta no *workflow* iniciaram a sua execução. Esse tipo de derivação de informação de proveniência através de uso de mecanismo de inferência é defendido pelo modelo *OPM*, como foi apresentado na Seção 2.3.2.

Essa base de conhecimento adota *Prolog* como linguagem de representação. *Prolog* é interessante de ser utilizada por ser uma linguagem que utiliza fortemente recursos de inferência, facilitando, assim, o processo de derivação dos dados de proveniência. A Figura 3.3 apresenta um exemplo simples de alguns tipos de inferência que podem ser feitos a partir de dados sobre a execução de um *workflow*. Utilizando a terminologia do OPM, o processo P1 utilizou o artefato A1 e gerou o artefato A2. A partir de inferência, é possível concluir que o artefato A2 foi gerado a partir do artefato A1. A seguir, o processo P2 utilizou o artefato A2 gerado pelo processo P1. Esta informação também pode indicar que o processo P2 foi inicializado pelo processo P1, pois este último precisou ser executado primeiro para gerar o artefato necessário (A2) para o processo P2 iniciar sua execução.

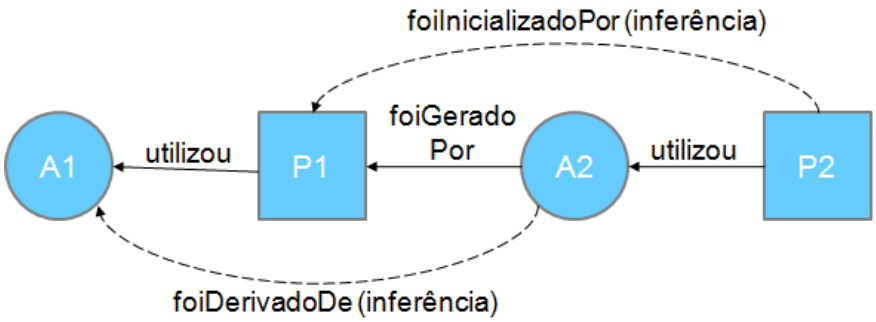

Figura 3.3 - Exemplo de inferência em dados de proveniência (adaptação de (2007))

Além disso, a partir da aplicação de uma base de conhecimento, é possível utilizar a abordagem de agentes inteligentes (Jennings 2000) no contexto de proveniência, na construção de agentes que auxiliem nas diversas funcionalidades providas pela análise de proveniência. Por exemplo, controle de qualidades dos dados gerados no experimento, auditoria, depuração, monitoramento da execução do experimento, etc. Estes agentes, além de facilitar o trabalho de análise do cientista

automatizando algumas tarefas, diminuem a complexidade de se trabalhar diretamente com a linguagem *Prolog*.

Em resumo, a abordagem *ProvManager* pode ser caracterizada pelas seguintes características principais:

- Independência de *SGWfC;*
- Mecanismo de captura de proveniência no nível de atividade (com algumas adaptações);
- Configuração do mecanismo de captura de proveniência feita de maneira semiautomática;
- Utilização de uma base de conhecimento com recursos de inferência para armazenar e derivar os dados de proveniência;
- Utilização de agentes inteligentes para manipulação dos dados de proveniência; e
- Interface *web* de acesso.

3.4 Arquitetura

A Figura 3.4 apresenta a arquitetura do *ProvManager* representada de acordo com o estilo camadas (Qin et al. 2008). Esta arquitetura é definida pelas seguintes camadas:

- **Core**: Esta camada contém os componentes que são a base de funcionamento do *ProvManager*. Por exemplo, o *Modelo de proveniência*, que define todos os dados de proveniência manipulados na abordagem. Outros componentes são o repositório de dados de proveniência, representado pela *Base de conhecimento*, e os *agentes inteligentes*, que interagem com os dados armazenados nessa base. Para representar estes dois últimos componentes, a abordagem *ProvManager* utiliza uma máquina de processos denominada *Charon* (Murta 2002). A máquina de processos *Charon* usa técnicas de agentes inteligentes para interagir e manipular os dados de uma base de conhecimento representados em *Prolog*.

- **Configuração:** Esta camada representa os componentes da arquitetura que são utilizados para viabilizar a execução do *ProvManager*. O componente *Mecanismo de captura* é responsável por coletar os dados de proveniência do experimento para serem publicados na *Base de conhecimento*. Como foi discutido anteriormente, o *Mecanismo de captura* trabalha no nível de atividade, porém apresenta algumas características adicionais. Uma delas é a inserção de um mecanismo que realiza a configuração das atividades do *workflow* para viabilizar a captura de proveniência. Os componentes *Adaptadores de proveniência* são utilizados para facilitar esse processo de configuração do mecanismo de proveniência, automatizando a tarefa de adaptação das atividades do *workflow*.

- ***API* de serviços:** Esta camada representa os componentes que viabilizam a publicação dos dados de proveniência no *ProvManager*. O componente *Publicador de proveniência prospectiva* representa a *API* de serviços relacionada à publicação de dados de proveniência prospectiva. De maneira análoga, o componente *Publicador de proveniência retrospectiva* representa os serviços relacionados à publicação de proveniência retrospectiva. Estes componentes são necessários devido às características do mecanismo de captura de proveniência que trabalha no nível de atividade. Com isso, os dados de proveniência são publicados remotamente a partir de diversas máquinas de acordo com a distribuição das atividades do *workflow*. Os serviços da API são listados no Anexo A (API de Serviços de Publicação de Proveniência Prospectiva) e Anexo B (API de Serviços de Publicação de Proveniência Retrospectiva).

- **Interface web:** Essa camada representa as aplicações da abordagem *ProvManager* disponíveis para o cientista através de uma interface *web*. Em suma, essa interface disponibiliza componentes de *Edição de experimentos*, permitindo que cientistas cadastrem e editem seus experimentos; *Monitoramento*, possibilitando que cientistas acompanhem a execução dos seus experimentos; *Consulta*, viabilizando que cientistas façam consultas acerca dos dados de proveniência capturados do experimento; e *Agentes de proveniência*, agentes

inteligentes que podem ser construídos para ajudar o cientista em determinada análise de proveniência. Além destes componentes, outros podem ser acrescentados na interface de acordo com a necessidade.

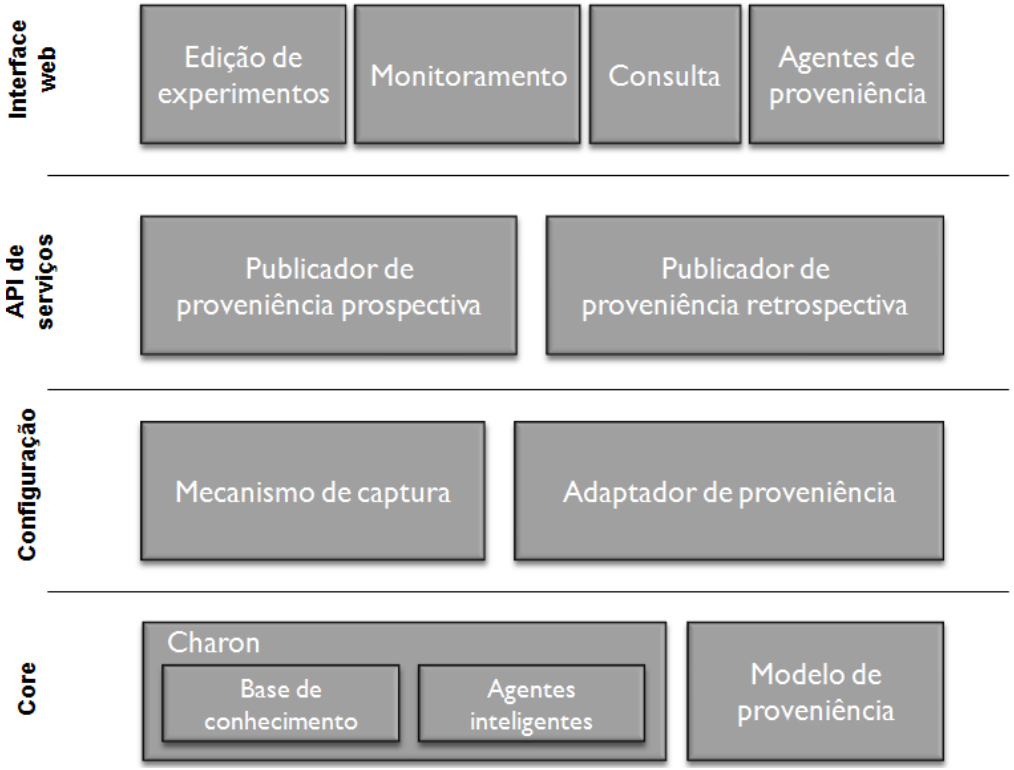

Figura 3.4 – Arquitetura em camadas do *ProvManager*

Alguns destes componentes apresentados na arquitetura são descritos com mais detalhes nas próximas seções.

3.5 *Modelo de proveniência*

O modelo de proveniência define as informações de proveniência da abordagem. Este modelo lida tanto com proveniência prospectiva quanto retrospectiva. A proveniência prospectiva é fornecida pelo componente configurador de proveniência, durante a adaptação do *workflow* (mais detalhes na Seção 3.7), enquanto a proveniência retrospectiva é fornecida pelo mecanismo de captura que envia informações sobre o *workflow* em tempo de execução (*runtime*).

O modelo de proveniência do *ProvManager* é baseado no *OPM*, porém possui algumas características adicionais que complementam as deficiências do *OPM*. Uma delas é o suporte às informações de proveniência prospectiva, além da proveniência

retrospectiva suportada pelo *OPM*. Além disso, o modelo de proveniência do *ProvManager* permite a construção de *workflows* do tipo grafo direcionado cíclico (Ghiya e Hendren 1996). Isso significa que este modelo fornece estruturas de controle, tais como ponto de decisão, ponto de sincronismo, além de outras estruturas derivadas, como, por exemplo, repetição (*for*, *loop*, etc.*)*.

Outra característica importante é que o modelo de proveniência do *ProvManager* foi definido para suportar recursividade, possibilitando a construção de atividades de *workflow* que são compostas por elas mesmas. Isto é possível, pois o modelo permite a reutilização das atividades do *workflow* através do conceito de instanciação de atividades, viabilizando que uma atividade seja definida apenas uma vez e seja instanciada diversas vezes em um *workflow*. Em resumo, essas características são importantes para tornar o modelo de proveniência do *ProvManager* o mais genérico possível, a fim de ser compatível com a maioria dos tipos de dados de proveniência suportados pelos *SGWfC*.

A Figura 3.5 apresenta o modelo de proveniência do *ProvManager*, representado em um diagrama de classes *UML* (OMG 2010). As entidades principais desse modelo são:

- Experimento: Entidade que representa conceitualmente um experimento. Um experimento é constituído por um ou mais *workflows*, viabilizando a representação de experimentos que são executados em ambientes distribuídos;
- Versão: Entidade que representa um estado de configuração do experimento em determinado instante de tempo. Uma versão contém os *workflows* que foram definidos em um experimento em determinado instante de tempo;
- *Workflow*: Entidade que representa um fragmento da definição de execução do experimento, ou seja, um conjunto de atividades que seguem uma determinada ordem de execução. Um *workflow* é instanciado e executado em um *SGWfC* específico;
- *SGWfC*: Entidade que representa os sistemas gerenciadores de *workflow* científico utilizados no experimento para executar os *workflows*;

- Atividade simples: Entidade que representa um elemento de processamento atômico do *workflow*. Uma atividade pode estipular um conjunto de artefatos que devem ser fornecidos como insumos para viabilizar a sua execução. Uma atividade pode gerar um ou mais artefatos (produtos);
- Atividade composta: Entidade que representa uma atividade do *workflow* que agrega uma ou mais atividades, simples ou compostas, formando um *subworkflow*;
- Instância de atividade (simples ou composta): Entidade que representa a atividade (simples ou composta) concretizada em um *workflow*;
- Artefato: Entidade que representa os dados consumidos (insumos) e gerados (produtos) por uma atividade;
- Parâmetro: Entidade que representa um valor configurável de uma atividade que influencia no seu comportamento de execução;
- Ponto de sincronismo: Entidade que permite sincronizar um conjunto de fluxos de processos. O ponto de sincronismo pode ser utilizado tanto para definir que diversos fluxos de saída devem ser ativados em paralelo, quanto para definir que o próximo elemento só será executado após a ativação de todos os fluxos de chegada;
- Ponto de decisão: Entidade que permite a escolha, dentre um conjunto de opções, de quais fluxos serão ativados para prosseguir a execução do *workflow* do processo composto;
- Fluxo: Entidade que representa a ligação entre dois elementos do *workflow*, definindo a ordem de precedência necessária para a sua execução;
- Execução: Entidade que representa uma execução de um experimento; e
- Execução Elemento *Workflow*: Entidade que representa as informações de execução de um elemento de *workflow* em uma determinada execução do experimento.

As demais entidades (*Elemento de workflow* e *Atividade*) são utilizadas com o objetivo de facilitar a organização do modelo.

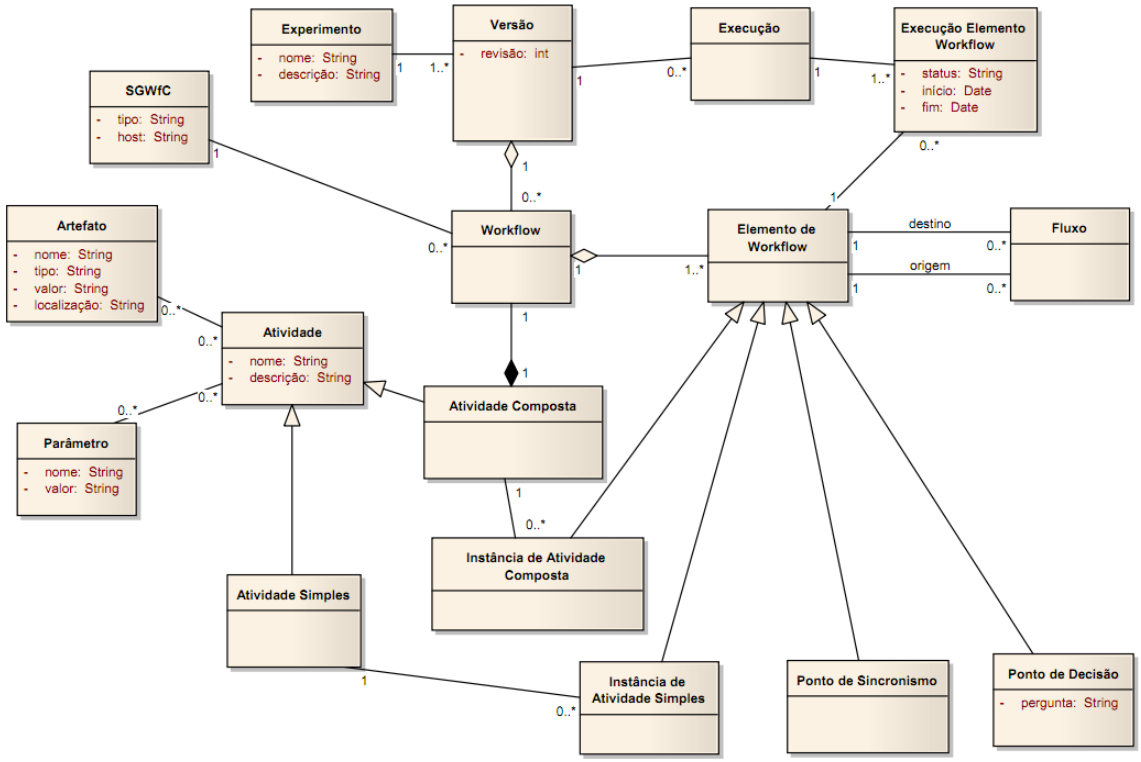

Figura 3.5 - Modelo de proveniência do *ProvManager*

3.6 *Charon*

A ferramenta *Charon* é uma máquina de processos que auxilia o desenvolvedor na execução do processo de desenvolvimento de *software* (Murta 2002). O conceito de processo utilizado na *Charon* possui a mesma conotação do conceito de *workflow* em experimentação científica. Um processo define a ordem de execução de um conjunto de atividades, explicitando os artefatos consumidos e produzidos por estas atividades. A partir da *Charon*, processos são modelados, simulados e executados. A execução do processo é feita de maneira interativa com o usuário. Durante a execução, a *Charon* informa o estágio de execução do processo apresentando as atividades executadas e as atividades pendentes. As atividades pendentes ganham status de executada quando seus artefatos são fornecidos pelo usuário. Por exemplo, uma atividade do tipo "desenvolver plano de teste" termina sua execução quando a *Charon* recebe do usuário o artefato "plano de teste".

Internamente, a *Charon* é caracterizada por trabalhar com uma base de conhecimento e utilizar técnicas de agentes inteligentes para interagir com essa base. A

Figura 3.6 apresenta uma visão geral de funcionamento da ferramenta. A *Charon* foi concebida para permitir que seja utilizado um ambiente de desenvolvimento para realizar a modelagem gráfica do processo, de acordo com a parte (a) da Figura 3.6. Logo após, na parte (b) da Figura 3.6, esse modelo gráfico é convertido para a base de conhecimento da *Charon*. Essa base de conhecimento utiliza Prolog como linguagem de execução. Por fim, para que o processo instanciado na base de conhecimento seja executado, agentes inteligentes devem se conectar com a base alterando ou inserindo novas informações, de acordo com a parte (c) da Figura 3.6.

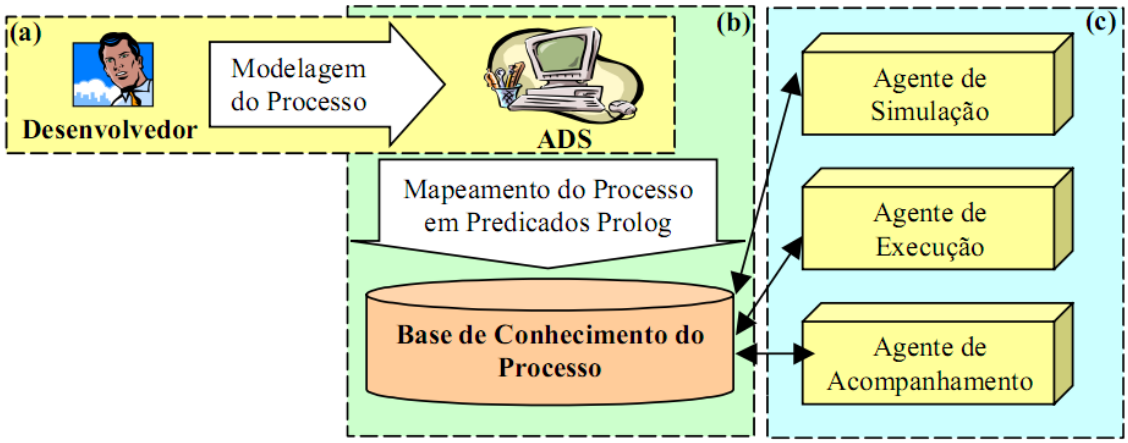

Figura 3.6 - Visão geral de funcionamento da Charon (retirado de Murta (2002))

Inicialmente, a *Charon* foi desenvolvida para ser utilizada em conjunto com o ambiente de desenvolvimento *Odyssey* (Werner et al. 1999). Posteriormente, uma versão contendo apenas o seu *kernel* (base de conhecimento e agentes inteligentes) foi desenvolvida para facilitar a sua utilização em outras aplicações (Lopes et al. 2006). No entanto, para ser compatível com as características e requisitos do *ProvManager*, a *Charon* teve que sofrer algumas adaptações para modificar funcionalidades antigas e inserir novas. As principais adaptações foram:

- **Inserção de mecanismo de persistência:** permite que os dados inseridos na base de conhecimento sejam persistidos em um banco de dados relacional. A única persistência que a *Charon* possuía era um mecanismo que guardava todos os dados da base de conhecimento em arquivo. No entanto, essa persistência não era feita de forma automática a cada inserção ou alteração de dados na base, e sim de maneira manual através da solicitação do usuário. Uma vantagem de persistir os dados de proveniência em um banco de dados relacional é ter uma opção alternativa de consulta através da linguagem *SQL*.

- **Inserção de versionamento:** permite que a base de conhecimento armazene e relacione diversas versões de um mesmo *workflow*. A partir dessa funcionalidade, é possível construir um mecanismo de visualização de histórico de modificações de um *workflow*.
- **Adaptação de fatos e regras da base de conhecimento:** Os fatos e regras da base de conhecimento foram alterados para serem compatíveis com os conceitos básicos de experimentação científica e o modelo de proveniência da abordagem *ProvManager*.
- **Criação de API de carregamento de experimento:** interface com conjunto de operações que permite que o *ProvManager* carregue as informações de um experimento na base de conhecimento da *Charon*.

3.7 *Adaptador de proveniência*

Como foi discutido anteriormente, o mecanismo de captura do *ProvManager* situa-se no nível de atividade. Isso implica que cada atividade do *workflow* deve prover o seu próprio mecanismo de captura. No entanto, *workflows* comuns que são utilizados por cientistas em experimentos não possuem esse tipo de mecanismo incorporado. Logo, é necessário que algumas adaptações no *workflow* sejam feitas para que as atividades do *workflow* consigam capturar as informações de proveniência e enviá-las para o *ProvManager*. Por questões de praticidade, a abordagem *ProvManager* poderia delegar esta tarefa de adaptação das atividades do *workflow* para o cientista. Entretanto, para reduzir o esforço do cientista na utilização da abordagem, *ProvManager* fornece apoio para realizar essa adaptação de maneira semiautomática por meio do componente *Adaptador de proveniência*. Este componente tem como funcionalidade principal realizar a configuração automática do mecanismo de captura de proveniência do *ProvManager* em um experimento. Durante esse processo de configuração, o componente *Adaptador de proveniência* realiza também a captura de proveniência prospectiva.

O processo de configuração do mecanismo de captura de proveniência em um experimento consiste em quatro etapas:

1. O cientista fornece a especificação do *workflow* do experimento para o *adaptador de proveniência*;

2. O *adaptador de proveniência* captura as informações de proveniência prospectiva contidas na especificação do *workflow*;
3. O *adaptador de proveniência* instrumenta o *workflow* com o mecanismo de captura de proveniência;
4. Uma nova especificação do *workflow* adaptada é devolvida para o cientista; e
5. O cientista recarrega a especificação do *workflow* adaptada no *SGWfC.*

Analisando esse processo, algumas questões merecem ser tratadas com mais atenção. A primeira delas é sobre a especificação do *workflow*, que pode ser escrita de maneira diferente de acordo com o *SGWfC* escolhido. Ou seja, cada *SGWfC* possui características próprias como, por exemplo, linguagem de especificação de *workflow*, suporte a determinados tipos de atividades, modelos de execução distintos, etc. Por esses motivos, o *adaptador de proveniência* deve estar preparado para configurar *workflows* modelados em diferentes *SGWfC*. Outra questão está relacionada com a forma de adaptação das atividades do *workflow* para suportarem o mecanismo de captura de proveniência. Mais uma vez, a solução é dependente do *SGWfC*. Alguns sistemas escrevem o código da atividade diretamente na especificação do *workflow*, outros trabalham apenas com serviços *web* ou executáveis. Além disso, como discutido anteriormente, algumas aplicações utilizadas no experimento não são passíveis de serem adaptadas, como, por exemplo, serviços web, aplicações proprietárias que não disponibilizam código fonte, aplicações legadas, etc. Portanto, a adaptação das atividades do *workflow* é uma tarefa complexa, pois varia de acordo com o *SGWfC*, além de possuir limitações em função do tipo de aplicações utilizadas no experimento. Com isso, é necessário estabelecer uma solução mais independente das propriedades do *SGWfC* e das aplicações utilizadas.

