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

Tatiane Ornelas Matins Alves

SciProvMiner: Captura e Consulta de Proveniência utilizando Recursos Web Semânticos para Ampliação do Conhecimento Gerado e Otimização do Processo de Coleta

Juiz de Fora
2013

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

Tatiane Ornelas Matins Alves

SciProvMiner: Captura e Consulta de Proveniência utilizando Recursos Web Semânticos para Ampliação do Conhecimento Gerado e Otimização do Processo de Coleta

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

ORIENTADORA: Regina Maria Maciel Braga Villela

Juiz de Fora
2013

Ao meu marido Daniel e meu filho Lucas, que são a razão da minha vida.

AGRADECIMENTOS

Primeiramente a Deus, que me sustentou em todo o tempo e me encorajou a prosseguir apesar das adversidades.

Ao meu pai Sylas e minha mãe Maria José (*in memoria*), que me forneceram toda a bagagem de vida e de educação para que eu pudesse chegar até aqui, e aos meus irmãos Guilherme, Emílio e Luciana que também contribuíram para a minha formação como pessoa.

Ao meu esposo Daniel, pela compreensão e auxílio durante esses anos dedicados ao mestrado e ao meu filho Lucas que, mesmo sem saber, contribuiu grandemente para a realização deste trabalho.

A minha orientadora, professora Regina Maciel Braga Villela, pela orientação, dedicação, paciência e compreensão no decorrer de todo o mestrado.

Aos membros da Banca Examinadora pelo trabalho de avaliação.

Aos amigos-irmãos Simone Carneiro, José Márcio Carneiro e ao Pr. Sérgio Paulo M. Silva pela cobertura espiritual, através de orações e por compartilhar de momentos difíceis, cumprindo a orientação bíblica de chorar com os que choram e se alegrar com os que se alegram.

A minha querida amiga Sara Marques Ribeiro Lima que por várias vezes cuidou do meu filho Lucas para que eu pudesse realizar minhas atividades do mestrado.

Ao Programa de Pós-Graduação em Ciência da Computação, PGCC, pela oportunidade de realização de trabalhos em minha área de pesquisa e ao programa PROQUALI da Universidade Federal de Juiz de Fora que me forneceu bolsa de incentivo a qualificação.

Ao meu chefe, professor Rúbens Fonseca, atual diretor do Instituto de Ciências Exatas da UFJF, pelo apoio a mim fornecido desde o início deste projeto até sua conclusão e aos meus colegas de trabalho Daves Martins e Alcindo Gandhi que me deram cobertura para eu poder tirar licença para a conclusão do mestrado.

Ao professor Custódio que se dispôs prontamente a me auxiliar em questões pontuais do meu trabalho e ao colega Ely Mattos, que se prontificou a analisar a contribuição do meu trabalho e me deu muita força no final dessa etapa.

A Daniel Crawl, que forneceu o workflow por ele desenvolvido no *Third Provenance Challenge* e me passou todas as diretrizes para a sua execução.

A todos aqueles que de alguma forma contribuíram para a conclusão deste projeto.

“Pois quem despreza o dia das coisas pequenas, os humildes começos, os primeiros passos, alegrar-se-á quando ver o prumo na mão de Zorobabel, a obra concretizada."

Zacarias 4.10

RESUMO

Prover informação histórica de experimentos científicos com o objetivo de tratar o problema de perda de conhecimento do cientista sobre o experimento tem sido o foco de diversas pesquisas. No entanto, o apoio computacional ao experimento científico em larga escala encontra-se ainda incipiente e é considerado um grande desafio. Este trabalho tem o intuito de colaborar para as pesquisas nessa área, apresentando a arquitetura SciProvMiner, cujo principal objetivo é coletar proveniência prospectiva e retrospectiva de experimentos científicos fazendo uso de recursos Web semânticos para otimizar o processo de captura das informações de proveniência e aumentar o conhecimento do cientista sobre o experimento realizado.