Para resolver o primeiro problema, a arquitetura do *adaptador de proveniência* foi projetada para ser extensível e permitir as variações dos diversos *SGWfC*. O configurador de proveniência suporta a conexão de adaptadores específicos *de SGWfC* (ou seja, *plugins* específicos de *SGWfC*), como é ilustrado na Figura 3.7. Esses adaptadores são responsáveis por configurar *workflows* desenvolvidos em *SGWfC* específicos. A partir disso, para que o *adaptador de proveniência* cubra uma vasta variedade de *SGWfC*, é necessário que sejam implementados diversos tipos de adaptadores para cada um desses sistemas. A Figura 3.7 ilustra a arquitetura do

adaptador de proveniência. Nesta arquitetura, por exemplo, temos os adaptadores específicos para os *SGWfC VisTrails*, *Kepler* e *Taverna*.

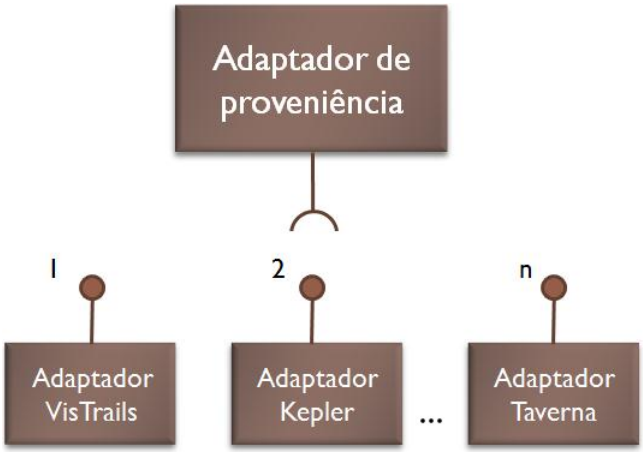

Figura 3.7 - Arquitetura do adaptador de proveniência

Com relação ao problema da adaptação das atividades, uma possível solução é subir o nível de adaptação da atividade do nível da aplicação para o nível da especificação do *workflow*. Em outras palavras, ao invés de realizar a adaptação diretamente no código da aplicação, uma alternativa é modificar a estrutura do *workflow* para que o comportamento da atividade seja alterado. Isto pode ser feito a partir da construção de *wrappers* na especificação do *workflow* que encapsulem as atividades. Cada *wrapper* encapsula uma atividade do *workflow* e permite a inserção do mecanismo de captura de proveniência, responsável pela coleta e publicação das informações de proveniência da atividade no *ProvManager*. Essa publicação é feita utilizando os componentes da *API* de serviços *web* do *ProvManager*, destacados na Seção 3.4. A Figura 3.8 apresenta o conceito da estrutura *wrapper* de uma atividade adaptada, que é composta pela atividade original e pelo mecanismo de captura de proveniência. É possível observar que os dados de entrada da atividade adaptada e os dados de saída da atividade original são transferidos para o mecanismo de captura, que os repassa para o *ProvManager* utilizando a *API* de serviços *web*.

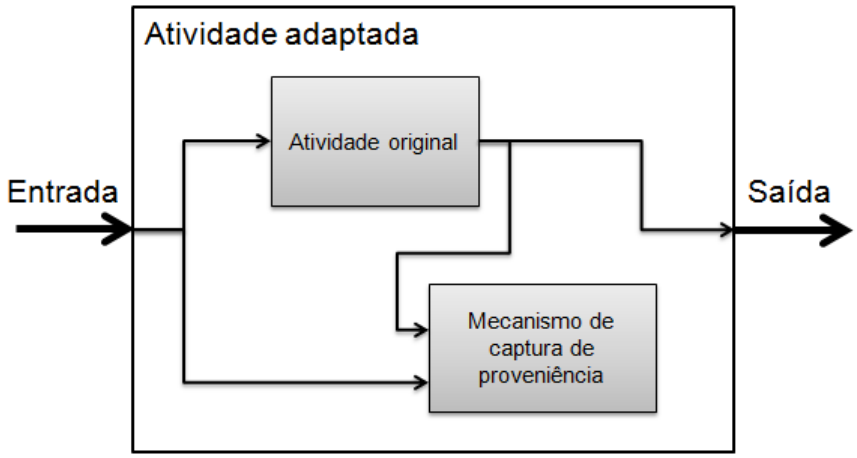

Figura 3.8 – Atividade adaptada utilizando estrutura de *wrapper*

Uma forma de construir uma estrutura *wrapper* no nível do *workflow* é utilizando o conceito de atividade composta, provida pela maioria dos *SGWfC*. Uma atividade composta agrega um conjunto de atividades, formando um *subworkflow*. Com isso, para cada atividade do *workflow*, uma atividade composta pode ser criada para "empacotar" a atividade original e adicionar atividades especiais para realizarem a captura e a publicação dos dados de proveniência. Essas atividades especiais são denominadas de atividades coletoras de proveniência (*provenance gathering activities - PGA*). Cada *PGA* é responsável por capturar e publicar informações de proveniência retrospectivas específicas como, por exemplo, sinalizações de início e fim de execução de atividades, artefatos produzidos, etc. A *API* de proveniência retrospectiva provida pelo *ProvManager* é utilizada para este propósito pelos *PGA*.

A Figura 3.9 ilustra um exemplo de *workflow* desenvolvido no *VisTrails*, sendo adaptado com o mecanismo de captura de proveniência do *ProvManager* a partir da estratégia de empacotamento utilizando atividades compostas. O *workflow* localizado na parte superior é o *workflow* original, composto por três atividades (*GetData*, *PrepareData*, *Simulate*), e o *subworkflow* localizado na parte inferior representa a atividade composta que substitui a atividade original *getData*. No *VisTrails*, a funcionalidade de atividade composta é denominada *Group*. É importante observar que alguns PGA são colocados antes da execução da atividade (PGA1), e outros são colocados após (PGA2 e PGA3). Essa decisão depende do tipo de informação que precisa ser coletada. Por exemplo, os PGA que usam a operação da API de proveniência retrospectiva *notifyActivityExecutionStart* devem ser executados antes da atividade original no *subworkflow*. O oposto acontece com os PGA que usam a operação da API

notifyActivityExecutionEnd. Observe também que dois ou mais PGA podem ser executados em paralelo para diminuir o tempo de execução do *workflow*.

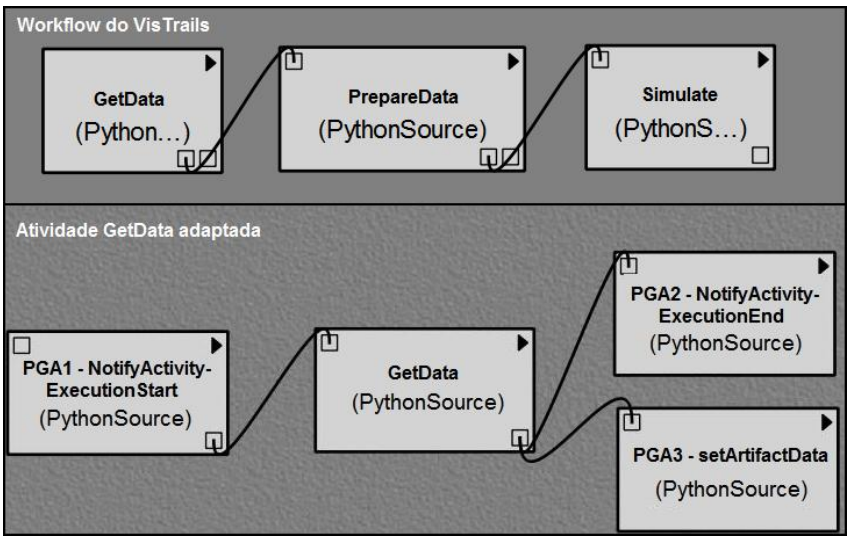

Figura 3.9 - Exemplo de instrumentação de *workflow* no VisTrails

3.8 *Agentes de proveniência*

No *ProvManager*, a máquina de processos *Charon* é utilizada para armazenar os dados de proveniência de um experimento. A *Charon* utiliza uma base de conhecimento implementada em *Prolog* para armazenar estes dados. Embora *Prolog* seja interessante no processo de derivação de dados de proveniência devido ao seu forte mecanismo de inferência, não é uma linguagem simples de ser usada em consultas para usuários que não possuem experiência em computação. Isso pode ser considerado uma barreira, visto que a maioria dos cientistas que manipulam um experimento é de áreas não tecnológicas. Além disso, mesmo utilizando uma linguagem de consulta simples, experimentos de larga escala são complexos de serem analisados devido à grande quantidade de dados disponíveis. Com isso, uma forma de subir o nível de abstração da análise dos dados de proveniência é utilizando a abordagem de agentes inteligentes da *Charon* para criar *agentes de proveniência*. Estes agentes são responsáveis por facilitar a geração de consultas pré-estabelecidas de proveniência e auxiliar a interpretação dos dados de proveniência do experimento.

Um agente inteligente é uma entidade que captura as informações do ambiente em que está inserido, processa estas informações, levando em conta informações próprias, tais como objetivos e planos, e responde ao ambiente (Russell e Norvig 1995).

Segundo essa abordagem, o agente de proveniência tem a mesma definição, porém com um enfoque no contexto de proveniência. De uma maneira geral, o agente de proveniência é responsável por interagir com o repositório de proveniência (no caso da abordagem *ProvManager*, a base de conhecimento da *Charon*) para fornecer respostas específicas para o cientista, de acordo com objetivos e planos pré-definidos.

A partir disso, uma estratégia para auxiliar o cientista no processo de análise do experimento é construir um conjunto de agentes que lidam com aplicações específicas de proveniência. Estes agentes devem ser construídos de forma pré-fabricada, permitindo que o cientista possa configurá-los de acordo com o cenário onde ele esteja inserido. A *Charon* permite essa característica, pois possui uma arquitetura extensível, viabilizando que novos agentes sejam criados de acordo com as necessidades dos cientistas.

Nas próximas seções, são descritos alguns exemplos de agentes de proveniência que podem ser criados para ajudar os cientistas no processo de análise de um experimento. É importante notar que a análise feita pelos agentes não ocorre apenas após a execução do experimento. Por exemplo, o agente monitor de proveniência, detalhado na Seção 3.8.1, atua durante a fase de execução do experimento fornecendo informações em tempo real sobre a execução do experimento para o cientista.

3.8.1 Agente monitor de proveniência

De maneira análoga aos mecanismos de monitoramento providos por alguns *SGWfC*, o *agente monitor de proveniência* realiza a mesma funcionalidade, que é prover, em tempo real, informações sobre a execução do experimento. No entanto, este agente trabalha em um nível mais alto, uma vez que realiza o monitoramento de experimentos que são executados em ambientes distribuídos, ou seja, experimentos que podem utilizar mais de um *SGWfC* e/ou máquinas de execução para serem executados. Neste tipo de cenário, o mecanismo de monitoramento de um *SGWfC* representa apenas uma região do experimento. A Figura 3.10 ilustra esta diferença de nível de monitoramento entre um *SGWfC* e o *agente monitor de proveniência*. Enquanto os *SGWfC* têm conhecimento apenas da execução do *workflow* que está implantado no seu sistema, o agente monitor acompanha a execução de todos os *SGWfC*, fornecendo uma

visão geral de toda a execução do experimento. Isto é possível devido ao mecanismo de captura de dados do *ProvManager* que coleta os dados de todos *SGWfC* envolvidos.

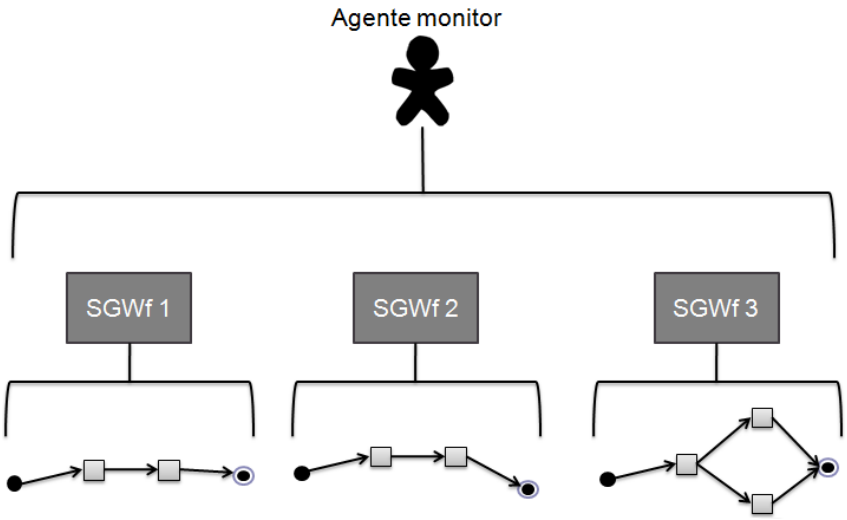

Figura 3.10 - Níveis de monitoramento entre *SGWfC* e agente monitor

O agente monitor de proveniência pode funcionar de dois modos: pró-ativo ou reativo. No modo pró-ativo, o agente verifica de tempos em tempos o repositório de proveniência para encontrar novas informações sobre a execução do experimento. No modo reativo, o agente é notificado por eventos, gerados pelo repositório, informando que novas informações foram armazenadas.

Além disso, este agente possui um mecanismo de configuração que permite que o cientista controle a quantidade e o tipo de informação que ele deseja monitorar. Esse tipo de configuração é importante, pois existem situações onde o cientista quer analisar apenas uma região do *workflow* e não quer ter sua atenção desviada devido a outras informações que não são importantes no momento.

3.8.2 Agente visualizador de proveniência

O *agente visualizador de proveniência* auxilia o cientista na recuperação e na análise de informações de execuções do *workflow* no repositório de proveniência. Este agente utiliza uma linguagem de consulta de alto nível, a fim de simplificar o processo de consulta para o cientista. As consultas escritas na linguagem de alto nível são traduzidas para *Prolog* e, posteriormente, enviadas para a *Charon*. O agente visualizador de proveniência trabalha com mecanismos de filtros que permitem a regulagem da granularidade da informação de proveniência. Estes filtros, por exemplo,

permitem que o cientista regule a informação a ser visualizada ajustando para um nível de atividade ou para um nível de *subworkflow*, ou seja, através do encapsulamento de um conjunto de atividades, formando uma atividade composta. Na literatura, esta funcionalidade é conhecida como "*user view*" (Cohen et al. 2006).

A Figura 3.11 apresenta dois exemplos de filtragem dos dados de proveniência em um *workflow*. O filtro 1 representa a linha tracejada que engloba as atividades A1, A2 e A3. Neste filtro, todos os dados trocados por essas atividades, ou seja, todo o tráfego de dados que ocorre dentro da área tracejada, não será apresentado para o usuário. O usuário terá conhecimento apenas dos dados que entraram, consumidos pela atividade A1, e dos que saíram da área tracejada, gerados pela atividade A3. O filtro 2 funciona de maneira análoga, porém engloba as atividades A3 e A4.

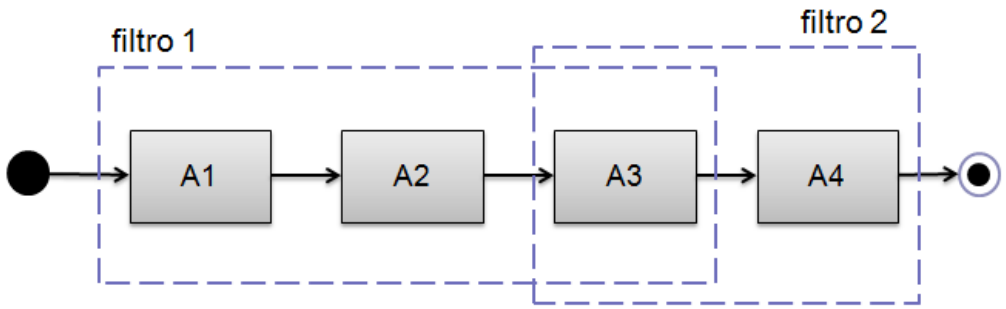

Figura 3.11 - Exemplo de filtro de dados de proveniência

3.8.3 Agente auditor de proveniência

O *agente auditor de proveniência* possui a responsabilidade de ajudar o cientista a realizar auditorias sobre os dados de proveniência. A partir deste agente, o cientista, por exemplo, pode descobrir qual foi o processo (i.e., sequência de atividades executadas e dados intermediários utilizados) que gerou determinados produtos. Em domínios como medicina, a auditoria é importante quando novas drogas são descobertas e o processo de geração das mesmas precisa ser identificado e verificado para viabilizar a criação de patentes (Miles et al. 2007b).

Outra possível funcionalidade desse agente é a sua aplicação na verificação do grau de utilização dos recursos de um experimento. O agente pode retornar estatísticas de um ou mais recursos, indicando se estes estão sendo realmente utilizados ou não. Este tipo de informação é interessante, pois ajuda o cientista a realizar otimizações no *workflow*. Por exemplo, um experimento que utiliza serviços *web* proprietários,

tarifados por uma taxa mensal fixa, pode ser ajustado para um esquema de tarifação por demanda, caso o cientista descubra que esses serviços são pouco utilizados mensalmente.

3.8.4 Agente depurador de proveniência

Dados de proveniência podem ajudar os cientistas a depurar um *workflow*, uma vez que contém informações sobre a execução do *workflow*. No entanto, uma execução do *workflow* pode gerar uma enorme quantidade de dados, dependendo do tamanho do experimento. Com isso, fazer uma análise manual desses dados é uma tarefa demorada e suscetível a erros.

Para mitigar esse problema, o *agente depurador de proveniência* entra em ação. Este agente facilita o processo de depuração, pois utiliza proveniência tanto prospectiva quanto retrospectiva para encontrar inconformidades no *workflow*. A proveniência prospectiva é utilizada para obter a especificação do *workflow*, enquanto que a retrospectiva é usada para obter os dados sobre a execução do *workflow*. Com a análise em conjunto desses dois tipos de informação, o agente depurador indica ao cientista atividades ou regiões do *workflow* que não estão funcionando de maneira adequada.

3.9 Considerações finais

Considerando os problemas das abordagens de gerenciamento de proveniência, destacados no Capítulo 2, a abordagem *ProvManager*, apresentada neste capítulo, visa solucioná-los provendo mecanismos/funcionalidades com as seguintes características:

1) **Gerenciamento de proveniência de experimentos executados em ambientes distribuídos:** Permitindo que os dados de proveniência de experimentos complexos que são executados em diversos *SGWfC* ou máquinas de execução sejam capturados, armazenados em um repositório central e acessados de forma integrada.

2) **Mecanismo de captura de proveniência independente de *SGWfC*:** Possibilitando que as informações de proveniência sejam capturadas de qualquer sistema de *workflow* ou máquina de execução e, com isso, viabilizando o gerenciamento de proveniência de experimentos executados em ambientes distribuídos.

3) **Captura abrangente de dados de proveniência:** *ProvManager* captura dados de proveniência tanto do tipo prospectivo quanto do tipo retrospectivo. Além disso, *ProvManager* faz um mapeamento desses dois tipos de proveniência, permitindo que a partir de uma informação retrospectiva se obtenha informações prospectivas relacionadas, e vice-versa. Por exemplo, obter o tempo de execução (proveniência retrospectiva) da atividade "A" do *workflow* "W" na versão "5" (proveniência prospectiva).

4) **Acesso integrado aos dados de proveniência do experimento:** Permitindo que os dados de proveniência de experimentos distribuídos em diversos sistemas, com interfaces e modelo de dados de proveniência diferentes, sejam acessados a partir de uma interface única e padronizada.

5) **Facilidade de configuração:** A abordagem *ProvManager* apresenta mecanismos que facilitam a sua configuração, evitando que o usuário tenha que realizar tarefas complexas que não estão relacionadas diretamente ao processo de análise de proveniência. Um exemplo é o mecanismo de adaptação automática das atividades para suportar o mecanismo de captura de proveniência.

A Tabela 2.1 apresenta o comparativo das abordagens de proveniência realizado no Capítulo 2 atualizado com a abordagem *ProvManager*. Nessa tabela, é possível observar que *Provmanager* é a única abordagem a dar suporte a experimentos que são executados em ambiente com alta complexidade em heterogeneidade (descritos na Seção **Error! Reference source not found.**), além de ser a abordagem a satisfazer um maior conjunto dos critérios estabelecidos. Vale ressaltar que as abordagens utilizadas no comparativo apresentam finalidades específicas e podem ter sido desfavorecidas devido à perspectiva de comparação de interesse desse trabalho.

Com isso, com o objetivo de concretizar as ideias propostas neste capítulo, o Capítulo 4 apresentará um protótipo implementado segundo a abordagem *ProvManager*.

Tabela 3.1 - Quadro comparativo atualizado com as abordagens de proveniência

Critérios	*ProvManager*	*Karma*	*PASOA*	*Wings/ Pegasus*	*VisTrails*	*myGrid/ Taverna*	*Kepler*	*Swift*
Complexidade de distribuição	*Baixa*	*Baixa*	*Baixa*	*Baixa*	*Baixa*	*Baixa*	*Baixa*	*Baixa*
Complexidade de heterogeneidade	*Alta*	*Baixa*	*Baixa*	*Baixa*	*Baixa*	*Baixa*	*Baixa*	*Baixa*
Suporte de uso de aplicações proprietárias	*Sim*	*Não*	*Não*	*Sim*	*Sim*	*Sim*	*Sim*	*Sim*
Independência de *SGWfC*	*Sim*	*Não*	*Sim*	*Não*	*Não*	*Não*	*Não*	*Não*
Tipo de proveniência	*Prosp e Retrosp*	*Prosp e Retrosp*	*Prosp e Retrosp*	*Prosp e Retrosp*	*Prosp e Retrosp*	*Prosp e Retrosp*	*Prosp e Retrosp*	*Prosp e Retrosp*
Mapeamento entre proveniência prospectiva e retrospectiva	*Sim*	*Sim*	*Sim*	*Não*	*Não*	*Sim*	*Sim*	*Sim*
Versionamento	*Sim*	*Não*	*Não*	*Não*	*Sim*	*Não*	*Sim*	*Não*
Mecanismo de captura	*Atv (adaptado)*	*Atv*	*Atv*	*Wf e Atv*	*Wf*	*Wf*	*Wf*	*Wf*
Complexidade de utilização	*Baixa*	*Alta*	*Alta*	*Média*	*Baixa*	*Média*	*Baixa*	*Alta*
Visualização de dados	*Médio*	*Médio*	*Nulo*	*Nulo*	*Alto*	*Alto*	*Nulo*	*Nulo*
Modelo de dados	*Relacional*	*Relacional*	*XML*	*RDF*	*Relacional*	*RDF*	*XML*	*Relacional*

Capítulo 4. Protótipo

4.1 Introdução

Este capítulo apresenta o protótipo desenvolvido segundo a abordagem *ProvManager*, proposta no Capítulo 3. Para facilitar a demonstração das funcionalidades do protótipo, um experimento científico é utilizado como exemplo de uso, explorando as principais características da abordagem *ProvManager*. Esse experimento, pertencente ao domínio de cristalografia, serve para determinar a disposição de átomos em sólidos (Drenth 1999) e foi definido para a quarta edição do evento *Provenance Challenge* (ProvChallenge 2011), que ocorre alternadamente ao *workshop IPAW* (IPAW 2010), para discutir os desafios de proveniência. Além disso, do ponto de vista técnico, é realizado um detalhamento sobre as tecnologias utilizadas no protótipo.