Como contribuições específicas do SciProvMiner, podemos destacar:

- Desenvolvimento de um modelo para contemplar a proveniência prospectiva e retrospectiva como uma extensão do *Open Provenance Model* (OPM), que em sua forma original modela somente proveniência retrospectiva.
- Especificação e implementação de um coletor de proveniência que utiliza a tecnologia de serviços *Web* para capturar ambos os tipos de proveniência segundo o modelo acima;
- Desenvolvimento de uma ontologia denominada OPMO-e, que estende a ontologia *Open Provenance Model Ontology* (OPMO) de forma a modelar o conhecimento acerca da proveniência prospectiva além da retrospectiva já contemplada na OPMO e onde são implementadas as regras de completude e inferência definidas na documentação do modelo OPM. Estas regras aumentam o conhecimento do cientista sobre o experimento realizado por inferir informações que não foram explicitamente fornecidas pelo usuário e tornando possível a otimização do processo de captura de proveniência e a consequente diminuição do trabalho do cientista para instrumentalizar o workflow;
- Especificação de um banco de dados relacional onde são armazenadas as informações de proveniência capturadas pelo coletor, que pode ser utilizado para ser consultado a respeito da proveniência explicitamente capturada, além de fornecer dadosparaas demais funcionalidades do SciProvMiner.

Palavras-chaves: Web Semântica. Serviços Web. Ontologia. Proveniência. OPM.

ABSTRACT

To provide historical scientific information to deal with knowledge loss about scientific experiment has been the focus of several researches. However, the computational support for scientific experiment on a large scale is still incipient and is considered one of the challenges set by the Brazilian Computer Society for 2006 to 2016 period. This work aims to contribute in this area, presenting the SciProvMiner architecture, which main objective is to collect prospective and retrospective provenance of scientific experiments, using ontologies and inference engines to provide useful information in order to increase the knowledge of scientists about a given experiment.

We can highlight as specific contributions of SciProvMiner:

- Development of a model that encompass prospective and retrospective provenance as an extension of the *Open Provenance Model* (OPM), which originally onlydeals with retrospective provenance.
- Specification and implementation of a provenance collector that uses *Web services* technology to capture both types of provenance (prospective and retrospective) according to the above model;
- Development of an ontology,named Extended OPMO-e, that extends the *Open Provenance Model Ontology* (OPMO) in order to model prospective provenance beyond the retrospective provenance already covered in OPMO and where are implemented inference and completeness rules defined in OPM documentation. These rules increase the knowledge of scientists on the experiment inferring information that were not explicitly provided by the user and making it possible to optimize the provenance capture mechanism and the consequent decrease on scientist work in order to instrument the workflow.
- A relational database specification, where captured provenance information are stored. These information can be used to formulate queries about the provenance explicitly captured, besides provide data to other functionalities of SciProvMiner.

Keywords: Semant Web. Web Service.Ontology. Provenance. OPM.

LISTA DE FIGURAS

LISTA DE TABELAS

Sumário

1. INTRODUÇÃO

1.1. MOTIVAÇÃO

Computação em larga escala tem sido amplamente utilizada como metodologia para a realização de pesquisa científica: existem vários casos de sucesso em muitos domínios, incluindo a física, bioinformática, engenharia e ciências geográficas (WONG et al. 2005).

Apesar de o conhecimento científico continuar sendo gerado de forma tradicional, *invivo e invitro*, nas últimas décadas experimentos científicos passaram a utilizar procedimentos computacionais para simular seus próprios ambientes de execução, dando origem a modalidade de experimentos científicos *in virtuo* (MARINHO 2011). Além disso, até mesmo os objetos e os participantes de um experimento passaram a ser simulados, surgindo a categoria de experimentos *in silico* (TRAVASSOS; BARROS 2003). Essas computações em larga escala, que sustentam um processo científico, são geralmente referidas com *e-Science*, segundo MATTOSO et al. (2008), o termo e-ciência ou *e-Science* significa o apoio ao cientista para o desenvolvimento de ciência em larga escala utilizando infraestrutura computacional.

Em (MATTOSO et al.2008) os autores identificam e discutem diversos desafios para prover apoio computacional ao desenvolvimento de ciência em larga escala, embasados no segundo desafio dos grandes desafios da SBC (CARVALHO et al. 2006), onde é dito que: "O objetivo deste desafio é criar, avaliar, modificar, compor, gerenciar e explorar modelos computacionais para todos esses domínios e aplicações". Dentre os desafios de *e-Science*encontra-se o de prover informação histórica dos experimentos científicos, ou proveniência dos dados, em vistas a tratar o problema de perda de conhecimento do cientista sobre o experimento. Segundo Miles et al. (2006), em experimentos científicos, é de vital importância armazenar o processo experimental para posterior uso, tal como na interpretação dos resultados, na verificação de que ocorreu o processo correto ou para rastrear a origem dos dados.Um experimento científico com tal apoio visa a ser reproduzível, permitindo o rastreamento dos processos que levaram a um determinado resultado, facilitando assim o compartilhamento e compreensão do processo científico por diversos pesquisadores. Este rastreamento dos dados e dos processos que os geraram é denominado proveniência de dados.