A Seção 4.2 discute os detalhes técnicos sobre a implementação do protótipo, apresentando as principais tecnologias utilizadas, padrões adotados, etc. A Seção 4.3 apresenta o experimento de cristalografia que será utilizado como exemplo na demonstração da utilização do protótipo. A Seção 4.4 demonstra o protótipo em funcionamento, apresentando suas principais funcionalidades seguindo um processo de utilização, definido na própria seção.

4.2 Detalhamento técnico

A Figura 4.1 ilustra a arquitetura do protótipo através de um diagrama de implantação UML. O protótipo foi implementado na forma de aplicação *web*, utilizando a tecnologia *Java EE* (Oracle 2010a). O estilo arquitetural *MVC* (Krasner e Pope 1988) também foi utilizado para permitir uma melhor separação entre a lógica de negócio e a lógica de apresentação. Além disso, o *framework JSF (Java Server Faces)* (Oracle 2010b) foi utilizado para desenvolver a aplicação *web* segundo a tecnologia *Java EE*. *JSF* implementa o estilo arquitetural *MVC* e possui um conjunto de componentes pré-fabricados que facilitam o desenvolvimento de aplicações *web*. Por exemplo, *JSF* disponibiliza componentes de internacionalização e acessibilidade, mecanismos de

controle de sessão, mecanismos de validação de formulários, etc. Além dos componentes padrões disponibilizados, é possível utilizar componentes desenvolvidos por terceiros, uma vez que *JSF* permite o desenvolvimento de componentes de interface gráfica que possam ser reutilizados em outras aplicações. Um exemplo destes componentes, é a biblioteca *RichFaces* (Jboss 2010a), disponibilizada pela comunidade *Jboss* (Jboss 2010b), que fornece interfaces gráficas avançadas feitas em *Ajax* (Holdener III 2008). A partir desta biblioteca, o usuário ganha agilidade no desenvolvimento, pois não precisa ter conhecimentos avançados sobre *Ajax* para trabalhar com páginas dinâmicas complexas. No contexto do protótipo, o *RichFaces* foi utilizado na construção de tabelas e menu dinâmicos.

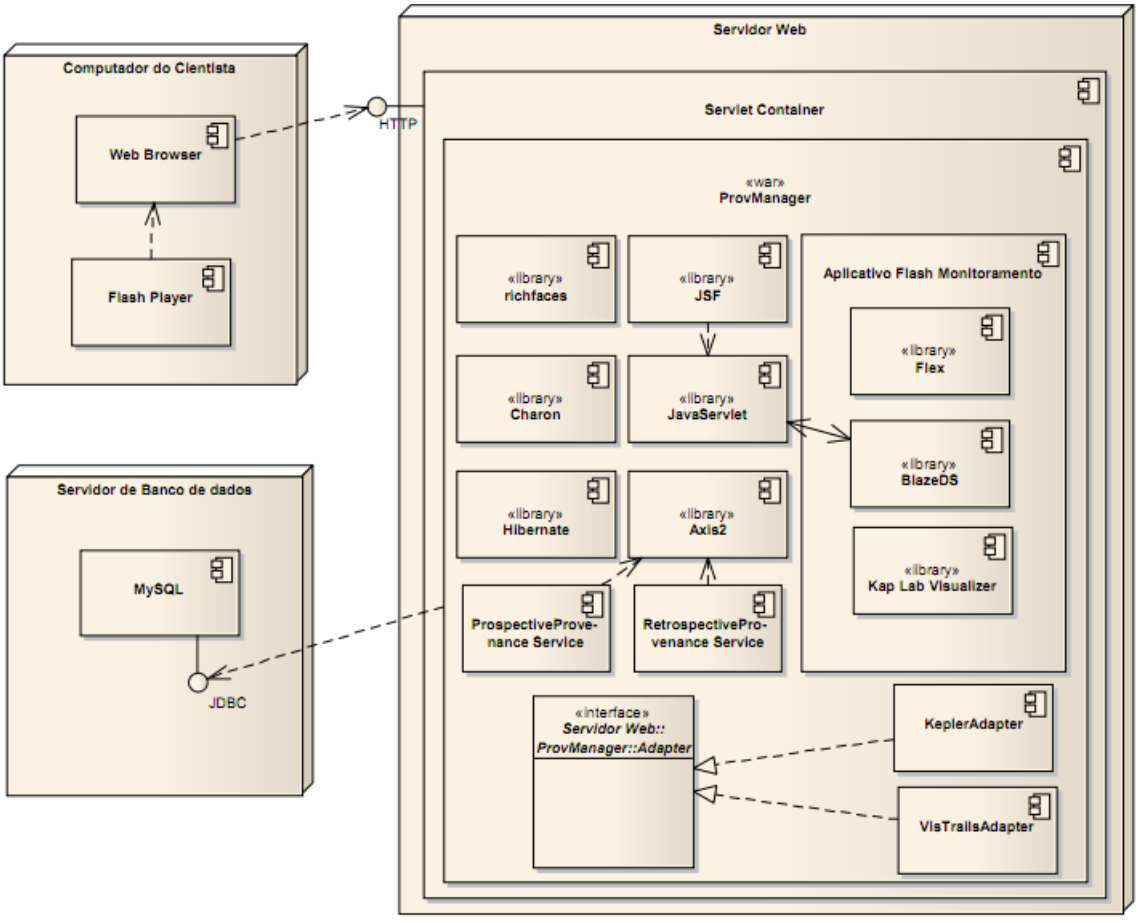

Figura 4.1 – Arquitetura do protótipo

Além dos componentes gráficos disponibilizados pelo *JSF* e *RichFaces*, foi desenvolvida uma aplicação em *Flash* (Adobe 2010a), a partir da plataforma *Flex* (Adobe 2010b), para representar graficamente o monitoramento da execução de um *workflow*. A biblioteca *Kap Lab Visualizer* (Kap IT 2010) foi utilizada como base para desenvolver um visualizador de elementos gráficos (no caso do protótipo, um

workflow). Além disso, a ferramenta *BlazeDS* (Adobe 2010c) foi utilizada para permitir a comunicação da aplicação *web Java* com o aplicativo *Flash*. Essa comunicação é necessária devido às interações entre as duas aplicações durante o monitoramento da execução do experimento. Por exemplo, em determinado momento, a aplicação *web* precisa passar atualizações sobre a execução do *workflow* para a aplicação *Flash*, ou então a aplicação *Flash* recebe eventos do usuário e repassa essas informações para a aplicação *web Java*. O resultado do mecanismo de monitoramento é apresentado na Seção 4.4.4.

Com relação à persistência, o protótipo persiste suas informações em um banco de dados *MySQL* (MySQL 2010). O *framework Hibernate* (Jboss 2010c) também foi utilizado para automatizar o mapeamento das informações do banco de dados com os objetos da aplicação *web Java*. Além do mapeamento objeto-relacional, o *Hibernate* fornece diversas outras vantagens, a saber: gerenciamento de transações; mecanismo de *pool* de conexões, permitindo a reutilização de conexões já utilizadas em outras transações; e mecanismo de carga tardia (*lazy load)*, que retarda o carregamento de certas informações do banco de dados com o objetivo de ganhar desempenho; dentre outras.

Como foi definido na abordagem *ProvManager*, o protótipo utiliza a máquina de processos *Charon* para armazenar as informações de proveniência e derivar novas informações através de mecanismos de inferência e de sua infra-estrutura de agentes inteligentes. Para publicar as informações de proveniência na *Charon*, foi criada uma classe, denominada *CharonAPI,* com todos os métodos de publicação de proveniência prospectiva e retrospectiva. No entanto, os métodos da classe *CharonAPI* precisam ser acessados remotamente pelas atividades coletoras de proveniência (*PGA*) que são inseridas nos *workflows*, sendo executadas em diferentes máquinas em um ambiente distribuído. Com isso, dois componentes foram desenvolvidos e expostos como serviços *web*: *ProspectiveProvenanceService* e *RetrospectiveProvenanceService*. Estes serviços representam os componentes da camada de *API* de serviços da arquitetura do *ProvManager*, definida na Seção 3.4. Para expor essas duas classes como serviço *web*, foi utilizado o *framework Axis2* (Apache 2010).

Foram desenvolvidos dois adaptadores de proveniência específicos de *SGWfC*, que realizam a configuração do mecanismo de captura de proveniência nos *workflows*: *adaptador de VisTrails* (Palmieri e Portugal 2010) e *adaptador de Kepler* (Araujo

2010). Estes dois adaptadores, desenvolvidos em dois projetos finais de curso de graduação, foram codificados em *Java* e implementam uma interface requerida pelo protótipo para realizarem a comunicação. Esta interface, denominada *Adapter*, precisa ser implementada por todo adaptador específico de *SGWfC* utilizado no protótipo. De acordo com a Figura 4.1, os componentes *VistrailsAdapter* e *KeplerAdapter* implementam a interface *Adapter* para o *adaptador de VisTrails* e para o *adaptador de Kepler*, respectivamente.

4.3 Experimento de cristalografia

Cristalografia é a ciência experimental de determinar a disposição de átomos em sólidos (Drenth 1999). Métodos cristalográficos dependem da análise do padrão das difrações que surgem de uma amostra de cristal quando exposta a feixes de raios-X. No experimento de cristalografia apresentado nesta seção, cientistas realizam uma série de atividades para produzir um conjunto de coordenadas de átomos a partir de um cristal que são publicadas posteriormente em uma base de dados pública.

A Figura 4.2 apresenta o *workflow* conceitual (*i.e.*, definição de alto nível do *workflow* independente de sistema de *workflow*) do experimento de cristalografia. O experimento é dividido em cinco estágios:

- **Primeiro estágio:** raios-X são aplicados em amostras de cristal, gerando arquivos de imagens de difração. Estas imagens são inspecionadas pelo cientista, que decide se estas vão ou não ser utilizadas no próximo estágio.
- **Segundo estágio:** as imagens de difração são processadas para extrair os pontos de intensidade. No final, um arquivo de reflexão com a média das intensidades é gerado.
- **Terceiro estágio:** o arquivo de reflexão é processado para definir a estrutura da proteína. Para isso, é realizada uma série de iterações para refinar o resultado.
- **Quarto estágio:** os resultados da estrutura da proteína são publicados e disponibilizados na *web* em uma página de discussão, para que sejam avaliados e novos detalhes sejam acrescentados. No final, um artigo descrevendo os resultados do experimento é gerado.
- **Quinto estágio:** O artigo é publicado em uma revista de bioquímica para que novos feedbacks sobre o resultado sejam adquiridos.

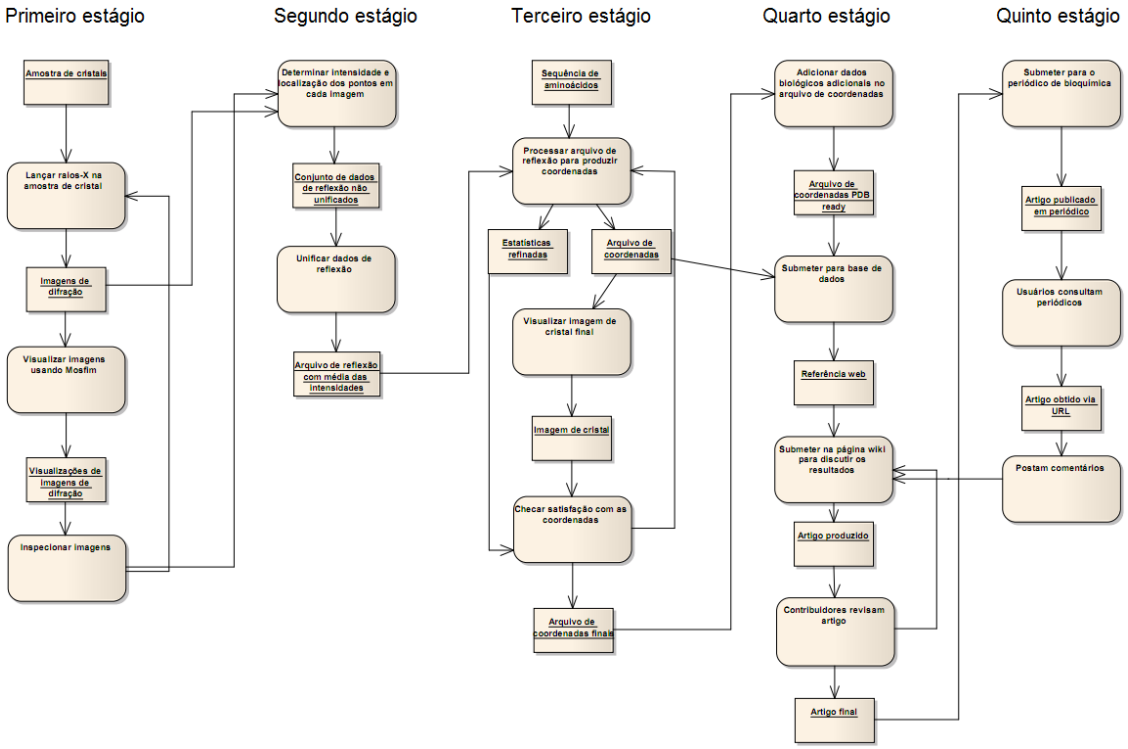

Figura 4.2 - *Workflow* conceitual do experimento de cristalografia (adaptado de (ProvChallenge 2011))

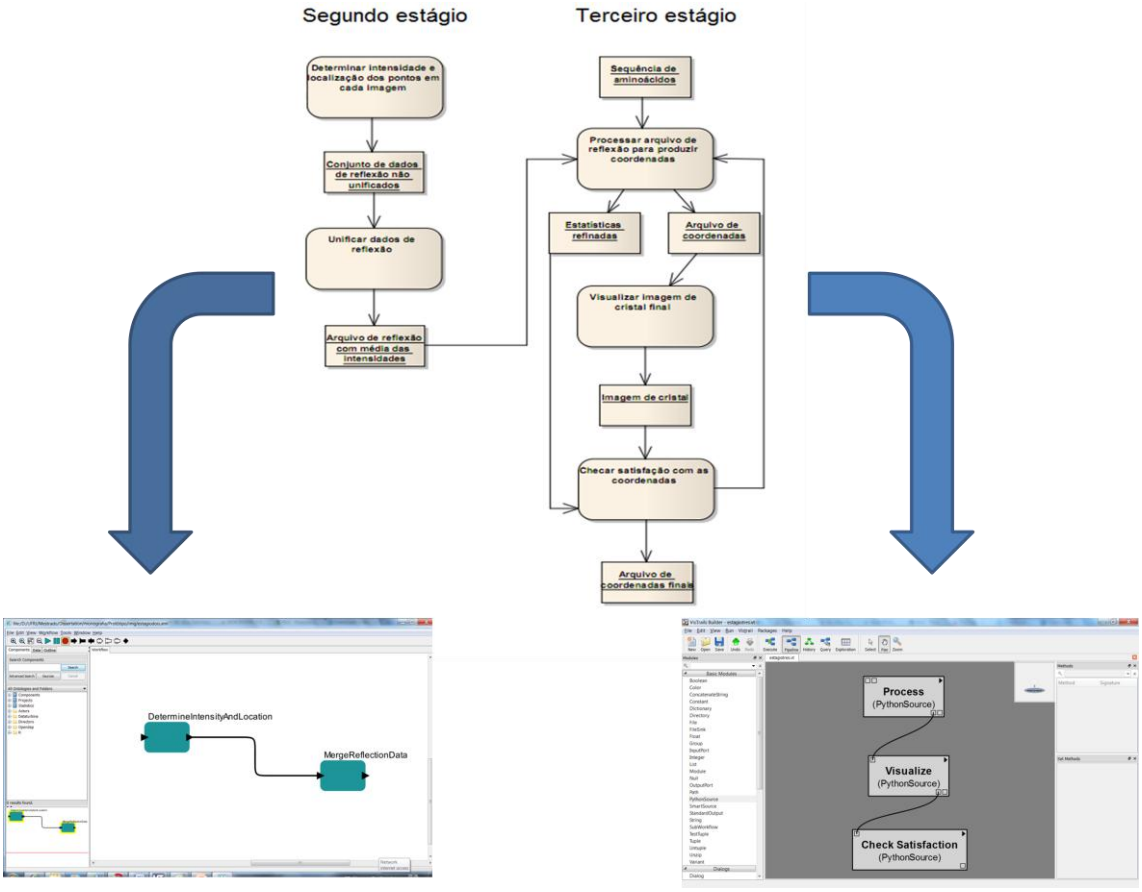

Figura 4.3 - *Workflow* simplificado implementado no *Kepler* e *VisTrails*

Por questões de simplicidade, apenas o segundo e o terceiro estágio deste *workflow* são utilizados como exemplo neste capítulo. A Figura 4.3 apresenta a estrutura do experimento com apenas estes dois estágios, implementados nos *SGWfC VisTrails* e *Kepler*. Com o intuito de explorar as características do *ProvManager*, foi assumido um cenário onde estes dois sistemas foram utilizados em conjunto devido às características e necessidades de cada estágio do *workflow*. O estágio dois, por exemplo, é executado no Kepler, pois este sistema apresenta as bibliotecas necessárias para determinar os pontos de intensidade das imagens recolhidas dos cristais. Além disso, esse *workflow* é reutilizado de outro experimento preexistente. O estágio três é executado no *VisTrails* para utilizar os recursos de visualização gráfica de dados providos por este sistema, uma vez que este estágio possui atividades onde o cientista precisa visualizar as imagens dos cristais.

4.4 Utilização do protótipo

A utilização do protótipo pode ser definida por um processo que contempla as seguintes atividades:

1. Cadastrar um novo experimento;
2. Criar uma nova versão para o experimento ou trabalhar com a versão corrente;
3. Cadastrar um ou mais *workflows* em uma determinada versão do experimento:
    a. Durante o cadastro, deve(m) ser passada as especificação(ões) do(s) *workflow(s)* desenvolvida(s) no(s) *SGWfC*;
    b. O adaptador de proveniência realiza a adaptação da(s) especificação(ões) do(s) *workflow(s)*;
    c. Durante a adaptação, os dados de proveniência prospectiva são extraídos da especificação(ões) do(s) *workflow(s)*;
    d. Após a adaptação, deve-se baixar as especificação(ões) adaptada(s) e carregá-la(s) no(s) *SGWfC* para ser(em) executada(s);

4. Configurar as dependências dos *workflows*;
5. O(s) *Workflow(s)* é(são) executado(s) nos *SGWfC*;
6. As informações de proveniência retrospectiva são capturadas pelas atividades *PGA* e publicadas no protótipo através da *API* de serviços;
7. Durante a execução do experimento, é possível realizar o monitoramento da execução diretamente no protótipo através de uma interface gráfica que apresenta as atividades do(s) *workflow(s)* que já foram, estão, ou que serão ainda executadas; e
8. Após a execução do experimento, é possível realizar consultas sobre o experimento no protótipo através de uma interface gráfica utilizando a linguagem *Prolog*.

A Figura 4.4 e a Figura 4.5 ilustram esse processo em um diagrama de atividades UML.

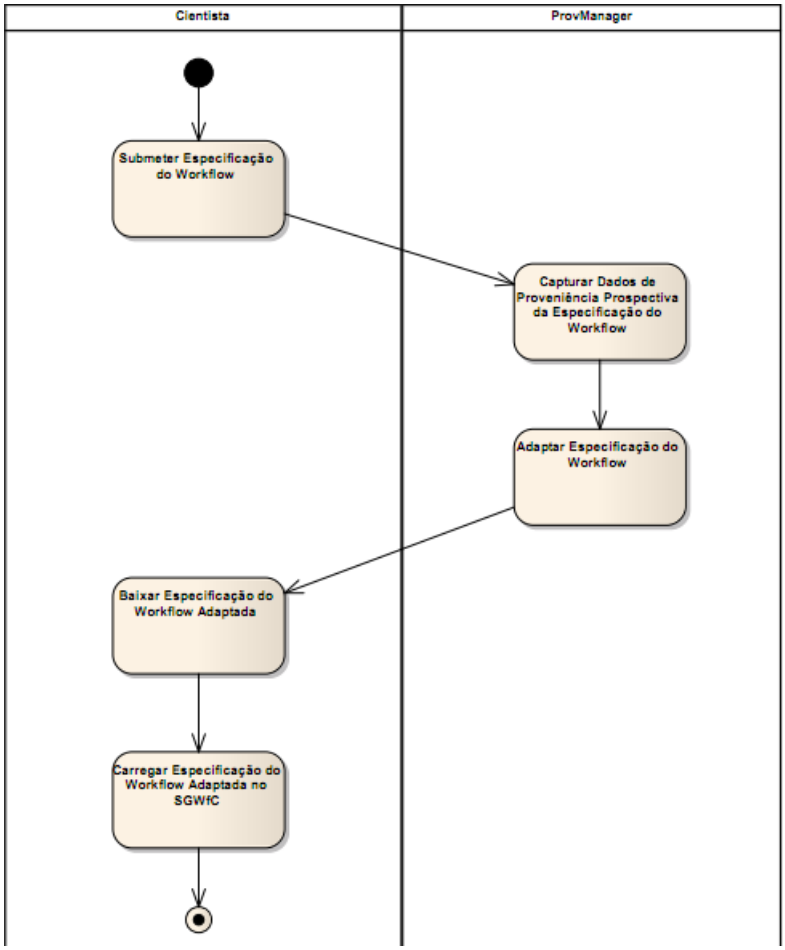

Figura 4.4 – Diagrama da atividade *Cadastrar workflow*

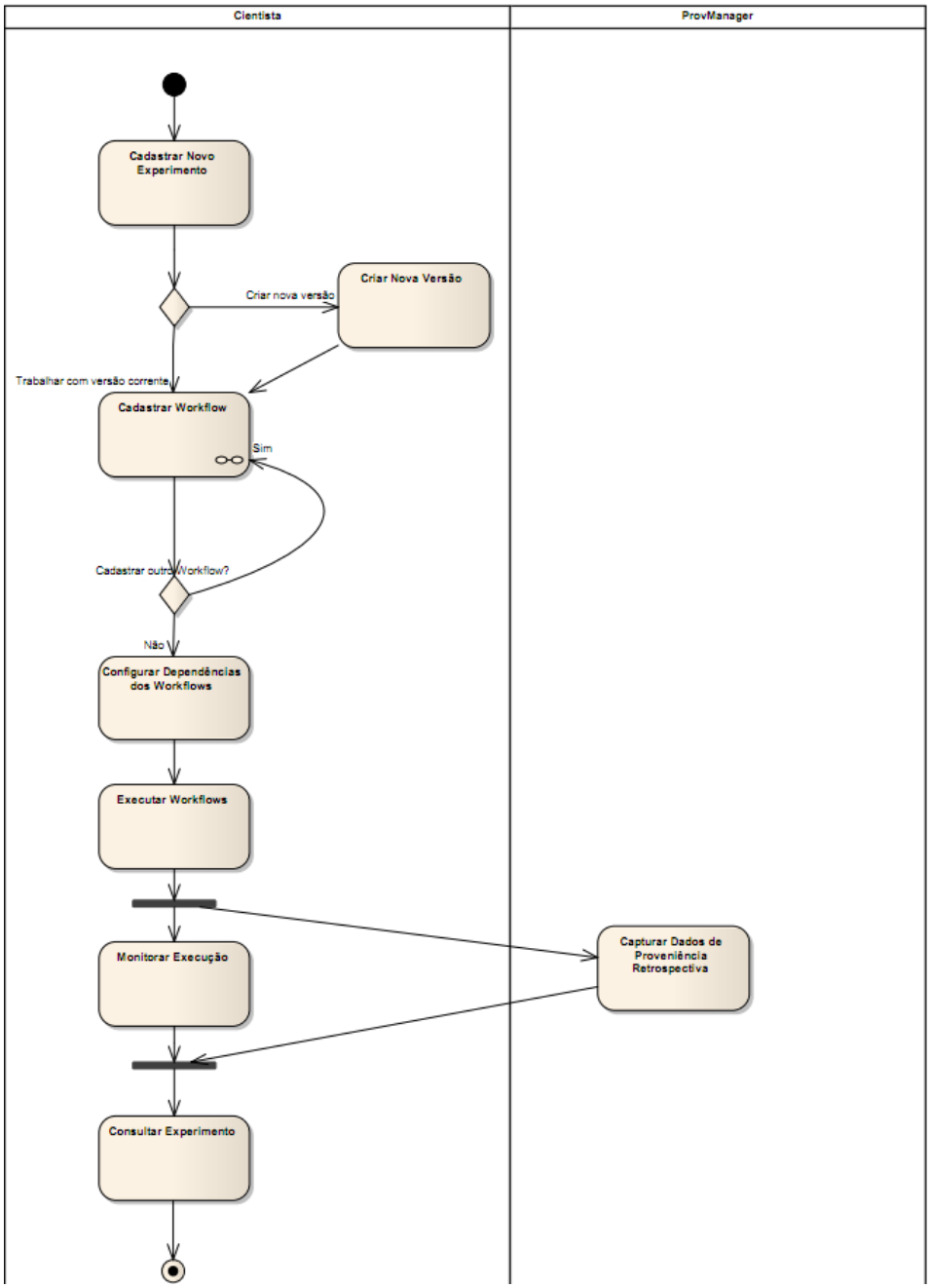

Figura 4.5 – Diagrama de atividades do processo de utilização do protótipo

Essas atividades que envolvem o processo de utilização do protótipo são detalhadas nas próximas subseções, tomando como exemplo o experimento de cristalografia definido na Seção 4.3.