Considerando este cenário em que as informações de origem do dado têm potencial para agregar valor de maneira significativa na gerência e análise dos resultados de experimentos científicos, encontra-se a principal motivação para o presente trabalho.

1.2. OBJETIVO

Tomando por base que para se obter benefícios das informações de proveniência, é necessário que estas sejam capturadas, modeladas e armazenadas de modo integrado para posterior consulta (MARINHO 2011), o presente trabalho tem como objetivo principal especificar uma arquitetura para a coleta, armazenamento e consulta da proveniência de dados e processos no contexto de experimentos científicos realizados através de simulações computacionais, respaldado na hipótese de que a captura e gerência de dados de proveniência irá fornecer ao cientista informações importantes a respeito do experimento realizado com o potencial de auxiliá-lo a formar uma visão da qualidade, da validade e da atualidade acerca da informação produzida no contexto do experimento científico modelado computacionalmente.

A arquitetura proposta, denominada SciProvMiner, provê uma camada de interoperabilidade capaz de interagir com os SGWfC (Sistemas de Gerenciamento de Workflows Científicos) tendo como finalidade capturar as informações de proveniência prospectiva e retrospectiva geradas a partir de workflows científicos processados computacionalmente, utilizando para isso um coletor de proveniência baseado em tecnologia de serviços Web o que torna o mecanismo de coleta independentemente do SGWfC no qual o workflow foi desenvolvido. A camada de consulta aos dados coletados utiliza recursos da Web semântica tais como ontologia e máquina de inferência para ampliar o conhecimento do cientista a respeito do experimento realizado, disponibilizando para ele informações além daquelas explicitamente informadas na captura. Além disso, a utilização de recursos Web semânticos possibilita que o procedimento de captura dos dados de proveniência seja otimizado, diminuindo o processamento necessário para que a tarefa seja realizada bem como o trabalho do cientista na instrumentalização do workflow que terá a proveniência capturada.

Com o objetivo de facilitar a interoperabilidade de proveniência entre diversos SGWfCs heterogêneos, foi desenvolvido o modelo de proveniência denominado *Open Provenance Model*, concebido como resultado do primeiro e segundo episódio da série *Provenance Challenge* realizado em 2006 e 2007 (MOREAU et al. 2011). Mais recentemente foi desenvolvido outro modelo de proveniência denominado PROV pelo grupo incubador de

proveniência da W3C com o mesmo objetivo do OPM (BIVAR et al. 2013). Ambos os modelos têm o foco apenas na proveniência retrospectiva, visando representar os acontecimentos passados e não os acontecimentos futuros. Porém, de acordo com Lim et al. (2011), ambos os tipos de proveniência, prospectiva, que captura a especificação abstrata do workflow como uma receita para derivação de dados futuros, e retrospectiva, que captura a execução do workflow e as informações de derivação dos dados fornecem informações importantes para a análise de resultados científicos. Como resultado muitas consultas relacionadas à especificação do workflow não podem ser respondidas baseadas no modelo OPM e nem no modelo PROV.

Como o presente trabalho objetiva capturar ambos os tipos de proveniência, foi realizada uma extensão do *Open Provenance Model* (OPM) para englobar também a proveniência prospectiva. O modelo OPM foi escolhido para ser utilizado no SciProvMiner por ser mais simples que o PROV (BIVAR et al. 2013), por ser, até o presente momento, o modelo de proveniência mais utilizado, e por possuir em sua documentação regras de completude e inferências bem definidas que são implementadas na ontologia que o SciProvMiner utiliza em sua camada de consulta Web semântica aos dados de proveniência.