4.4.1 Acesso ao ProvManager

Uma versão beta do protótipo está em execução para teste e pode ser acessada a partir do endereço http://reuse.cos.ufrj.br/provmanager. Uma conta com *login* e senha

precisa ser criada para viabilizar o acesso. A Figura 4.6 apresenta a tela principal do protótipo. O sistema apresenta um mecanismo de controle de acesso, viabilizando o acesso a apenas usuários autorizados. Além disso, um mecanismo de internacionalização foi incorporado, permitindo a visualização das páginas em inglês e português.

Figura 4.6 - Tela inicial do *ProvManager*

No protótipo, um experimento é estruturado da mesma maneira como foi definido no *modelo de proveniência* da abordagem *ProvManager*, apresentada na Seção 3.5. Ou seja, o protótipo faz uso das seguintes entidades:

- **Experimento:** Entidade que representa conceitualmente um experimento. Um experimento é constituído por um ou mais *workflows*, viabilizando a representação de experimentos que são executados em ambientes distribuídos.
- **Versão:** Entidade que representa um estado de configuração do experimento em determinado instante de tempo. Uma versão contém os *workflows* que foram definidos em um experimento em determinado instante de tempo.
- ***Workflow*:** Entidade que representa um fragmento da definição de execução do experimento, ou seja, um conjunto de atividades que seguem uma determinada ordem de execução. Um *workflow* é instanciado e executado em um *SGWfC* específico.

4.4.2 Cadastro de experimento

Para cadastrar um novo experimento, um usuário, devidamente autenticado, deve acessar o *menu Experimento* e selecionar a opção *Novo*, de acordo com a Figura 4.7.a. A tela de cadastro de experimento é composta por um formulário simples que possui os campos: *nome* e *descrição* do experimento. A Figura 4.7.b apresenta essa tela com o formulário preenchido, referente ao experimento de cristalografia. Ao clicar no botão *confirmar*, o experimento é cadastrado e uma nova tela que lista os experimentos cadastrados pelo usuário é apresentada, de acordo com a Figura 4.8.

Figura 4.7 - (a) Tela de menu de experimento; (b) Tela de cadastro de novo experimento

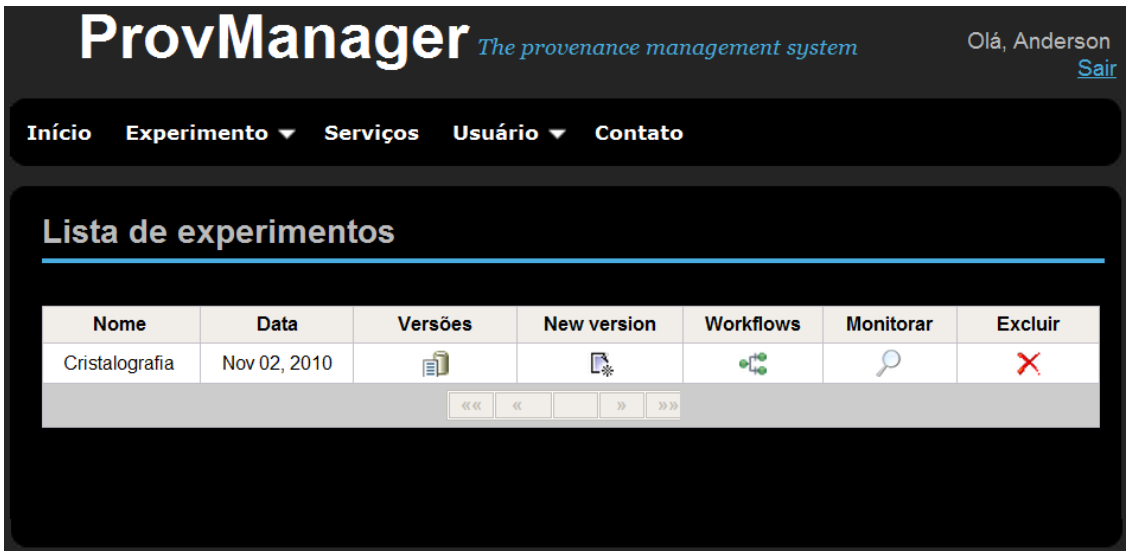

Figura 4.8 - Tela de lista de experimentos.

Na *lista de experimentos* (Figura 4.8), cada experimento possui as seguintes opções: *versões*, para listar as versões existentes do experimento; *nova versão*, para gerar uma nova versão do experimento sem *workflows* cadastrados; *workflows*, para listar os *workflows* cadastrados no experimento; *monitorar*, para acompanhar a execução do experimento, caso exista alguma instância do experimento em execução; e *excluir*, para remover o experimento da lista de experimentos do usuário.

As opções *nova versão* e *versões* presentes na Figura 4.8 são responsáveis por realizar o versionamento de um experimento. Quando um experimento é criado, uma versão inicial é gerada e associada ao experimento automaticamente. Todas as ações feitas no experimento (e.g., edição de *workflows*, execução de experimento, etc.) estarão associadas a essa versão. Quando o usuário desejar "congelar" essa versão em um determinado estado, este deve criar uma *nova versão*. Esta nova versão não possui informações sobre o experimento em um primeiro momento, sendo necessário que o usuário realize as configurações necessárias a partir do zero. Já a versão anterior não poderá ser mais alterada, servindo apenas como histórico. A Figura 4.9 ilustra a tela de *lista de versões de experimento* que apresenta todas as versões de um experimento. Nesta lista, o usuário escolhe uma versão específica para listar os *workflows* do experimento, podendo posteriormente visualizá-los ou alterá-los (caso seja a última versão, uma vez que só esta pode sofrer alterações).

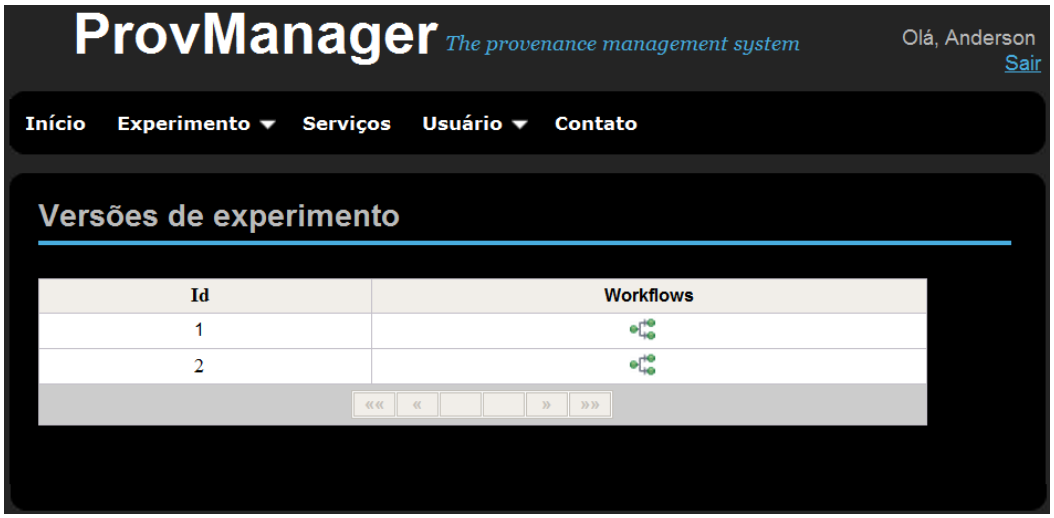

Figura 4.9 - Tela de lista de versões de um experimento

4.4.3 Cadastro de workflow

O experimento de cristalografia simplificado, descrito na Seção 4.3, apresenta um *workflow* dividido em dois estágios, que foram instanciados em *SGWfC* distintos.

De acordo com o modelo do *ProvManager*, estes dois estágios representam os *workflows* do experimento de cristalografia. Para cadastrar estes dois *workflows* no *ProvManager*, o usuário tem que acessar a funcionalidade *lista de workflows*. Esta funcionalidade pode ser acessada de duas formas: na tela de *lista de experimentos*, acessando o *link versões* e, posteriormente, listando os *workflows* de uma versão específica; ou então acessando diretamente o *link workflows* na *lista de experimentos*. Nesta última opção, *ProvManager* lista automaticamente os *workflows* associados à versão mais recente do experimento, ou seja, a versão que pode ser alterada.

Na tela *lista de workflows*, apresentada na Figura 4.10, os *workflows* cadastrados em uma determinada versão do experimento são apresentados em uma tabela. Acima dessa tabela, há um formulário para adicionar novos *workflows*. Para cadastrar um *workflow*, as informações necessárias são: *nome*; *tipo* do *SGWfC* no qual o *workflow* é executado; e a *especificação do workflow* definida no *SGWfC*. Estas duas últimas informações são necessárias para viabilizar a adaptação do *workflow* para suportar o mecanismo de captura de proveniência do *ProvManager*. A informação que descreve o tipo de *SGWfC* é necessária para que o *ProvManager* possa selecionar os adaptadores de proveniência adequados, uma vez que existe um adaptador de proveniência específico para cada *SGWfC* (como foi discutido na Seção 3.7). Já o arquivo da especificação do *workflow* é utilizado para extrair as informações de proveniência prospectiva e para construir (*i.e.*, adaptar) a nova especificação do *workflow*, adicionando as atividades de coleta de proveniência (*PGA*).

Figura 4.10 - Tela de lista de *workflows*

De acordo com o experimento de cristalografia, é necessário que sejam cadastrados dois *workflows* no *ProvManager*. Um *workflow* com o nome "estagio2", do tipo *Kepler*, passando o arquivo de especificação do *workflow* "estagio2.xml", e outro *workflow* com o nome "estagio3", do tipo *VisTrails*, passando o arquivo de especificação do *workflow* "estagio3.vt". Uma vez cadastrados, estes dois *workflows* são listados no *ProvManager*, de acordo com a Figura 4.10. Cada *workflow* listado pode ser configurado com relação às suas dependências a outros *workflows*, através da funcionalidade *dependências*. A relação de dependência estipula a ordem de execução dos *workflows* no experimento. De acordo com o experimento de cristalografia, o *workflow* "estagio2" é configurado para ser executado antes que o *workflow* "estagio3". Em outras palavras, o *workflow* "estagio3" *depende* que o *workflow* "estagio2" seja executado primeiro. A Figura 4.11 apresenta a tela de configuração de dependência de *workflows*. Atualmente, essa dependência não pode ser feita no nível de atividade, ou seja, determinar que uma atividade do *workflow* "estagio3" depende de outra atividade do *workflow* "estagio2". Além disso, não é possível definir os dados que são gerados em um *workflow* e consumidos por outro *workflow*.

Figura 4.11 - Tela de configuração de dependências

Por fim, o usuário deve baixar as especificações adaptadas para serem carregadas de volta nos *SGWfC*, a fim de que os *workflows* sejam executados com o mecanismo de captura de proveniência instrumentado. Com isso, a cada execução do *workflow*, as informações de proveniência serão capturadas e enviadas para o

ProvManager através da *API* de serviços *web* (como foi discutido no Capítulo 3). A título de exemplo, a Figura 4.12 apresenta o *workflow* "estagio2" feito em *Kepler* adaptado com o mecanismo de captura de proveniência. As duas atividades (*DetermineIntensityAndLocation* e *MergeReflectionData*) foram encapsuladas em duas atividades compostas de mesmo nome, de acordo com a Figura 4.12.a. Externamente, a mudança do *workflow* é mínima, apenas as atividades simples foram substituídas por atividades compostas. Isso é importante, pois não dificulta a compreensão do *workflow* por parte do usuário. Os detalhes da adaptação acontecem dentro das atividades compostas. Em cada atividade composta, são inseridas, além da atividade original, atividades especiais (*PGA*) responsáveis pela coleta dos dados de proveniência retrospectiva. A Figura 4.12.b apresenta com mais detalhes a estrutura da atividade "*MergeReflectionData"* adaptada.

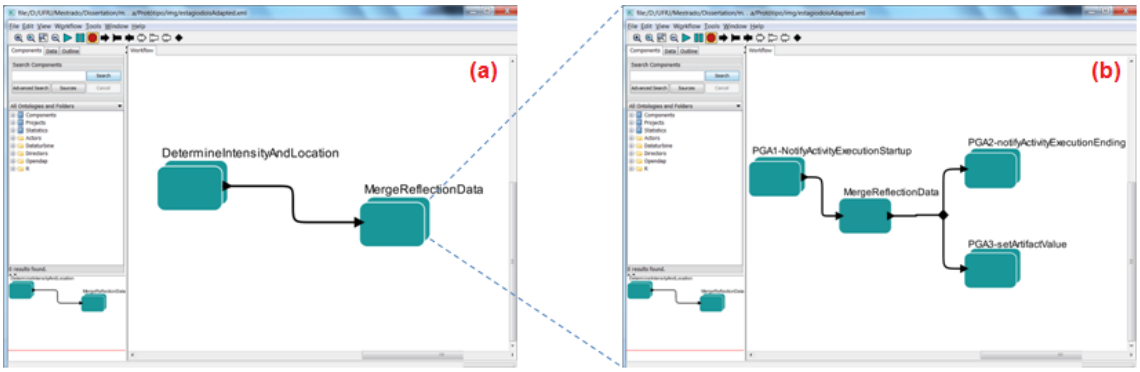

Figura 4.12 - Exemplo de *workflow* "estagio2" escrito em *Kepler* adaptado com o mecanismo de captura de proveniência

4.4.4 Monitoramento de execução de experimento

Durante a execução do experimento, *ProvManager* disponibiliza uma interface de monitoramento que permite uma visão geral de execução de todos os *workflows* envolvidos no experimento. Nessa interface, o *workflow* é exibido utilizando uma notação similar ao diagrama de atividades da *UML* (OMG 2010), com a exceção da não utilização dos elementos de início e fim e objetos (produtos requeridos ou gerados pelas atividades). Com isso, os elementos utilizados são: atividade simples ( ), atividade composta ( ), sincronismo ( ) e decisão ( ). A descrição desses elementos é similar aos elementos de mesmo nome do modelo de proveniência definido na Seção 3.5. Além disso, é utilizado um conjunto de cores para representar o estado de execução dos elementos do *workflow*. As cores e seus significados são listados abaixo:

- **Cinza:** Elemento pronto pra ser executado, ou seja, todas as suas dependências foram satisfeitas.
- **Amarelo:** Elemento que está sendo executado no presente momento.
- **Verde:** Elemento que já foi executado e completou sua execução com sucesso.

A Figura 4.13 apresenta a tela de monitoramento no *ProvManager*. Como já foi discutido na Seção 4.2, o aplicativo de monitoramento é feito em *Flash* e encontra-se incorporado na aplicação *web*. Na parte superior do aplicativo de monitoramento, existem algumas ferramentas para facilitar a visualização do *workflow* como, por exemplo, mecanismo de *zoom* e mecanismos de *drag-and-drop* (Jia et al. 2006) para manipular a disposição dos elementos do *workflow* na tela. Em especial, um mecanismo relevante é o de *layout*, que disponibiliza diversos modelos de visualização, possibilitando que um *workflow* seja reorganizado em diversas disposições na tela (*e.g.*, radial, hierárquico, circular, etc.).

Além disso, o mecanismo de monitoramento possui recursos de ajuste de nível de detalhamento do monitoramento do *workflow*, similares às funcionalidades de *drill down* e *drill up* (Gorla 2003), comumente utilizadas em *data warehouse* para filtrar o nível de detalhamento da informação. A partir disso, o detalhamento do monitoramento pode ser aprofundado para o nível do *subworkflow* de uma atividade composta (*drill down*), ou então diminuído, passando para o nível do *workflow* no qual esta atividade composta está inserida (*drill up*). A Figura 4.13 apresenta a tela de monitoramento, usando como exemplo o *workflow* de cristalografia. De acordo com o mecanismo, o estágio dois do *workflow* de cristalografia já foi executado e o estágio três está sendo executado. Por exemplo, para se obter mais detalhes sobre a execução do estágio três, o usuário deve selecionar essa atividade composta e clicar no botão "*drill down*" no *menu*. Com isso, a tela de monitoramento é atualizada com o *workflow* do estágio três, como é ilustrada na Figura 4.14. Para retornar para o nível anterior, o usuário deve clicar no botão "*drill up*".

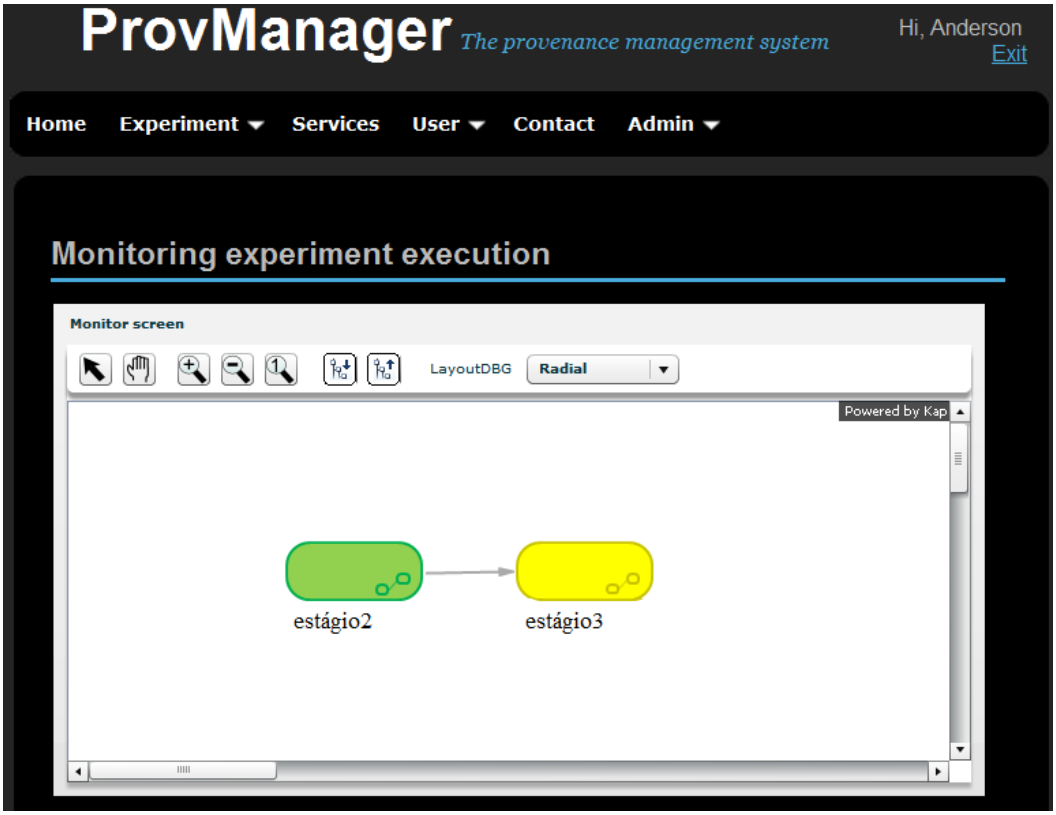

Figura 4.13 - Tela de monitoramento de execução de experimento

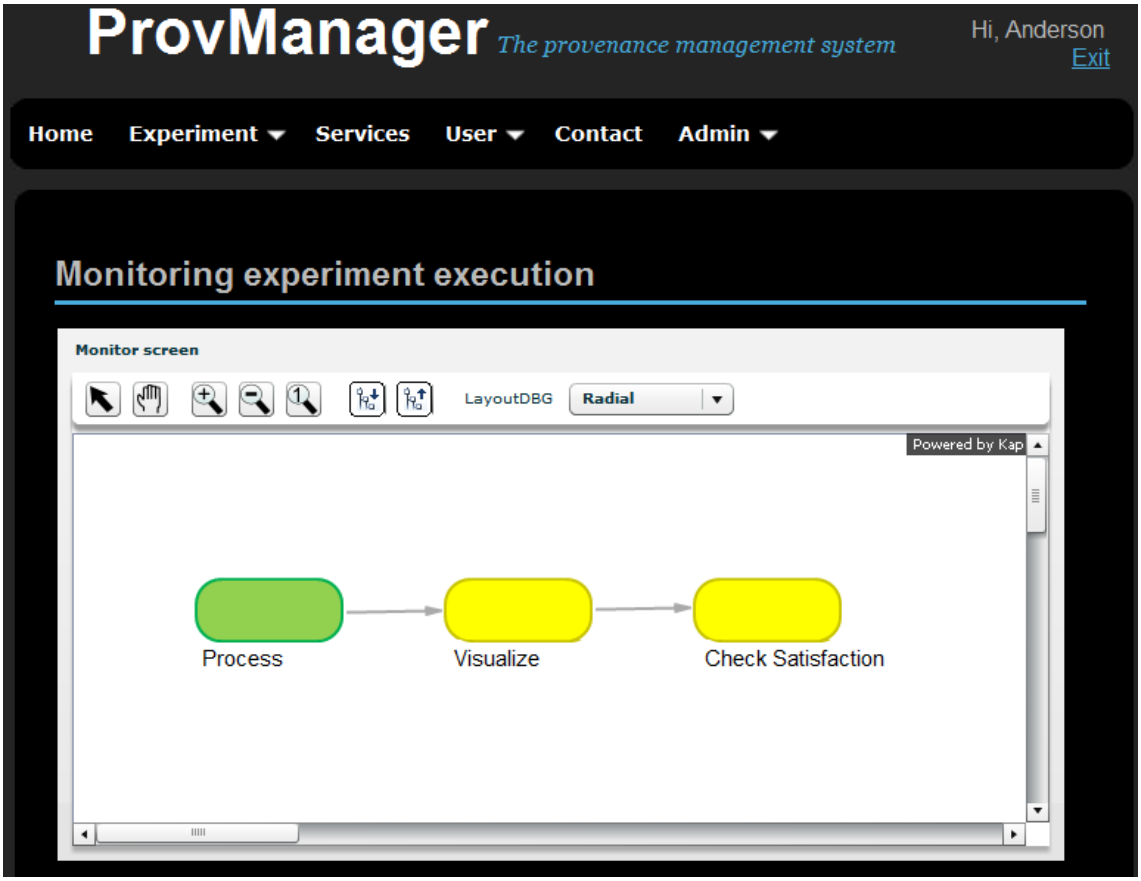

Figura 4.14 - Monitoramento do *workflow* "estágio3"

4.4.5 Consulta de dados de proveniência

Além do mecanismo de monitoramento, o protótipo possui um mecanismo básico de consulta de dados de proveniência. Este mecanismo utiliza *Prolog* como linguagem de consulta. A partir de uma interface gráfica (Figura 4.15), o cientista digita uma expressão de consulta no campo de texto do formulário, denominado *expressão*, e a resposta é retornada na área de texto abaixo, denominada *resultado*. Na Figura 4.15, por exemplo, é ilustrada uma consulta que lista todas as atividades do experimento de cristalografia simplificado.

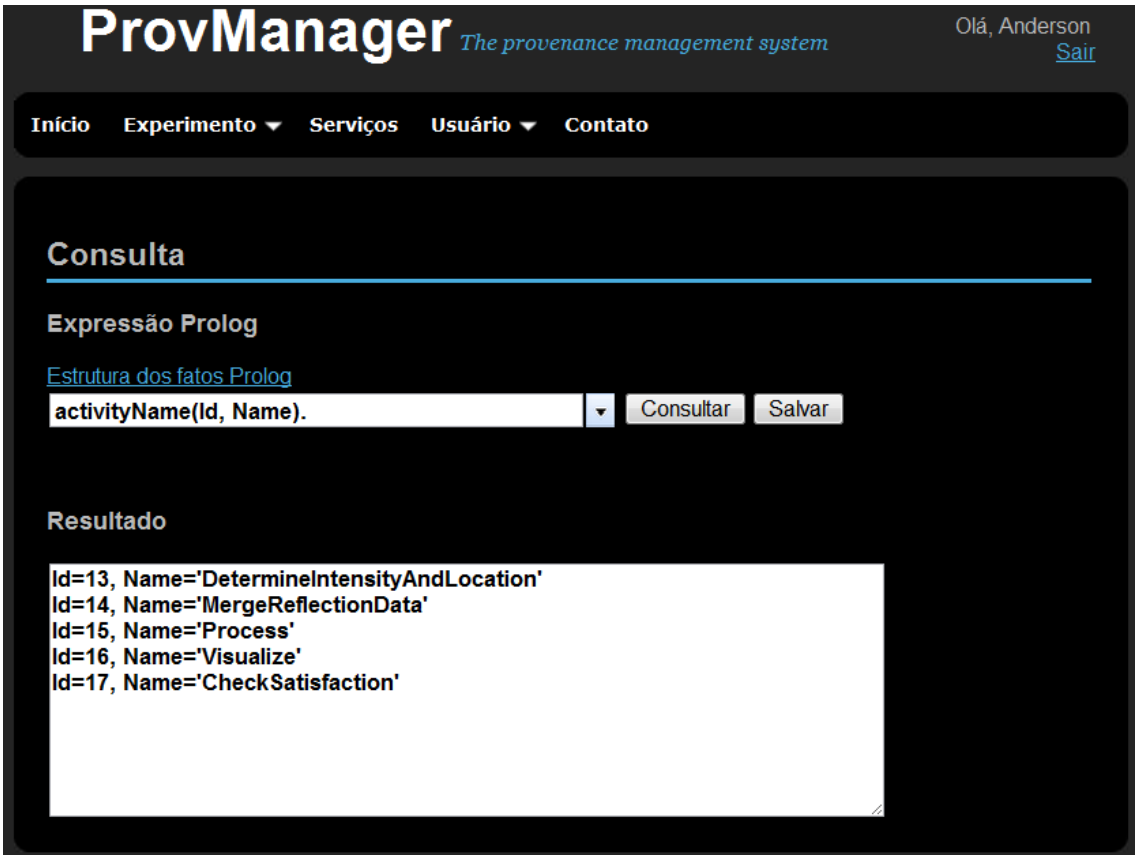

Figura 4.15 - Tela de consulta de proveniência

Além disso, para auxiliar a construção de expressões *Prolog*, o protótipo possui um mecanismo de sugestões que auxilia o usuário fornecendo modelos de consulta à medida que o usuário digita a expressão de consulta, de acordo com a Figura 4.16.a. As expressões de consultas criadas pelo usuário podem ser acrescentadas na lista de sugestões. Para acrescentar uma sugestão, o usuário deve digitar uma expressão e pressionar o botão *salvar*, de acordo com a Figura 4.16.b. A partir disso, uma janela é apresentada solicitando que o usuário digite uma descrição para a expressão que será

salva. Com esse mecanismo, torna-se possível a construção de expressões de consultas de alto nível que abstraiam informações de baixo nível do *ProvManager*, como também evita que usuários tenham que construir expressões *Prolog*, facilitando, principalmente, usuários que não possuem conhecimentos apurados nessa linguagem.

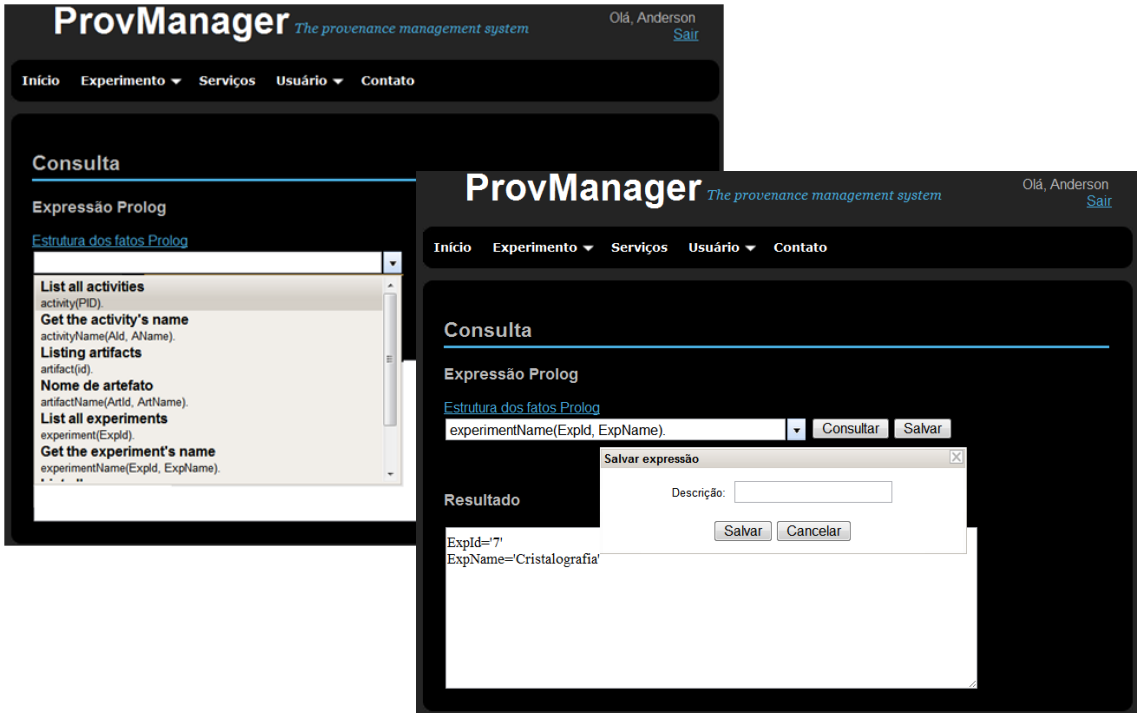

Figura 4.16 - Tela de consulta: (a) Sugestões de consultas pré-armazenadas; (b) Salvando expressão *Prolog* de consulta

4.5 *Considerações finais*

Este Capítulo apresentou um protótipo que implementa a abordagem *ProvManager*. Este protótipo mostrou que as soluções apresentadas no Capítulo 3 sobre a abordagem ProvManager são passíveis de implementação, abrindo a oportunidade para avaliações posteriores. Um ensaio de avaliação foi a utilização do protótipo em um experimento real no domínio de cristalografia que apresenta aspectos de execução em ambiente distribuído.

Embora este protótipo tenha contribuído para avaliação da abordagem *ProvManager*, é necessário uma avaliação mais controlada e sistematizada que apresente indícios que essa abordagem realmente confirma a hipótese de pesquisa deste trabalho, definida no Capítulo 1. Com isso, o Capítulo 5 apresentará essa avaliação realizada na abordagem *ProvManager*.

Capítulo 5. Avaliação

5.1 Introdução

Como foi definido no Capítulo 1, a questão de pesquisa desse trabalho é:

A promoção do gerenciamento de proveniência do nível de workflow para o nível do experimento torna mais produtiva a análise de proveniência por parte dos cientistas em ambientes distribuídos e heterogêneos?

Para responder esta questão de pesquisa, diversos aspectos precisam ser avaliados, dentre eles: aspectos de usabilidade, verificando se a utilização de um sistema de proveniência integrado é mais vantajosa do que usar diversos sistemas descentralizados; aspectos de desempenho, verificando se a estratégia de instrumentação do *workflow* para coletar os dados de proveniência influencia no tempo de execução do mesmo; aspectos sobre o modelo de proveniência da abordagem, avaliando se o conjunto de dados de proveniência coletados de um experimento pela abordagem *ProvManager* realmente auxilia o cientista no processo de análise de proveniência. Este capítulo apresenta um estudo avaliando apenas este último aspecto, considerado um dos mais importantes. Outros estudos analisando os demais aspectos são trabalhos futuros.

A partir disso, o protótipo descrito no Capítulo 4 foi utilizado na aplicação de um estudo preliminar de observação para avaliar a abordagem *ProvManager*. Foram observados 5 alunos de pós-graduação da *COPPE-UFRJ* utilizando duas estratégias de análise de proveniência (uma das estratégias consistia no uso do protótipo *ProvManager*) para responder questões de proveniência a respeito de um experimento distribuído hipotético. Apesar de restrito, o resultado deste estudo pôde ser utilizado para entender as limitações atuais tanto da abordagem quanto do protótipo, além de determinar os próximos passos para a utilização do *ProvManager* no contexto de experimentos reais.

Nas próximas seções deste capítulo, o estudo realizado é detalhado, descrevendo seus objetivos, discutindo seu planejamento e apresentando os resultados obtidos. A

Seção 5.2 apresenta os objetivos gerais do estudo. A Seção 5.3 define com mais detalhe o estudo realizado. A Seção 5.4 explica o procedimento de execução do estudo. A Seção 5.5 apresenta os resultados e as observações obtidas sobre a execução do estudo. A Seção 5.6 sumariza a avaliação feita pelos participantes sobre o estudo. Finalmente, a Seção 5.7 discute a validade do estudo.

5.2 Objetivo

Visando analisar se o conjunto de dados de proveniência coletados de um experimento pela abordagem *ProvManager* auxilia o cientista no processo de análise de proveniência do experimento, foi realizado um estudo de observação. Em estudos deste tipo, o participante realiza uma tarefa enquanto é observado por um experimentador. Este tipo de estudo tem a finalidade de coletar dados sobre como determinada tarefa é realizada (Shull et al. 2001). Por meio dessas informações, pode-se obter uma compreensão de como um novo processo é utilizado.

O objetivo deste estudo de observação pode ser apresentado de acordo com o modelo descrito por WOHLIN et al. (1999), ilustrado na Tabela 5.1.

Tabela 5.1 - Definição do objetivo do estudo de observação

Analisar o conjunto de dados de proveniência coletados de um experimento pelo *ProvManager*
Com o propósito de caracterizar a capacidade da abordagem *ProvManager* de responder questões de proveniência
Em relação à análise de proveniência de experimentos científicos distribuídos
Do ponto de vista do cientista
No contexto de alunos de pós-graduação, analisando as informações de proveniência de um experimento distribuído, com e sem o uso do protótipo *ProvManager*

5.3 Definição do Estudo

Um experimento hipotético, denominado *DistExp* (*Distributed Experiment*), foi elaborado para ser executado em um cenário de ambiente distribuído. Segundo este cenário, este experimento é fragmentado em dois *workflows*, instanciados em *SGWfC* distintos, a saber: *VisTrails* e *Kepler*. O *DistExp* é discutido com mais detalhe no Anexo C. O estudo consistiu em analisar o desempenho dos participantes, que assumiam o

papel de cientistas, no processo de análise de proveniência do experimento *DistExp* adotando cada uma das seguintes estratégias:

a) Utilizar em conjunto os mecanismos de proveniência existentes nos *SGWfC VisTrails* e *Kepler*;
b) Utilizar apenas o protótipo *ProvManager*.

O processo de análise de proveniência consiste em fornecer ao participante um conjunto de questões de proveniência referentes ao experimento *DistExp* para que sejam respondidas. Dentre essas questões, existe um subconjunto com questões de proveniência retrospectiva (*e.g.*, descrever as portas de entrada e saída de uma atividade) e um subconjunto com questões de proveniência retrospectiva (*e.g.*, descobrir o tempo de execução do experimento ou de uma atividade). Além disso, há questões de proveniência mais complexas que exploram a análise no experimento como um todo, não apenas em um *workflow*. Por exemplo, uma das questões solicita ao participante que descreva a sequência de transformações de um dado até o seu estado final. Essa questão só pode ser respondida analisando os dois *workflows* do experimento em conjunto. Estas questões de proveniência utilizadas no estudo estão presentes no Anexo D.

Para responder o conjunto de questões de proveniência usando a estratégia (a), o participante tinha que usar as funcionalidades de proveniência existentes nos sistemas *Kepler* e *VisTrails* e tirar as suas conclusões. Além disso, os participantes podiam utilizar outras funcionalidades dos *SGWfC* não específicas de proveniência para facilitar o processo de análise (*e.g.*, consultar a tela de edição/visualização de *workflow* do sistema para extrair informações). Por outro lado, usando a estratégia (b), o participante tinha que usar apenas o mecanismo de consulta de proveniência do *ProvManager* para responder as questões. Neste mecanismo, o participante tinha que saber construir consultas *Prolog* e interpretar os resultados das consultas que eram exibidos textualmente na interface do protótipo.

O objetivo principal do estudo era verificar se a abordagem *ProvManager* permite responder um conjunto maior de questões de proveniência do que a estratégia de usar uma combinação dos sistemas *Vistrails* e *Kepler*. Além disso, outros pontos secundários observados foram:

1. O uso de um único sistema integrado facilita a análise de proveniência?
2. A linguagem Prolog dificulta o processo de consulta de proveniência?

Este estudo foi realizado em um ambiente acadêmico, no Laboratório de Engenharia de Software (*LENS*) da *COPPE/UFRJ*. Cinco estudantes de pós-graduação da linha de Engenharia de Software da COPPE/UFRJ se voluntariaram para participar deste estudo. Nenhum destes estudantes possuía muita experiência com *workflows* científicos e, com isso, tiveram que receber treinamento apropriado. Além disso, os estudantes receberam treinamento específico em *Prolog* para saberem construír consultas de proveniência no *ProvManager*. Por fim, não houve compensação (monetária ou de qualquer outro tipo) para os participantes.

5.4 Procedimento de execução

Cada sessão do estudo utilizou um único participante e foi dividida em duas etapas. Uma etapa tinha como finalidade observar o comportamento do participante respondendo as questões de proveniência adotando a estratégia (a) de análise de proveniência (*i.e.*, utilizando em conjunto os mecanismos de proveniência existentes nos *SGWfC VisTrails* e *Kepler*). A outra etapa tinha como objetivo analisar o comportamento do participante respondendo as questões de proveniência usando a estratégia (b) de análise de proveniência (*i.e.*, utilizando apenas o protótipo *ProvManager*). Os participantes foram divididos em dois grupos, onde cada grupo, a ordem das etapas era diferente. Em um grupo, a primeira etapa da sessão é a que utiliza a estratégia (a), depois a etapa que utiliza a estratégia (b). Já no outro grupo, a ordem era a inversa. No total, cada sessão durou cerca de 70 minutos.

Inicialmente, cada participante foi informado sobre a avaliação através do Formulário de Consentimento (Anexo E). Caso concordasse em participar da avaliação, o participante preenchia o Questionário de Caracterização (Anexo G). Este questionário avalia o nível de conhecimento e experiência do participante em diferentes temas relacionados ao estudo. Esses dados foram utilizados para garantir que os participantes estavam aptos a executar o estudo e também para interpretar os resultados obtidos por cada um dos participantes. O preenchimento dos formulários iniciais levou cerca de 5 minutos.

Em seguida, supondo uma sessão em que a primeira etapa é aquela que utiliza a estratégia (a), com o início da primeira etapa da sessão, o participante recebeu um treinamento de aproximadamente 10 minutos sobre os principais conceitos de

experimentação científica. Além disso, o participante recebeu também um treinamento específico da etapa, de aproximadamente 10 minutos, sobre os *SGWfC Kepler e VisTrails*, aprendendo a manipular suas principais funcionalidades. Ao final do treinamento, foi entregue ao participante o documento de Descrição Geral da Tarefa (Anexo H), que descreve o contexto em que ele estaria inserido durante a execução das suas tarefas no estudo. Além disso, foi entregue o documento de descrição do experimento *DistExp* (Anexo C), bem como o documento de questionário de proveniência (Anexo D), que contém as questões de proveniência a serem respondidas pelo participante. Durante a execução das tarefas, o participante pôde utilizar apenas os sistemas *Kepler* e *VisTrails* para responder as perguntas, não podendo acessar outra fonte de informação.

Na segunda etapa, supondo uma sessão em que a segunda etapa é aquela que utiliza a estratégia (b), o participante recebeu outro treinamento, específico da etapa, de aproximadamente 5 minutos referente aos principais conceitos e funcionalidades do protótipo do *ProvManager*. O participante recebeu também um treinamento específico de Prolog de duração aproximada de 10 minutos para possibilitar que ele conseguisse construir consultas de proveniência *em Prolog* no *ProvManager*. Ao final do treinamento, foi entregue ao participante um novo questionário de perguntas de proveniência (Anexo D) que o participante teve que responder. Durante a execução das tarefas nesta etapa, o participante pôde utilizar apenas o protótipo *ProvManager* para responder as perguntas, não podendo consultar os sistemas *Kepler* e *VisTrails* onde o *DistExp* está instanciado. Para auxiliar na tarefa, o participante recebeu também uma lista (Anexo I), contendo uma breve descrição dos predicados *Prolog* do *ProvManager* necessários para responder o questionário de proveniência. A partir desta lista, o participante teve que analisar o(s) predicado(s) a ser(em) utilizado(s) para responder cada questão de proveniência. Vale destacar que na segunda etapa, é utilizado um novo questionário com algumas das questões de proveniência alteradas, além de se tratar de uma outra execução do experimento usando parâmetros diferentes, para que não possuam as mesmas respostas do primeiro questionário, facilitando, assim, a tarefa do participante. No Anexo D, existem duas versões do questionário de proveniência (Versão A e Versão B). Os gabaritos destes questionários podem ser conferidos no Anexo E.

Os participantes dispunham de meia hora para responder as questões de proveniência em cada etapa. Foram propostas, no total, 6 questões de proveniência. Os participantes deveriam tentar responder a maior quantidade possível de questões em cada etapa, não sendo obrigados a terminar todas. No final do tempo previsto para as tarefas, os participantes responderam a um Questionário de Avaliação (Anexo J), que diz respeito à experiência com as tarefas executadas e à utilização do protótipo *ProvManager*. O preenchimento deste questionário durou cerca de 15 minutos.

5.5 *Resultados e observações*

Participaram deste estudo, 5 alunos de pós-graduação em computação, sendo 3 de mestrado e 2 de doutorado. A Tabela 5.2 apresenta um resumo da caracterização desses participantes. Os níveis de experiência dos participantes em *workflow* científico eram bastante baixos, com exceção de um participante que já havia utilizado o sistema Kepler, mesmo assim não apresentava muita experiência, e de outro participante que já tinha lido algum material sobre o assunto. Com relação a *Prolog*, o grau de experiência dos participantes era, também, bastante baixo, tendo dois participantes apresentando nenhum conhecimento sobre o assunto. Isso foi interessante para simular o grau de conhecimento de cientistas de áreas não tecnológicas.

Tabela 5.2 - Resumo do questionário de caracterização dos participantes do estudo de observação

Critérios	P1	P2	P3	P4	P5
Formação acadêmica	Mestrando	Mestrando	Doutorando	Mestrando	Doutorando
Uso de algum *SGWfC*	*Kepler*	Nunca	Nunca	Nunca	Nunca
Experiência com *workflow* científico	Baixa	Muito baixa	Muito baixa	Muito baixa	Baixa
Participação em atividades que usam *workflows* científicos	Nunca	Nunca	Nunca	Nunca	Nunca
Leitura de publicações sobre *workflow* científico	Nunca	Nunca	Nunca	Sim	Nunca
Experiência com Prolog	Média	Baixa	Muito baixa	Muito baixa	Média

No geral, os participantes se comportaram de maneira semelhante durante o estudo. Todos tiveram mais dificuldade usando a estratégia (a) (*i.e.*, usando *Kepler* e *VisTrails* em conjunto) do que a estratégia (b) (*i.e.*, usando abordagem *ProvManager*). Utilizando a estratégia (a), os participantes tiveram êxito respondendo apenas às

questões de proveniência prospectiva e, de maneira incompleta, algumas das questões retrospectiva. As questões mais complexas para os participantes foram aquelas que necessitavam de informação em conjunto dos dois *SGWfC*. Por exemplo, o tempo de execução do experimento e o rastro de transformações de um artefato. No entanto, das questões que foram respondidas, os participantes as fizeram com bastante facilidade. A justificativa para isso é que as informações de proveniência eram extraídas de funcionalidades providas de recursos gráficos que permitiam o participante visualizar a estrutura do *workflow*. Contudo, vale destacar que estas funcionalidades usadas pelos participantes não eram especificas de proveniência, e sim relacionadas ao mecanismo de edição do *workflow*.