Como este trabalho tem por característica utilizar recursos da Web semântica para aumentar o conhecimento do cientista a respeito do experimento realizado, e o modelo OPM disponibilizar a *Open Provenance Model Ontology* (OPMO) que captura os conceitos do modelo OPM, foi realizada uma extensão desta ontologia, denominada OPMO-e, de forma a modelar o conhecimento acerca da proveniência prospectiva além da retrospectiva, já contemplada na OPMO. Além disso, foram implementadas regras ontológicas baseadas no conceito de cadeia de propriedades disponível na OWL2, com os seguintes objetivos:

- Viabilizar a implementação das regras de completude definidas no modelo OPM, que não eram passíveis de serem capturadas utilizando apenas conceitos de proveniência retrospectiva. Foram construídas propriedades a partir da combinação de propriedades relativas à captura da proveniência prospectiva em conjunto com propriedades relacionadas à captura da proveniência retrospectiva;
- Implementar as inferências em múltiplos passos definidas na documentação do modelo OPM;
- Implementar regras de otimização que dispensam a instrumentalização de certas dependências causais, por torná-las passíveis de serem inferidas;

- Implementar inferências a respeito da proveniência prospectiva, que infere, por exemplo, quando um artefato não é um parâmetro fornecido por usuário, ou quais componentes estão ligados como antecessor e sucessor de um determinado componente.

A criação dessas regras tem o objetivo de aumentar o conhecimento do cientista sobre o experimento realizado, inferindo informações que não foram explicitamente fornecidas pelo usuário e tornando possível a otimização do processo de captura de proveniência e a consequente diminuição do trabalho do cientista para instrumentalizar o workflow.

1.3. ESTRUTURA DO TRABALHO

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

No capítulo 2 são detalhados alguns conceitos fundamentais e tecnologias que embasam o presente trabalho, como a definição de proveniência de dados no contexto de workflows científicos, ontologias, modelos de proveniência entre outros. Este capítulo também apresenta os principais trabalhos relacionados ao SciProvMiner.

O capítulo 3 apresenta a arquitetura do SciProvMiner, detalhando cada uma de suas camadas, e a implementação da solução proposta, apresentando como a ontologia OPMO foi estendida tendo sido adicionada a ela a capacidade de trabalhar com dados de proveniência prospectiva e com as regras de completude e inferência definidas no modelo OPM, além de outras regras que aumentaram o poder semântico da ontologia, dando origem a ontologia OPMO-e.

No capítulo 4 é apresentada a prova de conceito da abordagem, a partir da utilização da arquitetura do SciProvMiner para a captura, gerência e consulta aos dados de proveniência de dois workflows: o *SimpleMathOperations* e o *Load* PC3. O *SimpleMathOperations* é um workflow construído com o objetivo de ser um modelo prático para validar cada uma das funcionalidades presentes no SciProvMiner e para comparar o conhecimento disponibilizado ao usuário pelo SciProvMiner com e sem a realização de otimizações na etapa de instrumentalização do workflow, verificando se a relação custo benefício, dada por uma possível perda de informação em comparação com o ganho na diminuição da tarefa de instrumentalização, é aceitável. O workflow *Load-PC3* é um workflow utilizado em aplicações reais escolhido como *benchmark* para as atividades do *Third Provenance Challenge* e por isso foi considerado adequado para testar a capacidade do SciProvMiner em

trabalhar com workflows utilizados em aplicações reais adequados para flexionar as diferentes características do modelo OPM

Por fim, no capítulo 5 são feitas as considerações finais, listando as contribuições deste trabalho, mostrando as limitações existentes e apresentando orientações para trabalhos futuros.

2. PRESSUPOSTOS TEÓRICOS

Este capítulo discute os principais conceitos e tecnologias envolvidos na elaboração deste trabalho, apresentando ainda os principais trabalhos relacionados à proposta.

2.1. PROVENIÊNCIA DE DADOS

O dicionário Oxford define proveniência como sendo "a fonte ou a origem de um objeto; sua história e linhagem; um registro das últimas variações e passagens de um item através de seus vários proprietários" (FREIRE et al. 2008). O conceito de proveniência é bem definido no contexto de artes ou livrarias digitais, onde existe a necessidade de se definir a história bem documentada de um achado ou de um objeto de valor (MOREAU et al. 2011).