Utilizando a estratégia (b), os participantes tiveram êxito em responder todas as questões de proveniência, tanto as prospectiva quanto as retrospectiva. Diferentemente da estratégia (a), os participantes responderam com facilidade as questões que englobavam a análise em conjunto dos dois *workflows* do experimento. No entanto, os participantes apresentaram um pouco de dificuldade na construção das consultas *Prolog*. A construção das consultas era feita manualmente pelo participante, sem o auxílio, por exemplo, de mecanismos de auto-completar consultas ou mecanismos de sugestão de correção de consultas realizadas equivocadamente. Com isso, quando o participante cometia um erro por desatenção de digitação da consulta, ele achava que não estava sabendo manipular corretamente os predicados *Prolog* do *ProvManager*. Quando isso ocorria, o experimentador apontava o erro de digitação para o participante. Além desse problema, os usuários não sentiram outras dificuldades para responder as perguntas.

A Tabela 5.3 sumariza o resultado de cada participante na resolução dos questionários, de acordo com as estratégias utilizadas. Os participantes de Grupo 1 responderam os questionários utilizando primeiramente a estratégia (a) e, posteriormente, a estratégia (b). Os participantes do Grupo 2 responderam os questionários seguindo a ordem inversa, começando pela estratégia (b). Uma questão certa é representada pelo símbolo (✔) , enquanto que uma questão respondida de maneira errada, ou não respondida, é representada pelo símbolo (✘).

Tabela 5.3 - Resultado dos participantes na resolução dos questionários

Questionário			Grupo 1			Grupo 2	
			P1	P2	P3	P4	P5
VisTrails + Kepler	Prospectiva	Questão 1	✓	✓	✓	✓	✓
		Questão 2	✓	✓	✓	✓	✓
	Retrospectiva	Questão 3	✗	✗	✗	✗	✗
		Questão 4	✗	✗	✗	✗	✗
		Questão 5	✗	✗	✗	✗	✗
		Questão 6	✗	✗	✗	✗	✗
ProvManager	Prospectiva	Questão 1	✓	✓	✓	✓	✓
		Questão 2	✓	✓	✓	✓	✓
	Retrospectiva	Questão 3	✓	✓	✓	✓	✓
		Questão 4	✓	✓	✓	✓	✓
		Questão 5	✓	✓	✓	✓	✓
		Questão 6	✓	✓	✓	✓	✓

5.6 *Avaliação dos participantes*

Depois de concluir as duas etapas do estudo, os participantes preencheram um questionário de avaliação com perguntas referentes às duas estratégias de análise de proveniência. Com relação à análise de proveniência utilizando os sistemas *VisTrails* e *Kepler*, os participantes ficaram um pouco insatisfeitos por não terem conseguido responder todo o questionário de proveniência, além de sentirem dificuldades na análise descentralizada. Alguns comentários dos participantes foram:

- "Necessário conhecer especificidades de cada *SGWfC* para responder as questões."
- "Algumas informações foram facilmente acessadas graficamente, porém as outras não foram encontradas."
- "As informações de proveniência desses sistemas são pobres e confusas."

- “Difícil encontrar todas as informações desejadas; várias respostas foram ‘chutes’.”
- “As informações básicas de um *workflow* não estão facilmente acessíveis nas ferramentas.”

A respeito da avaliação da abordagem *ProvManager*, os participantes se mostraram mais satisfeitos. Embora tenham reclamado um pouco da forma de consulta textual por *Prolog*. Alguns comentários dos participantes foram:

- “A estratégia consegue suprir todas as informações necessárias, mas isso poderia ser ainda mais facilitado pelo uso de uma ferramenta visual.”
- “Obtém-se de forma relativamente simples e rápida as informações em questão.”
- “Agrupar as informações em uma única fonte é a melhor forma de acesso aos dados.”
- “Uma estratégia padronizada facilita a análise de proveniência em qualquer *workflow* de qualquer *SGWfC*.”

Por fim, os participantes sugeriram algumas melhorias para o mecanismo de consulta do protótipo *ProvManager*, a saber:

- Mecanismo de histórico de consultas já realizadas;
- Mecanismo de sugestão de correção de consultas que foram realizadas erradas;
- Mecanismo de auto-completar digitação de consultas;
- Trabalhar com consultas com parâmetros pré-fixados (*e.g.*, fixar o parâmetro de identificador do experimento caso todas as consultas estarão relacionadas a um único experimento);
- Mecanismo de geração de relatório com as principais informações de proveniência, disponibilizando recurso de análise estatística;
- Exibição mais elaborada das informações retornadas da consulta pelo sistema;
- Visualização dos dados de proveniência diretamente através do modelo do *workflow*.

5.7 Validade

Este estudo preliminar foi executado a fim de verificar se a abordagem proposta responde em parte a questão de pesquisa levantada nesta dissertação. Os resultados obtidos a partir das observações e da avaliação dos participantes nos ajudaram a entender as limitações da abordagem *ProvManager* e do protótipo implementado, assim como identificar pontos que precisam ser melhorados. No entanto, devido às restrições deste estudo, estes resultados não podem ser generalizados.

Durante a execução das duas etapas do estudo, o experimentador teve que prestar suporte aos participantes em situações quando os mesmos não conseguiam evoluir na tarefa. Por exemplo, quando os participantes queriam acessar determinada funcionalidade nos sistemas *VisTrails* e *Kepler*, mas não se lembravam como fazê-lo, ou durante a utilização do protótipo *ProvManager*, quando cometiam erros de digitação na construção das consultas e não percebiam o erro por um longo tempo. Estes fatores podem influenciar o resultado do estudo, mas isto é de certa forma compensado pelo pequeno tempo de treinamento que os participantes tiveram.

O planejamento de execução do estudo em duas etapas, onde cada etapa o participante responde perguntas de proveniência usando uma estratégia de análise de proveniência diferente, pode causar um viés de aprendizagem na execução da segunda etapa, favorecendo uma determinada estratégia. No entanto, isso foi amenizado com a variação da ordem das estratégias, ou seja, três participantes realizaram a primeira etapa usando a estratégia (a), seguido da estratégia (b), na segunda etapa. Os demais participantes (dois) realizaram a primeira etapa usando a estratégia (b), seguido da estratégia (a), na segunda etapa. Além disso, em cada etapa, foram utilizados questionários com respostas diferentes e execuções de experimentos com parâmetros distintos.

Os predicados de alto nível do *ProvManager* (Anexo J), fornecidos aos participantes para responder o questionário de proveniência, foram construídos exclusivamente para o estudo e apresentam algumas limitações. Os predicados possuem simplificações nas suas estruturas com o intuito de facilitar a tarefa do participante na construção das consultas, uma vez que o mesmo não dispõe de muito tempo para assimilar o conhecimento do domínio. Essa medida, consequentemente, pode causar certa influência no resultado do estudo. Entretanto, isso é amenizado pelo pequeno

tempo de treinamento fornecido aos participantes. Se o treinamento fosse mais longo, eles teriam mais habilidade para lidar com predicados mais complexos.

Devido a algumas limitações do protótipo como, por exemplo, a não possibilidade de estabelecer fluxo entre *workflows* de um experimento nos níveis de atividades e artefatos via interface gráfica, algumas configurações do experimento *DistExp*, utilizado nesse estudo, foram feitas diretamente no código do protótipo ("*hard-coded*"). Entretanto, esse acontecimento não influencia nos resultados do estudo com relação à abordagem *ProvManager*, uma vez que esses tipos de informações de proveniência são previstos no seu modelo de proveniência.

A escolha dos sistemas *Kepler* e *Vistrails* não foi feita de maneira aleatória, mas também não foi realizado algum tipo de estudo elaborado para definir quais sistemas seriam os mais apropriados. Esses dois sistemas foram escolhidos por serem bastante conhecidos e utilizados na comunidade científica. Com isso, existe a possibilidade de que o resultado do estudo seja influenciado pela escolha desses sistemas. Vale ressaltar que as últimas versões dos sistemas foram utilizadas no estudo (*VisTrails* 1.6.1 e *Kepler* 2.0.0), garantindo, assim, o melhor uso das suas funcionalidades.

Grande parte da seleção dos participantes foi feita por meio da solicitação de voluntários dentro de um grupo de alunos que compartilham um mesmo laboratório de pesquisa do qual também fazem parte o experimentador e os pesquisadores responsáveis pelo estudo. Essa medida foi necessária devido a restrições de tempo e de pessoal disponível. Como consequência, o grupo escolhido pode não ser representativo para a população que se deseja avaliar e pode ser influenciado pela sua relação com o experimentador. Outro fator que pode ser um complicador é a quantidade reduzida de participantes no estudo. Existe a possibilidade que os resultados sejam influenciados pelo tamanho e pelas características específicas do grupo. Além disso, o uso de alunos de pós-graduação em computação pode influenciar nos resultados, uma vez que estes possuem mais facilidade na manipulação dos mecanismos e na construção das consultas *Prolog*, diferentemente de cientistas de áreas não tecnológicas.

5.8 Considerações Finais

Neste capítulo, foi descrito o estudo de observação desenvolvido para avaliar preliminarmente a abordagem *ProvManager*. Este estudo buscou analisar a capacidade da abordagem *ProvManager* em responder questões de proveniência a respeito de um

experimento, com características de execução em ambiente distribuído, em comparação a abordagens tradicionais. Com isso, o estudo foi dividido em duas etapas, onde uma etapa o participante respondeu questões de proveniência utilizando abordagens tradicionais de proveniência providas pelos sistemas *Kepler* e *VisTrails*. Na outra etapa, o participante respondeu questões de proveniência utilizando a abordagem *ProvManager*.

A partir deste estudo foi possível ter indícios de que a abordagem *ProvManager* é uma alternativa mais robusta para auxiliar o cientista na análise de proveniência de experimento científico, comparado a alguns exemplos de *SGWfC*. Além disso, isso se potencializa na análise de proveniência de experimentos científicos executados em ambiente distribuído, onde é necessário extrair informações de mais de um sistema. Nesse cenário, como foi observado nesse estudo, se um desses sistemas não suporta determinada informação, a análise fica comprometida. Além disso, com relação aos pontos secundários analisados pelo estudo, foram obtidos os seguintes resultados:

1. O uso de um único sistema integrado facilita a análise de proveniência? Aparentemente sim. Os participantes se sentiram mais confortáveis em utilizar apenas um sistema para realizar a análise de proveniência, uma vez que não é necessário aprender diferentes funcionalidades e ter conhecimento de diferentes modelos de proveniência.
2. A linguagem *Prolog* dificulta o processo de consulta de proveniência? Aparentemente sim. Os participantes apresentaram um pouco de dificuldade na escrita da consulta *Prolog*, que talvez possa ser justificado pela falta de treinamento e pela ausência de mecanismos mais automatizados para dar suporte à construção das consultas. Entretanto, alguns participantes alegaram a necessidade da realização de consulta através de uma interface gráfica.

Esse estudo representa um dos estudos necessários para a avaliação da abordagem *ProvManager*, como foi destacado no início deste capítulo. Além disso, o estudo realizado possui limitações em diferentes aspectos, apontadas na Seção 5.7, restringindo a generalização dos resultados alcançados. No entanto, o estudo serviu para mostrar indícios de que a abordagem *ProvManager* é relevante para o problema abordado nesta pesquisa de dissertação.

Capítulo 6. Conclusão

6.1 *Epílogo*

A experimentação científica é cada vez mais uma técnica essencial em pesquisa para investigar fenômenos, tendo como objetivo adquirir novos conhecimentos ou corrigir e integrar conhecimentos previamente estabelecidos (Wilson 1991). Devido à sua importância, pesquisas que almejam obter credibilidade em seus resultados usualmente fazem uso de um processo de experimentação. Esta característica pode ser observada em especial para pesquisas de grande porte. Entretanto, nestes casos, realizar um experimento não é uma tarefa simples. Para auxiliar esses tipos de pesquisas, pode ser aplicada a experimentação *in-silico* (Travassos e Barros 2003) (Zhao et al. 2004), que visa solucionar os problemas relacionados a custos, tempo e recursos, dentre outros fatores, através da simulação da experimentação utilizando aplicativos computacionais. Estes aplicativos adotam modelos matemáticos para tentar reproduzir a realidade e podem ser compostos e modelados em *workflows* utilizando Sistemas Gerenciadores de *Workflow* Científicos (*SGWfC*). Com isso, a partir de simulações, experimentos que demorariam meses para serem executados e seriam altamente custosos passam a ser executados em poucas horas e por custos bem menores.

Embora esses experimentos *in-silico* tragam vantagens para o desenvolvimento de pesquisas de grande porte, novos desafios surgem. Um dos mais importantes é como controlar a grande quantidade de dados gerados por estes experimentos. O gerenciamento desses dados é importante, pois é a partir deles que é possível analisar os resultados e os comportamentos de um experimento. Nesse contexto, um elemento fundamental é o conceito de proveniência. Um dado gerado em um experimento não é algo isolado, existe uma série de informações associadas que definem a sua origem (*i.e.*, sua proveniência) que são essenciais para a análise de um experimento. Por exemplo, o conjunto de passos no experimento que culminou em um determinando resultado, quando esse resultado foi gerado, quem reproduziu o experimento, etc. O gerenciamento dos dados de proveniência de um experimento é realizado por alguns *SGWfC*, porém, como foi apresentado neste trabalho, estes não atendem todas as

necessidades. Esse problema se agrava quando experimentos são fragmentados em diversos *workflows* e executados em ambientes distribuídos. Nesse cenário, a gerência de proveniência é realizada por diversos sistemas de maneira descentralizada, dificultando ainda mais a tarefa do cientista na análise do experimento.

Visando minimizar esse problema, este trabalho apresentou uma abordagem chamada *ProvManager* que torna possível o gerenciamento de proveniência de *workflows* em ambientes distribuídos e heterogêneos. A abordagem *ProvManager* tem como propósito promover o gerenciamento de proveniência do nível de *workflow*, realizado pelos *SGWfC*, para o nível de experimento, por meio da modelagem, captura e armazenamento das informações de proveniência que estão localizadas de maneira descentralizada e não integrada em um ambiente distribuído.

6.2 Contribuições

Esta dissertação apresentou os resultados de um trabalho de pesquisa que teve como objetivo propor uma abordagem para gerenciamento de proveniência de *workflows* científicos. Entre as suas principais contribuições, podemos destacar:

1) Definição de uma abordagem, denominada *ProvManager*, para o gerenciamento de proveniência de experimentos executados em ambientes distribuídos, permitindo que os dados de proveniência de experimentos complexos que são executados em diversos *SGWfC*, ou máquinas de execução, sejam capturados, armazenados e acessados de forma integrada;

    1.1) Definição de uma estratégia de captura de proveniência independente de *SGWfC* que trabalha no nível de atividade. Além disso, esta estratégia possui algumas características adicionais que mitigam as deficiências dos mecanismos de captura que trabalham no nível de atividade: extração de informações de proveniência prospectiva através da especificação do *workflow*; e adaptação semiautomática das atividades do *workflow* para suportar o mecanismo de captura de proveniência retrospectiva;

    1.2) Definição de um modelo de proveniência baseado no *Open Provenance Model* (*OPM*) com algumas extensões. A principal delas é a representação de informações de proveniência prospectiva, uma vez

que o OPM só trabalha com proveniência retrospectiva. Além disso, *ProvManager* contribui ao fazer um mapeamento desses dois tipos de proveniência, permitindo que a partir de uma informação retrospectiva se obtenha informações prospectivas relacionadas, e vice-versa;

1.3) Definição de dois mecanismos de análise de proveniência. Um básico, a partir de consultas *Prolog*, e um avançado, a partir de agentes de proveniência, que utiliza a abordagem de agentes inteligentes para manipular a base de conhecimento da *Charon* realizando tarefas de análise de proveniência.

2) Desenvolvimento de um protótipo que implementa a abordagem *ProvManager*. Este protótipo foi implementado utilizando especificações e padrões abertos, amplamente aceitos. Um dos componentes principais do *ProvManager*, a máquina de processos *Charon*, foi adaptada e reestruturada para o contexto de experimentação científica.

3) Avaliação preliminar da abordagem *ProvManager* por meio de um estudo de observação que analisou a capacidade da abordagem em responder questões de proveniência de um experimento fictício, executado em um ambiente distribuído e heterogêneo. Os resultados foram positivos, indicando que o uso da abordagem *ProvManager* propicia uma análise de proveniência mais completa do que o uso em conjunto de mecanismos de proveniência de *SGWfC*. Além disso, os resultados obtidos foram utilizados para definir os próximos passos de melhoria da abordagem e podem servir como base para o planejamento de uma avaliação mais completa.

6.3 Limitações

Durante a pesquisa realizada, alguns imprevistos limitaram algumas das características da abordagem. Além disso, outras limitações foram identificadas a partir de uma análise crítica da abordagem e do protótipo desenvolvido. Dentre essas limitações, as principais são:

- O gerenciamento de proveniência de experimentos executados em ambientes mais complexos de distribuição (cluster, nuvem computacional) não foram

tratados pela abordagem. Com isso, informações de proveniência específicas desses ambientes não são capturadas pela abordagem ProvManager.

- A definição do relacionamento entre *workflows* de um experimento não é feita no nível de atividade, apenas no nível de *workflow*, através da funcionalidade *dependências de workflow* do *ProvManager*. Com essa funcionalidade, por exemplo, não é possível informar que uma atividade "A1" de um *workflow* "Wf1" inicializa a execução de uma atividade "A2" pertencente a um *workflow* "Wf2". Além disso, não é possível definir os dados que são trocados entre *workflows* distintos. Uma solução seria utilizar uma interface gráfica, aos moldes do mecanismo de monitoramento, que apresentasse os *workflows* do experimento e o usuário pudesse configurar as conexões entre atividades de *workflows* diferentes.
- O funcionamento do mecanismo de captura de proveniência depende de como as informações estão explicitadas no *workflow*. Por exemplo, se uma atividade do *workflow* gera um arquivo no sistema e o cientista não explicita essa operação na especificação do *workflow* como um produto gerado pela atividade (*e.g.*, fornecendo o endereço do arquivo), o mecanismo de captura do *ProvManager* falha em capturar essa informação. Uma solução para esse problema é criar um módulo específico no mecanismo de captura que trabalhe no nível de *SO*, monitorando operações adicionais que ocorrem no sistema (nesse caso, geração de arquivos) que não foram explicitadas pelo cientista no *workflow*;
- No mecanismo de versionamento de experimentos implementado no *ProvManager*, o processo de geração de nova versão de experimento é limitado, pois a nova versão gerada não herda as configurações da última versão do experimento. A cada nova versão gerada, o usuário deverá realizar todo o processo de cadastro dos *workflows* associados ao experimento novamente;
- O mecanismo de consulta desenvolvido no *ProvManager* é básico e apresenta algumas deficiências. As primeiras delas é com relação à manipulação direta da linguagem *Prolog* pelo usuário para fazer consultas. Esse problema é de certa forma atenuado com o uso do mecanismo de

sugestão de consultas. Além disso, o resultado é apresentado em forma de texto, seguindo o padrão *Prolog* que não é muito intuitivo para o usuário que não tem muito conhecimento nessa linguagem. No entanto, existe uma pesquisa de mestrado em andamento que visa solucionar esses problemas. Um dos objetivos do trabalho é aplicar técnicas de *elastic list* (Stefaner e Muller 2007), uma espécie de busca facetada[1] avançada que utiliza mecanismos de pesos de relações, em consultas em proveniência;

- O mecanismo de monitoramento apresenta poucas informações durante a execução do *workflow*, se limitando a apenas mostrar os estados de execução das atividades do *workflow*. Este mecanismo poderia apresentar informações adicionais sobre as atividades como, por exemplo, o tempo de início e término de execução, os dados gerados e consumidos através de *links*, o número de execuções de cada atividade, etc;
- A avaliação da abordagem *ProvManager* foi limitada, sendo restrita a apenas um estudo de observação que avalia o modelo de proveniência do *ProvManager*. Como foi discutido no Capítulo 5, existe a necessidade de que, no mínimo, sejam feitos alguns estudos adicionais para avaliar aspectos de usabilidade e desempenho.

6.4 Trabalhos futuros

Este trabalho foi um primeiro passo para o desenvolvimento de um sistema integrado de gerenciamento de proveniência de *workflows* científicos. Nesta primeira etapa, a preocupação principal foi com a concepção e o desenvolvimento da infraestrutura do sistema, resolvendo questões chave como, por exemplo, a definição de uma estratégia de captura de proveniência em um ambiente distribuído, como armazenar esses dados capturados e como disponibilizá-los para o cientista. Com isso, este trabalho abriu novas oportunidades que podem ser tratadas em diversos trabalhos futuros. Alguns desses trabalhos futuros são:

- Estudo mais aprofundado em como capturar dados de proveniência em

[1] Busca facetada é uma técnica para acessar um conjunto de informação a partir da seleção de filtros (facetas) que classificam a informação (Ben-Yitzhak et al. 2008).

ambientes distribuídos mais complexos que envolvam execução de tarefas computacionais em clusters e nuvens computacionais. Nesses cenários, novas informações de proveniência mais complexas de serem coletadas surgem e, com isso, novas estratégias de captura precisam ser elaboradas;

- Implementação dos demais agentes de proveniência definidos em alto nível na abordagem *ProvManager*, além do agente de monitoramento que foi implementado no protótipo;
- Extensão do modelo de proveniência para suportar o mapeamento de informações de nível conceitual do *workflow* do experimento, além do nível concreto (ou seja, com características de implementação oriundas dos *SGWfC* onde foram desenvolvidos) atualmente suportado. Além disso, o modelo de proveniência pode ser estendido para suportar informações que vão além do experimento, contemplando informações organizacionais da entidade desenvolvedora do experimento, como é proposto por Zhao (2007);
- Desenvolvimento de uma estratégia de armazenamento de dados de proveniência descentralizada, favorecendo experimentos que trabalham com grandes quantidades de dados, inviáveis de serem trafegados pela rede. Com essa estratégia, os dados de proveniência seriam gerenciados localmente e apenas a referência destes dados seria publicada no *ProvManager*. Estratégia semelhante é proposta por Groth *et al.* (2006);
- Automatização do processo de publicação de *workflows* através da integração do *SGWfC* com o *ProvManager*. A partir dessa integração, todas as etapas do processo de publicação de *workflow* (*e.g.*, submeter especificação do *workflow* no *ProvManager*, obter especificação adaptada, carregar especificação adaptada no *SGWfC*, etc.) seriam feitas de forma automática e transparente para o usuário pelo *SGWfC*. A publicação do *workflow* seria acionada de forma transparente, por exemplo, toda vez que o usuário solicitasse uma execução do *workflow* no *SGWfC*;
- Realização de novas avaliações para obter indícios mais sólidos de que a abordagem *ProvManager* contribui para solucionar o problema estudado nessa pesquisa de dissertação.

Referências Bibliográficas

Adobe, (2010a). Adobe Flash. Disponível em: http://www.adobe.com/br/flashplatform/.

Adobe, (2010b). Adobe Flex. Disponível em: http://www.adobe.com/br/products/flex/.

Adobe, (2010c). Adobe BlazeDS. Disponível em: http://www.adobe.com/aboutadobe/pressroom/pressreleases/200712/121307BlazeDS.html.

Altintas, I., Barney, O., Jaeger-Frank, E., (2006), "Provenance Collection Support in the Kepler Scientific Workflow System". In: *Proc. of the International Provenance and Annotation Workshop, IPAW*, p. 118-132, Chicago, USA.

Anand, M., Bowers, S., Altintas, I., Ludaescher, B., (2010), "Approaches for Exploring and Querying Scientific Workflow Provenance Graphs". In: *Proc. of the International Provenance and Annotation Workshop, IPAW*, p. 17-26, New York, USA.

Apache, (2010). Apache Axis2. Disponível em: http://ws.apache.org/axis2/.

Araujo, I., (2010), *Adaptador VisTrails – Um mecanismo de adaptação extensível voltado para a especificação do SGWfC VisTrails*. B.Sc., UFRJ/NCE, RJ, Brasil.

Barga, R. S., Digiampietri, L. A., (2008), "Automatic capture and efficient storage of e-Science experiment provenance", *Concurr. Comput. : Pract. Exper.*, v. 20, n. 5, p. 419-429.

Ben-Yitzhak, O., Golbandi, N., Har'El, N., Lempel, R., Neumann, A., Ofek-Koifman, S., Sheinwald, D., Shekita, E., Sznajder, B., et al., (2008), "Beyond basic faceted search". In: *Proceedings of the international conference on Web search and web data mining*, p. 33–44, New York, NY, USA.

Bose, R., Frew, J., (2005), "Lineage retrieval for scientific data processing: a survey", *ACM Comput. Surv.*, v. 37, n. 1, p. 1-28.

Bowers, S., Mcphillips, T., Ludäscher, B., Cohen, S., Davidson, S. B., (2006), "A model for user-oriented data provenance in pipelined scientific workflows". In: *In IPAW*, p. 133-147, Chicago, USA.

Buyya, R., (1999), *High Performance Cluster Computing: Architectures and Systems, Vol. 1*. 1 ed. Prentice Hall.

Callahan, S. P., Freire, J., Santos, E., Scheidegger, C. E., Silva, C. T., Vo, H. T., (2006), "VisTrails: visualization meets data management". In: *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, p. 745-747, Chicago, USA.

Ceri, S., Gottlob, G., Tanca, L., (1989), "What You Always Wanted to Know About Datalog (And Never Dared to Ask)", *IEEE Transactions on Knowledge and Data Engineering*, v. 1, n. 1, p. 146-166.

Cohen, S., Cohen-Boulakia, S., Davidson, S., (2006), "Towards a Model of Provenance and User Views in Scientific Workflows", *Data Integration in the Life Sciences*, Springer, p. 264-279.

Cohen-Boulakia, S., Biton, O., Cohen, S., Davidson, S., (2008), "Addressing the provenance challenge using ZOOM", *Concurr. Comput. : Pract. Exper.*, v. 20, n. 5, p. 497-506.

Couvares, P., Kosar, T., Roy, A., Weber, J., Wenger, K., (2007), "Workflow

Management in Condor", *Workflows for E-science: Scientific Workflows for Grids, Springer Verlag*, p. 357-375.

Cruz, S. M. S. D., Barros, P. M., Bisch, P. M., Campos, M. L. M., Mattoso, M., (2008), "Provenance Services for Distributed Workflows". In: *Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, p. 526-533, Lyon, France.

Da Cruz, S. M. S., Campos, M. L. M., Mattoso, M., (2009), "Towards a Taxonomy of Provenance in Scientific Workflow Management Systems". In: *IEEE 2009 Third International Workshop on Scientific Workflows (SWF 2009)*, p. 259-266, Los Angeles, USA.

Da Cruz, S. M. S., Chirigati, F. S., Dahis, R., Campos, M. L. M., Mattoso, M., (2008), "Using explicit control processes in distributed workflows to gather provenance". In: *IPAW*, p. 186-199, Utah, USA.

Dong, G., Libkin, L., Su, J., Wong, L., (1999), "Maintaining transitive closure of graphs in SQL", *Int. J. Information Technology*, v. 5, p. 46–78.

Drenth, J., (1999), *Principles of protein X-ray crystallography*. Springer Verlag.

Estublier, J., (2000), "Software configuration management: a roadmap". In: *Proceedings of the Conference on The Future of Software Engineering*, p. 279-289, Limerick, Ireland.

Eugster, P. T., Felber, P. A., Guerraoui, R., Kermarrec, A., (2003), "The many faces of publish/subscribe", *ACM Comput. Surv.*, v. 35, n. 2, p. 114-131.

Finlay, P. N., Mitchell, A. C., (1994), "Perceptions of the benefits from the introduction of CASE: an empirical study", *MIS Q.*, v. 18, n. 4, p. 353-370.

Foster, I., Vöckler, J., Wilde, M., Zhao, Y., (2002), "Chimera: A Virtual Data System For Representing, Querying, and Automating Data Derivation". In: *In Proceedings of the 14th Conference on Scientific and Statistical Database Management*, p. 37-46, Heidelberg, Germany.

Freire, J., Koop, D., Santos, E., Silva, C., (2008), "Provenance for Computational Tasks: A Survey", *Computing in Science & Engineering*, v. 10, n. 3, p. 11-21.

Frey, J., Tannenbaum, T., Livny, M., Foster, I., Tuecke, S., (2002), "Condor-G: A Computation Management Agent for Multi-Institutional Grids", *Cluster Computing*, v. 5, n. 3, p. 237-246.

Ghiya, R., Hendren, L. J., (1996), "Is it a tree, a DAG, or a cyclic graph? A shape analysis for heap-directed pointers in C". In: *Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, p. 1-15, St. Petersburg Beach, United States.

Gil, Y., (2007), "Workflow Composition: Semantic Representations for Flexible Automation", *Workflows for E-science: Scientific Workflows for Grids, Springer Verlag*, p. 244-257.

Goderis, A., Sattler, U., Lord, P., Goble, C., (2005), "Seven Bottlenecks to Workflow Reuse and Repurposing". In: *Lecture Notes in Computer ScienceProc. of the 4th International Semantic Web Conference*, p. 323–337, Galway, Ireland.

Gonzalez, A. J., Dankel, D. D., (1993), *The Engineering of Knowledge-Based Systems*. Prentice Hall.

Gorla, N., (2003), "Features to consider in a data warehousing system", *Communications of the ACM*, v. 46, n. 11, p. 111-115.

Groth, P., Jiang, S., Miles, S., Munroe, S., Tan, V., Tsasakou, S., Moreau, L., (2006), *An Architecture for Provenance Systems*, Technical report, Electronics and Computer Science, University of Southampton.

Groth, P., Munroe, S., Miles, S., Moreau, L., (2008), "Applying the Provenance Data

Model to a Bioinformatics Case", *High Performance Computing and Grids in Action*, v. 16, p. 250-264.

Holdener III, A., (2008), *Ajax: the definitive guide*. O'Reilly.

Hollingsworth, D., (1995), "Workflow Management Coalition: The Workflow Reference Model", *The Workflow Management Coalition*

Huang, Y., Slominski, A., Herath, C., Gannon, D., (2006), "WS-Messenger: A Web Services-Based Messaging System for Service-Oriented Grid Computing". In: *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, p. 166-173, Singapore.

IPAW, (2010), "International Provenance and Annotation Workshop Series, http://www.ipaw.info/".

Jboss, (2010a). RichFaces - Rich AJAX enabled components for your JSF applications. Disponível em: http://www.jboss.org/richfaces.

Jboss, (2010b). Jboss Community. Disponível em: http://community.jboss.org/.

Jboss, (2010c). Hibernate. Disponível em: http://www.hibernate.org/.

Jennings, N., (2000), "On agent-based software engineering", *Artificial Intelligence*, v. 117, n. 2, p. 277-296.

Jia, J., Sun, J., Tang, C., Shum, H., (2006), "Drag-and-drop pasting". In: *ACM SIGGRAPH 2006 Papers on - SIGGRAPH '06ACM SIGGRAPH 2006 Papers*, p. 631, Boston, USA.

Kap IT, (2010). Kap Lab Visualizer. Disponível em: http://lab.kapit.fr/display/kaplabhome/Home.

Kowalczykiewicz, K., Weiss, D., (2002), "Traceability: Taming uncontrolled change in software development". In: *National Software Engineering Conference*, p. 33–42, Tarnowo Podgorne, Poland.

Krasner, G. E., Pope, S. T., (1988), "A cookbook for using the model-view controller user interface paradigm in Smalltalk-80", *J. Object Oriented Program.*, v. 1, n. 3, p. 26-49.

Lin, C., Lu, S., Lai, Z., Chebotko, A., Fei, X., Hua, J., Fotouhi, F., (2008), "Service-Oriented Architecture for VIEW: A Visual Scientific Workflow Management System". In: *Services Computing 2008 (SCC '08)*, p. 335-342, Honolulu, USA.

Lopes, L., Murta, L., Werner, C., (2006), "Odyssey-CCS: A Change Control System Tailored to Software Reuse". In: *Proceedings of the 9th International Conference on Software Reuse*, p. 170-183, Torino, Italy.

Ludascher, B., Altintas, I., Berkley, C., Higgins, D., Jaeger, E., Jones, M., Lee, E. A., Tao, J., Zhao, Y., (2006), "Scientific workflow management and the Kepler system", *Concurrency and Computation*, v. 18, n. 10, p. 1039-1065.

Mattoso, M., Werner, C., Travassos, G., Braganholo, V., Murta, L., Ogasawara, E., Oliveira, F., Martinho, W., (2009), "Desafios no apoio à composição de experimentos científicos em larga escala". In: *Seminário Integrado de Software e Hardware (SEMISH)*, p. 307-321, Bento Gonçalves, Brasil.

Mattoso, M., Werner, C., Travassos, G., Braganholo, V., Murta, L., (2008), "Gerenciando Experimentos Científicos em Larga Escala". In: *Seminário Integrado de Software e Hardware (SEMISH)*, p. 121-135, Belém, Brasil.

Miles, S., Deelman, E., Groth, P., Vahi, K., Mehta, G., Moreau, L., (2007a), "Connecting Scientific Data to Scientific Experiments with Provenance". In: *Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, p. 179-186, Bangalore, India.

Miles, S., Groth, P., Branco, M., Moreau, L., (2007b), "The Requirements of Using Provenance in e-Science Experiments", *Journal of Grid Computing*, v. 5, n. 1

(Mar.), p. 1-25.
Moreau, L., Freire, J., Myers, J., Futrelle, J., Paulson, P., (2007), *The Open Provenance Model*, Technical report, Electronics and Computer Science, University of Southampton.
Moreau, L., Groth, P., Miles, S., Vazquez-Salceda, J., Ibbotson, J., Jiang, S., Munroe, S., Rana, O., Schreiber, A., et al., (2008), "The provenance of electronic data", *Commun. ACM*, v. 51, n. 4, p. 52-58.
Munroe, S., Miles, S., Moreau, L., Vázquez-Salceda, J., (2006), "PrIMe: a software engineering methodology for developing provenance-aware applications". In: *Proceedings of the 6th international workshop on Software engineering and middleware*, p. 39-46, Portland, USA.
Murta, L. G. P., (2002), *Charon: Uma máquina de processos extensível baseada em agentes inteligentes*. M.Sc., UFRJ/COPPE, RJ, Brasil
MySQL, (2010). MySQL Server. Disponível em: http://www.mysql.com/.
Ogasawara, E., Rangel, P., Murta, L., Werner, C., Mattoso, M., (2009), "Comparison and versioning of scientific workflows". In: *Proceedings of the 2009 ICSE Workshop on Comparison and Versioning of Software Models*, p. 25-30, Vancouver, Canada.
Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M. R., et al., (2004), *Taverna: a tool for the composition and enactment of bioinformatics workflows*. Oxford Univ Press.
Oinn, T., Li, P., Kell, D. B., Goble, C., Goderis, A., Greenwood, M., Hull, D., Stevens, R., Turi, D., et al., (2007), "Taverna/myGrid: aligning a workflow system with the life sciences community", *Workflows for E-science: Scientific Workflows for Grids, Springer Verlag*, p. 300–319.
OMG, (2010), *Unified Modeling Language Specification (UML) Version 2.3*.
Oracle, (2010a). Java Platform, Enterprise Edition. Disponível em: http://www.oracle.com/technetwork/java/javaee/overview/index.html.
Oracle, (2010b). JavaServer Faces Technology. Disponível em: http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html.
Palmieri, M., Portugal, I., (2010), *Adaptador Kepler – Um mecanismo de adaptação extensível voltado para a especificação do SGWfC Kepler*. B.Sc., UFRJ/NCE, RJ, Brasil.
ProvChallenge, (2011), "Fourth Provenance Challenge, http://twiki.ipaw.info/bin/view/Challenge/FourthProvenanceChallenge".
Qin, Z., Xing, J., Zheng, X., (2008), *Software architecture*. Springer.
Russell, S. J., Norvig, P., (1995), *Artificial Intelligence: Modern Approach*. 1 ed. Prentice Hall.
Settimi, R., Cleland-Huang, J., Khadra, O. B., Mody, J., Lukasik, W., DePalma, C., (2004), "Supporting Software Evolution through Dynamically Retrieving Traces to UML Artifacts". In: *Proceedings of the Principles of Software Evolution, 7th International Workshop*, p. 49-54, Kyoto, Japan.
Shinskey, F. G., (1979), *Process Control Systems*. McGraw-Hill, Inc.
Shull, F., Carver, J., Travassos, G. H., (2001), "An empirical methodology for introducing software processes". In: *Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering*, p. 288–296, Vienna, Austria.
Simmhan, Y. L., Plale, B., Gannon, D., (2005), *A Survey of Data Provenance Techniques*, Technical report, Computer Science Department, Indiana

University.
Simmhan, Y. L., Plale, B., Gannon, D., (2006), "A Framework for Collecting Provenance in Data-Centric Scientific Workflows". In: *International Conference on Web Services*, p. 427-436, Chicago, USA.
Simpson, J., Weiner, E., (1989), *The Oxford English Dictionary (20 Volume Set)*. 2 ed. Oxford University Press, USA.
Stefaner, M., Muller, B., (2007), "Elastic lists for facet browsers". In: *Database and Expert Systems Applications, 2007. DEXA'07. 18th International Conference on*, p. 217–221, Regensburg, Germany.
Stevens, R., Glover, K., Greenhalgh, C., Jennings, C., Pearce, S., Li, P., Radenkovic, M., Wipat, A., Tower, C., et al., (2003a), "Performing in silico experiments on the Grid: a users perspective". In: *Proc UK e-Science programme All Hands Conference*, p. 2-4, Nottingham, UK.
Stevens, R. D., Robinson, A. J., Goble, C. A., (2003b), "myGrid: personalised bioinformatics on the information grid", *Bioinformatics*, v. 19, n. suppl_1 (Jul.), p. i302-304.
Stonebraker, M., (1975), "Implementation of integrity constraints and views by query modification". In: *Proceedings of the 1975 ACM SIGMOD international conference on Management of data*, p. 65-78, San Jose, California.
Taylor, I. J., Deelman, E., Gannon, D. B., Shields, M., (2006), *Workflows for e-Science: Scientific Workflows for Grids*. 1 ed. Springer.
Travassos, G., Barros, M., (2003), "Contributions of in virtuo and in silico experiments for the future of empirical studies in software engineering". In: *Workshop on Empirical Software Engineering: The Future of Empirical Studies in Software Engineering*, p. 117-130, Fraunhofer IRB Verlag, Roman Castles, Italy.
W3C, (1999), *XML Path Language (XPath) Version 1.0*.
W3C, (2007), *Web Ontology Language (OWL)*.
W3C, (2008a), *Extensible Markup Language (XML) 1.0 (Fifth Edition)*.
W3C, (2008b), *SPARQL Query Language for RDF*.
Weiss, A., (2007), "Computing in the clouds", *netWorker*, v. 11, n. 4, p. 16-25.
Werner, C., Mattoso, M., Braga, R., Barros, M., Murta, L., Dantas, A., (1999), "Odyssey: Infra-estrutura de Reutilização baseada em Modelos de Domínio". In: *XIII Simpósio Brasileiro de Engenharia de Software, Caderno de Ferramentas*, Florianópolis, Brasil.
WfMC, (2011). Workflow Management Coalition Terminology & Glossary, disponível em http://www.wfmc.org/.
Wilson, E. B., (1991), *An Introduction to Scientific Research*. Dover Publications.
Wohlin, C., Runeson, P., Höst, M., (1999), *Experimentation in Software Engineering: An Introduction*. 1 ed. Springer.
Yu, J., Buyya, R., (2005), "A taxonomy of scientific workflow systems for grid computing", *SIGMOD Rec.*, v. 34, n. 3, p. 44-49.
Zhao, J., (2007), *A conceptual model for e-science provenance*. D.Sc., School of Computer Science, University of Manchester, U.K.
Zhao, J., Goble, C., Stevens, R., Bechhofer, S., (2004), "Semantically linking and browsing provenance logs for e-science", *Semantics of a Networked World*, v. 3226, p. 158–176.
Zhao, Y., Hategan, M., Clifford, B., Foster, I., von Laszewski, G., Nefedova, et al., (2007), "Swift: Fast, Reliable, Loosely Coupled Parallel Computation". In: *2007 IEEE Congress on Services*, p. 206, 199, Hawaii, USA.

Anexo A - API de Serviços de publicação de proveniência Prospectiva

createExperiment

```
String createExperiment(String name)
```

Descrição:

Cria um experimento.

Parâmetros:

`name` - Nome do experimento.

Retorna:

Identificador do experimento (experimentId).

---

createExperimentNewVersion

```
String createExperimentNewVersion(String experimentId)
```

Descrição:

Cria uma nova versão do experimento.

Parâmetros:

`experimentId` - Identificador do experimento.

Retorna:

Identificador da nova versão do experimento gerada (experimentVersionId).

---

setExperimentRootCompositeActivity

```
String setExperimentRootCompositeActivity (String experimentId,
                                  String compositeActivityInstanceId)
```

Descrição:

Indica qual atividade composta do experimento é raiz.

Parâmetros:

experimentId - Identificador do experimento.

compositeActivityInstanceId - Identificador da instância da atividade composta que vai ser associada como raiz.

Retorna:

Resposta de sucesso da operação (true|false).

---

registerSWfMS

```
String registerSWfMS(String name, String host)
```

Descrição:

Registra um Sistema Gerenciador de Workflow Científico (SGWfC).

Parâmetros:

name - Nome do SGWfC.

host - Máquina onde o SGWfC está instalado.

Retorna:

Identificador do SGWfC (SWfMSId).

---

associateCompositeActivityToSWFMS

```
String associateCompositeActivityToSWFMS(
                    String compositeActivityInstanceId, String SWfMSId)
```

Descrição:

Associa uma atividade composta a um SGWfC.

Parâmetros:

compositeActivityInstanceId - Identificador da instância da atividade composta.

SWFMSId - Identificador do SGWfC.

Retorna:

Resposta de sucesso da operação (true|false).

---

defineCompositeActivity

```
String defineCompositeActivity (String type, String name)
```

Descrição:

Define uma atividade composta.

Parâmetros:

`type` - Tipo da atividade composta (e.g., local, remota).

`name` - Nome da atividade composta.

Retorna:

Identificador da atividade composta (compositeActivityId).

---

defineActivity

```
String defineActivity(String type, String name)
```

Descrição:

Define uma atividade simples.

Parâmetros:

`type` - Tipo da atividade simples (e.g., local, remota).

`name` - Nome da atividade simples.

Retorna:

Identificador da atividade simples (activityId).

---

defineArtifact

```
String defineArtifact()
```

Descrição:

Cria um artefato que representa um dado gerado ou consumido por uma atividade em um workflow.

Retorna:

Identificador do artefato (artifactId).

---

defineInPort

```
String defineInPort(String portName, String portDataType)
```

Descrição:

Cria uma porta de entrada que vai ser associado a uma atividade simples ou composta.

Parâmetros:

`portName` - Nome da porta.

`portDataType` - Tipo de dado suportado pela porta (e.g., String, short, int, long, float, double, boolean, byte).

Retorna:

Identificador da porta (portId).

---

defineOutPort

```
String defineOutPort(String portName, String portDataType)
```

Descrição:

Cria uma porta de saída que vai ser associado a uma atividade simples ou composta.

Parâmetros:

`portName` - Nome da porta.

`portDataType` - Tipo de dado suportado pela porta (e.g., String, short, int, long, float, double, boolean, byte).

Retorna:

Identificador da porta (portId).

---

addActivityPort

```
String addActivityPort(String activityId, String portId)
```

Descrição:

Associa uma porta de entrada ou de saída a uma atividade simples.

Parâmetros:

`activityId` - Identificador da atividade simples.

`portId` - Identificador da porta.

Retorna:

Resposta de sucesso da operação (true|false).

---

addCompositeActivityPort

```
String addCompositeActivityPort(String compositeActivityId,
                                String portId)
```

Descrição:

Associa uma porta de entrada ou de saída a uma atividade composta.

Parâmetros:

`compositeActivityId` - Identificador da atividade composta.

`portId` - Identificador da porta.

Retorna:

Resposta de sucesso da operação (true|false).

---

associateArtifactToActivityPort

```
String associateArtifactToActivityPort(String activityInstanceId,
                                       String portId, String artifactId)
```

Descrição:

Associa um artefato a uma porta de uma instância de atividade simples.

Parâmetros:

`activityInstanceId` - Identificador da instância da atividade simples.

`portId` - Identificador da porta.

`artifactId` - Identificador do artefato.

Retorna:

Resposta de sucesso da operação (true|false).

---

associateArtifactToCompositeActivityPort

```
String associateArtifactToCompositeActivityPort(
                                    String compositeActivityInstanceId,
                                    String portId, String artifactId)
```

Descrição:

Associa um artefato a uma porta de uma instância de atividade composta.

Parâmetros:

`compositeActivityInstanceId` - Identificador da instância da atividade composta.

`portId` - Identificador da porta.

`artifactId` - Identificador do artefato.

Retorna:

Resposta de sucesso da operação (true|false).

---

instantiateActivity

```
String instantiateActivity(String activityId, String name)
```

Descrição:

Cria uma instância de uma atividade simples.

Parâmetros:

`activityId` - Identificador da atividade simples a ser instanciada.

`name` - Nome da instância da atividade simples.

Retorna:

Identificador da instância da atividade simples (activityInstanceId).

---

instantiateCompositeActivity

```
String instantiateCompositeActivity(String CompositeActivityId,
                                    String name)
```

Descrição:

Cria uma instância de uma atividade composta.

Parâmetros:

`compositeActivityId` - Identificador da atividade composta a ser instanciada.

`name` - Nome da instância da atividade composta.

Retorna:

Identificador da instância da atividade composta (compositeActivityInstanceId).

---

createSynchronism

```
String createSynchronism()
```

Descrição:

Cria um elemento de ponto de sincronismo de workflow.

Retorna:

Identificador de ponto de sincronismo (synchronismId).

---

createDecision

```
String createDecision(String name)
```

Descrição:

Cria um elemento de ponto de decisão de workflow.

Parâmetros:

`name` - Nome do ponto de decisão.