Em se tratando de experimentação científica, proveniência de dados pode ser definida como a informação que auxilia a determinar a derivação histórica do produto de dado, a partir de suas fontes de origem (SIMMHAN et al. 2005). Segundo Freire et al. (2008), em experimentos científicos, proveniência nos ajuda a interpretar e compreender os resultados: através da análise da sequência de passos que levaram a um resultado, podemos obter *insights* sobre a linha de raciocínio utilizada em sua produção, verificar que o experimento foi realizado de acordo com procedimentos aceitáveis, identificar entradas do experimento e, em alguns casos, reproduzir o resultado.

Segundo Goble in Simmhan et al. (2005), pode-se ressaltar a existência de diversas aplicações relacionadas à proveniência de dados. Os dados ancestrais podem, por exemplo, serem usados para:

- Qualidade do dado: linhagem pode ser usada para estimar a qualidade e confiabilidade dos dados com base em dados de origem e nas transformações. Ela também pode fornecer afirmações sobre a prova da derivação dos dados.
- Auditoria: proveniência pode ser usada para auditar os dados, determinar o uso de recursos e detectar erros na geração dos dados.
- Replicação: informações de procedência detalhadas podem permitir a repetição de derivação de dados, e ser uma fórmula para a replicação.
- Autoria: *Pedigree* pode estabelecer os direitos autorais e propriedade dos dados, permitir a sua reprodutibilidade, e determinar a responsabilidade em caso de dados errados.

- Informativo: a utilização mais comum de proveniência é a de fazer consultas, com base nos metadados de linhagem, com o objetivo de descoberta de informação. A proveniência também pode ser utilizada para fornecer um contexto para a interpretação dos dados.

No domínio de workflow científico, segundo Davidson & Freire (2008), proveniência é um componente essencial para permitir reprodutibilidade do resultado, compartilhamento e reuso de conhecimento na comunidade científica. Neste contexto, workflow científico tem se tornado um paradigma computacional poderoso para estruturar e automatizar processos científicos complexos e distribuídos. Proveniência de dados em experimentação científica tornou-se assim um tema tão relevante que gerenciamento de proveniência tem sido identificado como um componente chave da arquitetura de referência para Sistemas Gerenciadores de Workflow Científico (SGWfC) (LIM et al. 2011).

2.1.1. Gerenciamento de Proveniência

Conforme dito, o gerenciamento de proveniência tem sido identificado como um componente chave da arquitetura de referência para Sistemas Gerenciadores de Workflow Científico (SGWfC) (LIM et al. 2011). De acordo com Marinho (2011), para se obter benefícios das informações de proveniência, é necessário que estas sejam capturadas, modeladas e armazenadas de modo integrado para posterior consulta.

2.1.1.1. Representação da informação em proveniência

Segundo Clifford et al. (2008) in Freire et al. (2008), existem duas formas distintas de proveniência: prospectiva e retrospectiva. Proveniência prospectiva captura a especificação de uma tarefa computacional (ou seja, um workflow), que corresponde aos passos que precisam ser seguidos (ou uma receita), para gerar um produto de dados ou classe de produtos de dados. Já a proveniência retrospectiva captura os passos que foram executados, bem como informações sobre os ambientes de execução utilizados para obter um produto de dado específico.

Uma parte importante da informação presente em proveniência de workflow é a informação sobre a causalidade: as relações de dependência entre os produtos de dados e os processos que os geram. Causalidade pode ser inferida a partir de ambas, proveniência prospectiva e retrospectiva, e captura a sequência de passos que, juntamente com dados de

entrada e parâmetros, causou a criação de um produto de dados (CLIFFORD et al. *in* FREIRE et al. 2008).

Outra componente importante em proveniência são informações definidas pelo usuário. Isto inclui a documentação que não pode ser automaticamente capturada, mas registram decisões e notas importantes. Esses dados são frequentemente capturados na forma de anotações (DAVIDSON; FREIRE 2008).

Segundo Freire et al. (2008), uma solução de gerenciamento de proveniência consiste em três componentes principais: um mecanismo de captura, um modelo de representação, e uma infraestrutura para o acesso, armazenamento e consultas. Esses componentes são melhores detalhados nas próximas seções.

2.1.1.2. Captura de Dados de Proveniência

De acordo com Freire et al. (2008) um mecanismo de coleta de proveniência pode trabalhar em três níveis principais: nível de workflow, de sistema operacional (SO), e de atividade.