Retorna:

Identificador de ponto de decisão (decisionId).

---

createDecisionOption

```
String createDecisionOption(String decisionPointId, String expression,
                            int toElementType, String toElementId)
```

Descrição:

Associa uma opção para um ponto de decisão.

Parâmetros:

`decisionPointId` - Identificador do ponto de decisão no qual a opção vai ser associada.

`expression` - Nome da instância da atividade composta.

`toElementType` - Tipo do elemento do workflow que vai ser desviado (atividade simples ou composta, ponto de decisão, ponto de sincronismo).

toElementId - Identificador do elemento do workflow que vai ser desviado. OBS: No caso de atividade, usar o id da instância da atividade.

Retorna:

Resposta de sucesso da operação (true|false).

---

associateElementToCompositeActivityWorkflow

```
String associateElementToCompositeActivityWorkflow(
                                String compositeActivity Id,
                                String elementType, String elementId)
```

Descrição:

Associa um elemento em um workflow de uma atividade composta.

Parâmetros:

compositeActivityId - Identificador da atividade composta.

elementType - Tipo do elemento do workflow que vai ser associado (atividade simples ou composta, ponto de decisão, ponto de sincronismo).

elementId - Id do elemento a ser associado. OBS: No caso de atividade, usar o id da instância.

Retorna:

Resposta de sucesso da operação (true|false).

---

defineFlow

```
String defineFlow(String compositeActivityId, String originElementType,
                String originElementId, String destinationElementType,
                String destinationElementId)
```

Descrição:

Define o fluxo entre elementos de um workflow de uma atividade composta.

Parâmetros:

`compositeActivityId` - Identificador da atividade composta. OBS: Deve ser passado o id da definiçao da atividade composta, não o id da instância.

`originElementType` - Tipo do elemento de origem do workflow (atividade simples ou composta, ponto de decisão, ponto de sincronismo).

`originElementId` - Id do Elemento de origem do workflow. OBS: No caso de atividade, usar o id da instância da atividade.

`destinationElementType` - Tipo do elemento de destino do workflow (atividade simples ou composta, ponto de decisão, ponto de sincronismo).

`destinationElementId` - Id do Elemento de destino do workflow. OBS: No caso de atividade, usar o id da instância da atividade.

Retorna:

Resposta de sucesso da operação (true|false).

Anexo B - API DE SERVIÇOS DE PUBLICAÇÃO DE PROVENIÊNCIA RETROSPECTIVA

initializeExperimentExecution

```
String initializeExperimentExecution(String experimentId)
```

Descrição:

Inicializa uma nova execução de um experimento.

Parâmetros:

`experimentId` - Identificador do experimento.

Retorna:

Resposta de sucesso da operação (true|false).

---

notifyActivityExecutionStartup

```
String notifyActivityExecutionStartup(String activityInstanceId,
                                      String[] context)
```

Descrição:

Notifica o início de execução de uma instância de uma atividade simples.

Parâmetros:

`activityInstanceId` - Identificador da instância da atividade simples que foi inicializada.

`context` - Sequência de indentificadores que define a localização da atividade simples no experimento.

Retorna:

Resposta de sucesso da operação (true|false).

---

notifyCompositeActivityExecutionStartup

```
String notifyCompositeActivityExecutionStartup(
                  String compositeActivityInstanceId, String[] context)
```

Descrição:

Notifica o início de execução de uma instância de uma atividade composta.

Parâmetros:

`compositeActivityInstanceId` - Identificador da instância da atividade composta que foi inicializada.

`context` - Sequência de indentificadores que define a localização da atividade composta no experimento.

Retorna:

Resposta de sucesso da operação (true|false).

---

notifyActivityExecutionEnding

```
String notifyActivityExecutionEnding(String activityInstanceId,
                                     String[] context)
```

Descrição:

Notifica o fim de execução de uma instância de uma atividade simples.

Parâmetros:

`activityInstanceId` - Identificador da instância da atividade simples que foi finalizada.

`context` - Sequência de indentificadores que define a localização da atividade simples no experimento.

Retorna:

Resposta de sucesso da operação (true|false).

---

notifyCompositeActivityExecutionEnding

```
String notifyCompositeActivityExecutionEnding(
                  String compositeActivityInstanceId, String[] context)
```

Descrição:

Notifica o fim de execução de uma instância de uma atividade composta.

Parâmetros:

`compositeActivityInstanceId` - Identificador da instância da atividade composta que foi finalizada.

context - Sequência de indentificadores que define a localização da atividade composta no experimento.

Retorna:

Resposta de sucesso da operação (true|false).

---

notifyDecisionPointEnding

```
String notifyDecisionPointEnding(String decisionPointId,
                                 String optionValue, String[] context)
```

Descrição:

Notifica o fim de execução de um ponto de decisão.

Parâmetros:

decisionPointId - Identificador do ponto de decisão que foi finalizado.

optionValue - Opção selecionada para o ponto de decisão.

context - Sequência de indentificadores que define a localização do ponto de decisão no experimento.

Retorna:

Resposta de sucesso da operação (true|false).

---

setArtifactValue

```
String setArtifactValue(String artifactId, String[] context,
                        String value)
```

Descrição:

Publica os dados do artefato.

Parâmetros:

artifactId - Identificador do ponto de decisão que foi finalizado.

context - Sequência de indentificadores que define a localização do artefato no experimento.

value - Valor do artefato.

Retorna:

Resposta de sucesso da operação (true|false).

publishArtifactValueLocation

```
String publishArtifactValueLocation(String artifactId,
                                    String[] context, String hostURL,
                                    String hostLocalPath)
```

Descrição:

Publica a localização dos dados do artefato.

Parâmetros:

`artifactId` - Identificador do ponto de decisão que foi finalizado.

`context` - Sequência de indentificadores que define a localização do artefato no experimento.

`hostURL` - URL da máquina onde o artefato está localizado.

`hostLocalPath` - Endereço local da máquina onde o artefato está localizado.

Retorna:

Resposta de sucesso da operação (true|false).

ANEXO C - EXPERIMENTO DISTEXP

Experimento *DistExp* (*Distributed Experiment*)

Descrição

O modelo conceitual do experimento *DistExp* pode ser visto na Figura B.1 no formato de diagrama de atividades. O experimento *DistExp* consiste em receber um arquivo compactado com parâmetros que deve ser descompactado, processado para extrair seus parâmetros que, finalmente, são usados para calcular o valor médio. De acordo com o diagrama, uma parte do *workflow* do experimento é executado no sistema de gerência de *workflow Kepler* e outra parte no sistema *VisTrails*. A Figura B.2 apresenta o experimento *DistExp* instanciado nos sistemas *Kepler* e *VisTrails*.

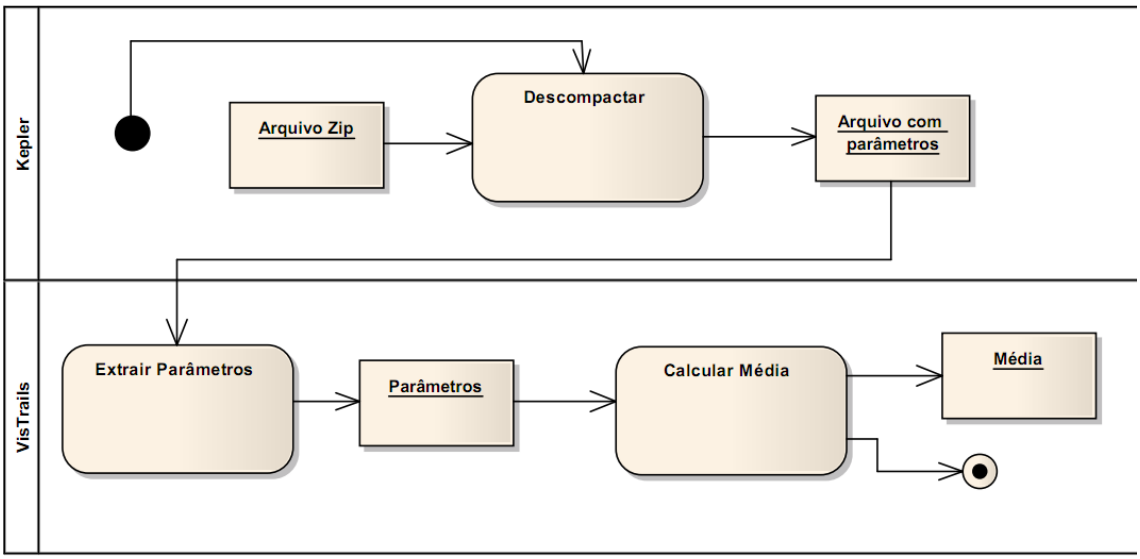

Figura B.1 - Diagrama de atividades do experimento *DistExp*

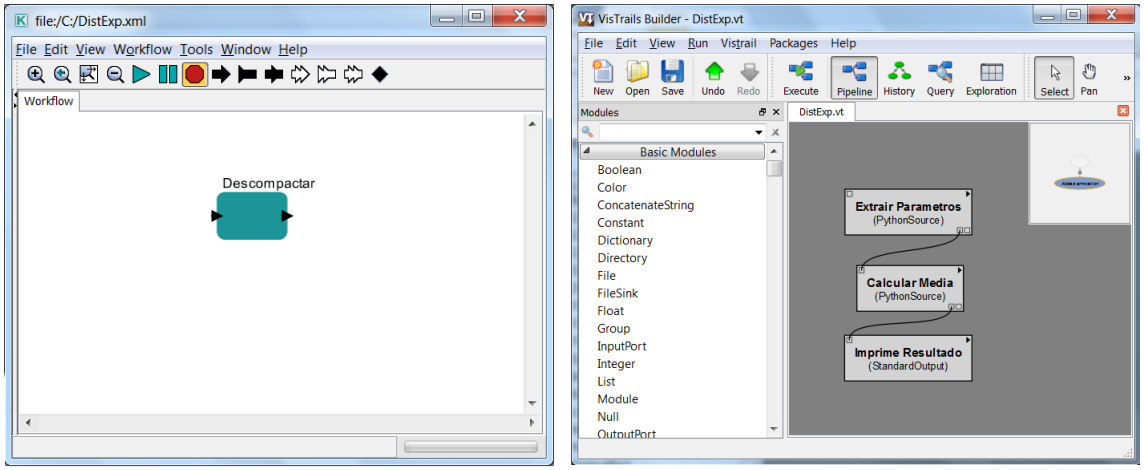

Figura B.2 - Experimento *DistExp* instanciado nos sistemas *Kepler* (janela à esquerda) e *VisTrails* (janela à direita)

ANEXO D - QUESTÕES DE PROVENIÊNCIA SOBRE O EXPERIMENTO DISTEXP

Questões de Proveniência sobre o Experimento DistExp

Questionário Versão A

Nome do Experimento: "DistExp"

Identificador do Experimento: _____

Identificador da Última Execução: _____

Proveniência Prospectiva

1) Quais são as portas da atividade "*ExtrairParametro"*?

R: _____

2) Quais são as atividades presentes no *workflow do VisTrails*?

R: _____

Proveniência Retrospectiva

3) Qual foi o tempo de execução do experimento "*DistExp*"?

R: _____

4) Qual o tempo de execução da atividade "*Descompactar"*?

R: _____

5) Quais foram os dados gerados e consumidos pela atividade “*Descompactar”*?

R: _____

6) Qual o rastro de transformações (*i.e.*, os valores intermediários) do único artefato gerado pela atividade “*CalcularMedia*” até o seu estado final?

R: 1° Estado: _____

2° Estado: _____

3° Estado: _____

Estado Final: _____

Questionário Versão B

Nome do Experimento: “DistExp”

Identificador do Experimento: _____

Identificador da Última Execução: _____

Proveniência Prospectiva

1) Quais são as portas de entrada e saída da atividade “*CalcularMedia”*?

R: _____

2) Quais são as atividades presentes no *workflow do Kepler*?

R: _____

Proveniência Retrospectiva

3) Qual foi o tempo de execução do experimento “*DistExp”*?

R: _____

4) Qual o tempo de execução da atividade “*Descompactar”*?

R: _____

5) Quais foram os dados gerados e consumidos pela atividade “*Descompactar”*?

R: _____

7) Qual o rastro de transformações (*i.e.*, os valores intermediários) do único artefato gerado pela atividade “*CalcularMedia*” até o seu estado final?

R: 1° Estado: _____

2° Estado: _____

3° Estado: _____

Estado Final: _____

ANEXO E - GABARITO DAS QUESTÕES DE PROVENIÊNCIA SOBRE O EXPERIMENTO DISTEXP

Questionário Versão A

Proveniência Prospectiva

8) Quais são as portas da atividade "*ExtrairParametro"*?

R: ArquivoParam, Param.

9) Quais são as atividades presentes no *workflow do VisTrails*?

R: ExtrairParametros, CalcularMedia.

Proveniência Retrospectiva

10) Qual foi o tempo de execução do experimento "*DistExp*"?

R: 15 segundos.

11) Qual o tempo de execução da atividade "*Descompactar"*?

R: 5 segundos.

12) Quais foram os dados gerados e consumidos pela atividade "*Descompactar"*?

R: Gerado: C:\data\param.zip; Consumido: C:\data\param.dat.

13) Qual o rastro de transformações (*i.e.*, os valores intermediários) do único artefato gerado pela atividade "*CalcularMedia*" até o seu estado final?

R: 1° Estado: C:\data\param.zip

2° Estado: C:\data\param.dat

3° Estado: 60;30;22;15

Estado Final:31.75

Questionário Versão B

Proveniência Prospectiva

6) Quais são as portas de entrada e saída da atividade "*CalcularMedia"*?

R: Param, Media.

7) Quais são as atividades presentes no *workflow do Kepler*?

R: Descompactar.

Proveniência Retrospectiva

8) Qual foi o tempo de execução do experimento "*DistExp"*?

R: 10 segundos.

9) Qual o tempo de execução da atividade "*Descompactar"*?

R: 3 segundos.

10) Quais foram os dados gerados e consumidos pela atividade "*Descompactar"*?

R: Gerado: C:\file.zip; Consumido: C:\file.dat

14) Qual o rastro de transformações (*i.e.*, os valores intermediários) do único artefato gerado pela atividade "*CalcularMedia*" até o seu estado final?

R: 1° Estado: C:\file.zip

2° Estado: C:\file.dat

3° Estado: 22;34;67

Estado Final: 41

ANEXO F - Formulário de Consentimento

Formulário de Consentimento

Estudo

Este estudo visa caracterizar a capacidade da abordagem *ProvManager* de responder questões de proveniência no contexto de análise de proveniência de experimentos científicos que são executados em cenários de ambiente distribuído.

Idade

Eu declaro ter mais de 18 anos de idade e concordar em participar de um estudo conduzido por Anderson Souza Marinho na Universidade Federal do Rio de Janeiro.

Procedimento

Este estudo acontecerá em uma única sessão composta de duas etapas, a primeira delas tem como finalidade responder um conjunto de questões de proveniência de um experimento distribuído usando os sistemas de gerência de workflow científico Kepler e VisTrails como suporte. Na segunda etapa, as mesmas perguntas de proveniência serão respondidas, porém usando o protótipo *ProvManager*. Eu entendo que, uma vez o experimento tenha terminado, os trabalhos que desenvolvi serão estudados visando entender a eficiência dos procedimentos e as técnicas propostas.

Confidencialidade

Toda informação coletada neste estudo é confidencial, e meu nome não será divulgado. Da mesma forma, me comprometo a não comunicar os meus resultados enquanto não terminar o estudo, bem como manter sigilo das técnicas e documentos apresentados e que fazem parte do experimento.

Benefícios e liberdade de desistência

Eu entendo que os benefícios que receberei deste estudo são limitados ao aprendizado do material que é distribuído e apresentado. Eu entendo que sou livre para realizar perguntas a qualquer momento ou solicitar que qualquer informação

relacionada a minha pessoa não seja incluída no estudo. Eu entendo que participo de livre e espontânea vontade com o único intuito de contribuir para o avanço e desenvolvimento de técnicas e processos para a experimentação científica.

Pesquisador responsável

Anderson Souza Marinho
Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

Professores responsáveis

Profª. Cláudia Maria Lima Werner
Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ

Prof. Leonardo Paulino Gresta Murta
Instituto de Computação - Universidade Federal Fluminense (UFF)

Nome (em letra de forma):_____

Assinatura:_____ **Data:**_____

ANEXO G - QUESTIONÁRIO DE CARACTERIZAÇÃO

Questionário de Caracterização

1) Formação acadêmica

( ) Doutorado

( ) Doutorando

( ) Mestrado

( ) Mestrando

( ) Graduação

Ano de ingresso: _____ Ano de conclusão (ou previsão de conclusão): _____

2) Formação geral

2.1) Experiência com workflows científicos

Você alguma vez já usou um sistema de gerência de workflow científico?

( ) sim ( ) não

Qual (is): _____

Qual é sua experiência com workflow científico?

( ) muito baixa ( ) baixa ( ) média ( ) alta ( ) muito alta

Você já participou de atividades onde o uso de workflows científicos é uma constante?

( ) sim ( ) não

Se sim, poderia citar? _____

Você já leu alguma publicação específica sobre o assunto?

( ) sim ( ) não

Se sim, poderia citar o nome da publicação ou do autor?_____

2.1) Qual é a sua experiência com a linguagem Prolog?

( ) muito baixa ( ) baixa ( ) média ( ) alta ( ) muito alta

ANEXO H - DESCRIÇÃO GERAL DA TAREFA

Descrição Geral da Tarefa

Instruções

Um experimento científico denominado *DistExp* (*Distributed Experiment*) está configurado e sendo executado em um ambiente distribuído. Neste ambiente, o experimento é composto de dois *workflows*, instanciados em sistemas de gerência de *workflow* científicos (*SGWfC)* distintos, a saber: *VisTrails* e *Kepler*. Você, que exerce o papel de cientista, terá que realizar uma análise básica de proveniência sobre o experimento *DistExp* respondendo um conjunto de questões de proveniência (descritas no documento *Questionário de Proveniência*). Entretanto, esta tarefa terá de ser feita em duas etapas, onde cada etapa adotará uma estratégia de análise de proveniência diferente, a saber:

I - Responder as perguntas de proveniência utilizando apenas os *SGWfC* Kepler e Vistrails como fonte de informação.

II - Responder as perguntas de proveniência utilizando apenas o protótipo ProvaManager como fonte de informação.

ANEXO I - LISTA DE PREDICADOS PROLOG DO PROVMANAGER

Lista de Predicados Prolog do ProvManager

Obter identificador do experimento a partir de seu nome

experimentName(ExperimentId, ExperimentName)

Parâmetros:

ExperimentId *– Identificador do experimento.*

ExperimentName *– Nome do experimento.*

Obter identificador da última instância de execução do experimento

lastExperimentExecution(ExperimentId, ExperimentInstanceId)

Parâmetros:

ExperimentId *– Identificador do experimento.*

ExperimentInstanceId *– Identificador da última Instância de execução do experimento.*

Listar as atividades instanciadas em um SGWfC

listActivitiesFromSWFMS(ExperimentId, SWFMSName, Activities)

Parâmetros:

ExperimentId *– Identificador do experimento.*

SWFMSName *– Nome do SGWfC. Ex: "Vistrails", "Kepler".*

Activities *– Lista das atividades (simples ou compostas) instanciadas no SGWfC.* Formato: [activity(Name),...]. Ex.: [actvity('A'), activity('B')]

Listar portas de uma atividade

activityPorts(ExperimentId, ActivityName, ActivityPorts)

Parâmetros:

ExperimentId *– Identificador do experimento.*

ActivityName *– Nome da atividade.*

ActivityPorts *– Lista das portas da atividade.* Formato: [[PortId, PortName],...]. Ex.: [['1', 'Porta1'], ['2', 'Porta2']].

Tempo de uma execução do experimento

experimentExecutionTime(ExperimentId, ExperimentInstanceId, ExecutionTime)

Parâmetros:

ExperimentId *– Identificador do experimento.*

ExperimentInstanceId *– Instância de execução do experimento.*

ExecutionTime *– Tempo de uma execução do experimento (em milisegundos).*

Tempo de execução de uma atividade do experimento

activityExecutionTime(ExperimentId, ExperimentInstanceId, ActivityName, ExecutionTime)

Parâmetros:

ExperimentId *– Identificador do experimento.*

ExperimentInstanceId *– Instância de execução do experimento.*

ExecutionTime *– Tempo de execução da atividade em uma execução do experimento (em milisegundos).*

Listar artefatos consumidos por uma atividade

artifactsConsumedByActivity(ExperimentId, ExperimentInstanceId, ActivityName, ArtifactIdList)

Parâmetros:

ExperimentId *– Identificador do experimento.*

ExperimentInstanceId *– Instância de execução do experimento.*

ActivityName *– Nome da atividade.*

ArtifactIdList *– Lista dos artefatos consumidos pela atividade.* Formato: [[ArtifactId, ArtifactValue],...]. Ex.: [[‘1’, ‘Valor1’], [‘2’, ‘Valor2’]].

Listar artefatos gerados por uma atividade

artifactsGeneratedByActivity(ExperimentId, ExperimentInstanceId, ActivityName, ArtifactIdList)

Parâmetros:

ExperimentId *– Identificador do experimento.*

ExperimentInstanceId *– Instância de execução do experimento.*

ActivityName *– Nome da atividade.*

ArtifactIdList *– Lista dos artefatos gerados pela atividade.* Formato: [[ArtifactId, ArtifactValue],...]. Ex.: [[‘1’, ‘Valor1’], [‘2’, ‘Valor2’]].

Listar artefatos ancestrais de um artefato

artifactAncestors(ArtifactId, ArtifactAncestors)

Parâmetros:

ArtifactId *– Identificador do artefato a ser verificado.*

ArtifactAncestors *– Lista dos artefatos ancestrais do artefato.* Formato: [[ArtifactId, ArtifactValue],...] A lista começa dos ancestrais mais próximos até os mais distantes. Ex.: [[‘1’, ‘Valor1’], [‘2’, ‘Valor2’]].

ANEXO J - QUESTIONÁRIO DE AVALIAÇÃO

Questionário de Avaliação

Etapa do Estudo que usa *VisTrails* e Kepler

1) Você sentiu dificuldades em responder as questões nesta etapa? Especifique.	☐ Sim ☐ Não ☐ Parcialmente

2) Você ficou satisfeito com o resultado final das tarefas nesta etapa? Especifique, se necessário.	☐ Sim ☐ Não ☐ Parcialmente

3) Você acha a estratégia usada nesta etapa para realizar a análise de proveniência adequada? Explique o porquê da sua resposta.	☐ Sim ☐ Não ☐ Parcialmente

Etapa do Estudo que usa ProvManager

4) Você sentiu dificuldades em responder as questões nesta etapa? Especifique.	☐ Sim ☐ Não ☐ Parcialmente

5) Você ficou satisfeito com o resultado final das tarefas nesta etapa? Especifique, se necessário.	☐ Sim ☐ Não ☐ Parcialmente

6) Você acha a estratégia usada nesta estapa para realizar a análise de proveniência adequada? Explique o porquê da sua resposta.	☐ Sim ☐ Não ☐ Parcialmente

7) Você tem alguma sugestão para melhorar a ferramenta? Especifique.	☐ Sim ☐ Não

Obrigado pela sua colaboração!