Quando a captura é realizada no nível workflow, um SGWfC é responsável por reunir todas as informações de proveniência. Mesmo sendo esta a abordagem mais popular, este nível tem a desvantagem de ser dependente do SGWfC. Mecanismos que funcionam no nível do sistema operacional usam as funcionalidades do sistema operacional para capturar informações de proveniência (por exemplo: sistema de arquivos, rastreador de chamadas de sistema, etc.). A vantagem dessa abordagem é a independência dos SGWfCs. No entanto, estes mecanismos não são acoplados com o workflow em todos os processos e, portanto, necessitam de pós-processamento para extrair as relações entre as chamadas do sistema e as tarefas. Já o nível de atividade tenta mesclar as melhores características dos outros dois níveis. Neste nível, cada atividade do workflow é responsável por coletar as suas próprias informações de proveniência. A vantagem deste nível é a independência dos SGWfCs, assim como no nível de SO, e informações mais precisas são recolhidas, assim como no nível de workflow. O problema deste nível é a necessidade de adaptação de atividades pré-existentes para incorporar funcionalidades de coleta de proveniência.

2.1.1.3. Armazenamento, Acesso, e Consulta a Proveniência

Segundo Moreau et al.(2011), várias abordagens existem para a captura e modelagem de proveniência, mas só recentemente o problema de coleta, acesso e consulta começou a receber

atenção. Os pesquisadores têm utilizado uma ampla variedade de modelos de dados e sistemas de armazenamento, variando de linguagens da Web Semântica a dialetos XML especializados e armazenados como arquivos de tuplas em tabelas do banco de dados relacional. Uma das vantagens de armazenamento em sistema de arquivos é que os usuários não precisam de infraestrutura adicional para armazenar informações de proveniência. Por outro lado, um banco de dados relacional fornece armazenamento centralizado e eficiente que um grupo de usuários pode compartilhar.

Infraestrutura efetiva e eficiente para consulta a dados de proveniência é um componente necessário em um sistema de gerenciamento de proveniência, especialmente quando grandes volumes de informação são capturados. Quando uma abordagem baseada no SO captura proveniência de granularidade muito fina, por exemplo, o volume de informações pode ser muito grande, o que torna o gerenciamento dos dados de proveniência complexo. Sobrecarga de proveniência também pode ser um problema para alguns sistemas baseados em workflow. Devido ao fato do workflow poder ser executado várias vezes com várias etapas, a quantidade de informação armazenada por um único workflow pode ser muito grande (MOREAU et al. 2011).

Por outro lado, a consulta a dados de proveniência é um componente importante de uma infraestrutura. A capacidade de consultar uma proveniência de tarefa computacional também possibilita a reutilização de conhecimento. Ao consultar um conjunto de tarefas e sua proveniência, os usuários podem não só identificar as tarefas apropriadas e reutilizá-las, mas também comparar e compreender as diferenças entre diferentes tarefas (MOREAU et al. 2011). Uma característica comum em muitas abordagens para consultar proveniência é que suas soluções estão intimamente ligadas aos modelos de armazenamento utilizado. Assim, elas exigem que os usuários escrevam consultas em linguagens como SQL, Prolog e SPARQL. Embora tais linguagens em geral sejam úteis para aqueles já familiarizados com sua sintaxe, elas não foram projetadas especificamente para proveniência, o que significa que consultas simples podem ser difíceis e complexas para serem escritas. No entanto, mesmo consultas que usam uma linguagem projetada para proveniência tendem a ser complicadas demais para muitos usuários, pois contém informações estruturais de proveniência representadas como um grafo (MOREAU et al. 2008).

Alguns modelos de proveniência usam a tecnologia da Web Semântica tanto para representar quanto para consultar informações de proveniência. Linguagens da Web Semântica, tais como RDF e OWL fornecem uma maneira natural de modelar grafos de

proveniência e habilidade de representar o conhecimento complexo, tais como anotações e metadados. Esta tecnologia tem o potencial de simplificar a interoperabilidade entre diferentes modelos de proveniência, mas é uma questão em aberto como essa tecnologia conseguirá tratar grandes volumes de dados gerados a partir de grandes repositórios de proveniência (MOREAU et al. 2008).

2.1.2. Modelos de Proveniência

Segundo Freire (2008) existe diversos modelos de proveniência propostos por pesquisadores na literatura. Todos estes modelos suportam alguma forma de proveniência retrospectiva, e a maioria dos SGWfCs fornecem meios para capturar proveniência prospectiva. Muitos dos modelos também suportam anotações. Embora estes modelos sejam diferentes de várias maneiras, todos eles compartilham um tipo de informação essencial: a dependência de processos e de dados.

Segundo Marinho (2011), a diversificação de modelos pode prejudicar a tarefa dos cientistas que manipulam vários sistemas de gerenciamento de proveniência. Em cada sistema, o pesquisador tem que saber quais informações de proveniência são suportadas e o modo como estas informações são acessadas. Adicionado a isto está o fato de que a diversidade de modelos prejudica também a comunicação de sistemas que queiram trocar informações de proveniência (LIM et al. 2011).

Preocupados com questões relacionadas à interoperabilidade dos dados de proveniência entre diferentes sistemas, pesquisadores da área promoveram uma série de conferências conhecidas como *Provenance Challenge*, que aconteceram no contexto do *International Provenance and Annotation Workshop* (IPAW), para abordar o tema. O *Provenance Challenge* teve quatro edições, iniciando em 2006 e terminando em 2010. (BIVAR et al. 2013). Como resultado das duas primeiras edições do *Provenance Challenge* foi criado o modelo de proveniência padrão denominado *Open Provenance Model* (OPM) (MOREAU 2011). Mais recentemente foi desenvolvido outro modelo de proveniência denominado PROV (BELHAJJAME et al.2012) pelo grupo incubador de proveniência da W3C com o mesmo objetivo do OPM. Ambos os modelos têm o foco apenas na proveniência retrospectiva, visando representar os acontecimentos passados e não os acontecimentos futuros.

2.1.2.1.OPM

Preocupados com a questão de prover interoperabilidade de sistemas por meio de troca de proveniência, pesquisadores participantes da segunda série do Provenance Challenge, workshop concebido no primeiro *International Provenance and Annotation Workshop* (IPAW) como um meio para construir um consenso em torno do que significa proveniência e identificar maneiras comuns de representar, coletar, partilhar e consultá-la através de sistemas de proveniência, descobriram que havia uma concordância substancial sobre a representação central de proveniência. Como resultado, foi definido um modelo de proveniência, chamado *Open Provenance Model* (OPM), designado para atender os seguintes objetivos (MOREAU et al. 2011):

- Permitir que informações de proveniência sejam trocadas entre sistemas, por meio de uma camada de compatibilidade baseada em um modelo de proveniência compartilhada.
- Permitir aos desenvolvedores construir e compartilhar ferramentas que operam neste modelo de proveniência.
- Definir o modelo de forma precisa, independente da tecnologia.
- Apoiar uma representação digital de proveniência para qualquer "coisa" que seja produzida por sistemas de computador ou não.
- Definir um conjunto básico de regras que identificam as inferências válidas que podem ser feitas em grafos de proveniência.

Ao especificar este modelo, alguns objetivos foram descartados do escopo do OPM, que são:

- Não é o propósito de o OPM especificar as representações internas que os sistemas têm de adotar para armazenar e manipular proveniência internamente; sistemas permanecem livres para adotar representações internas que são próprias para sua finalidade.
- Não é o propósito de o OPM definir uma sintaxe legível por computador; modelos de implementações em XML, RDF e outros estão sendo especificados em documentos separados.
- OPM não especifica protocolos para armazenar informações de proveniência em repositórios de proveniência.
- OPM não especifica protocolos para consultar repositórios de proveniência.

No modelo OPM supõe-se que a proveniência dos objetos (digitais ou não) pode ser representada por um grafo de causalidade anotada, que é um grafo acíclico dirigido, enriquecido com anotações capturadas de outras informações relativas à.

Em OPM, grafos de proveniência são compostos por três tipos de nós (MOREAU et al. 2011):

- Artefatos: representam um dado de estado imutável, que pode ter um corpo físico em um objeto físico, ou uma representação digital em um sistema de computador.
- Processos: representam ações realizadas ou causadas por artefatos, e resultam em novos artefatos.
- Agentes: representam entidades contextuais agindo como um catalisador de um processo, permitindo, facilitando, controlando, ou afetando sua execução.

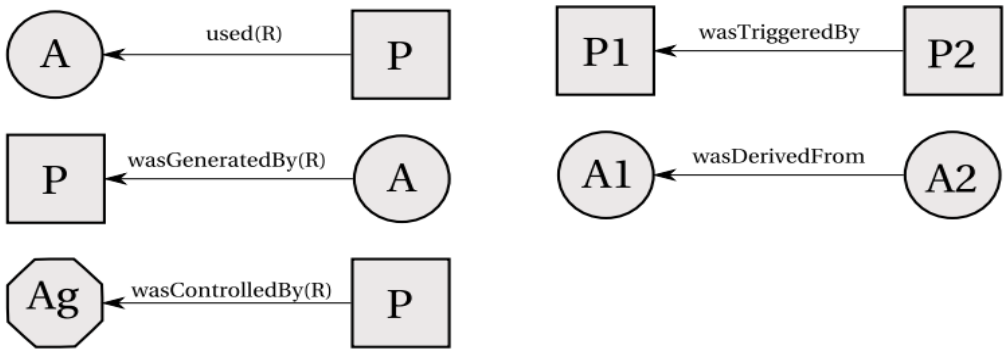


Figura 2.1. Arestas no OPM: origens são efeitos e destinos são causas (adaptado de MOREAU et al. 2011)

A Figura 2.1, adaptada de Moreau et al. (2011), ilustra essas três entidades e suas possíveis formas de relacionamento, chamadas também de dependências. À esquerda da Figura 2.1, as duas primeiras arestas 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. A terceira aresta indica que um processo foi controlado por um agente. Diferente das outras duas primeiras arestas mencionadas, esta representa um relacionamento de controle. A quarta aresta é usada em situações onde não se sabe exatamente quais artefatos foram utilizados por um processo, porém 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, a quinta aresta é utilizada 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 ainda tem sido o modelo de referência para solução de proveniência mais utilizada atualmente (BIVAR et al. 2013). É importante lembrar que o OPM não fornece suporte à representação de informações sobre a especificação do workflow e, por isso, pode ser classificado como um modelo apenas de proveniência retrospectiva.

Em Moreau et al. (2011) é dito ser esperado que algoritmos inteligentes possam explorar o modelo de dados OPM para fornecer novas e importantes funcionalidades aos usuários. Este mecanismo é possível a partir da utilização das regras de completude e inferências válidas no modelo OPM. São detalhadas a seguir as regras de completude segundo a documentação do modelo OPM.

A primeira regra de completude definida no modelo OPM define a transformação bidirecional apresentada na Figura 2.2.

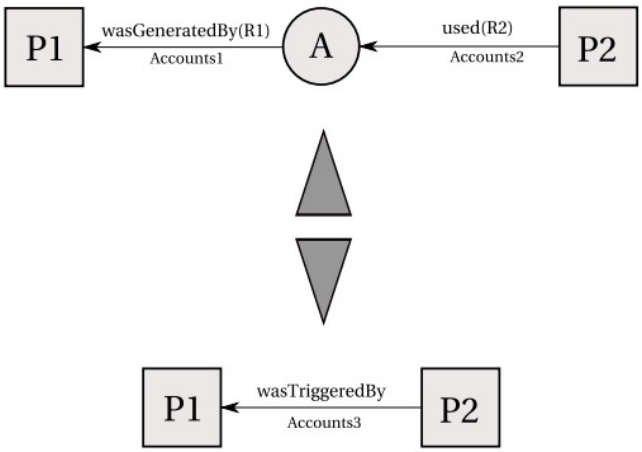


Figura 2.2. Completude: Eliminação e Introdução de Artefato (MOREAU et al. 2011)

De acordo com a transformação acima, uma aresta *wasTriggeredBy* pode ser obtida a partir da existência de uma aresta *used* e uma aresta *wasGeneratedBy*. Apesar de ser bidirecional, a transformação no sentido de baixo para cima da regra representada na Figura 2.2 é considerada transformação com perda de informação, uma vez que a dependência causal *wasTriggeredBy* é considerada um resumo com perda das arestas *used* e *wasGeneratedBy*, pois não é possível saber qual foi o artefato utilizado como causa do *used* e como efeito do *wasGeneratedBy*.

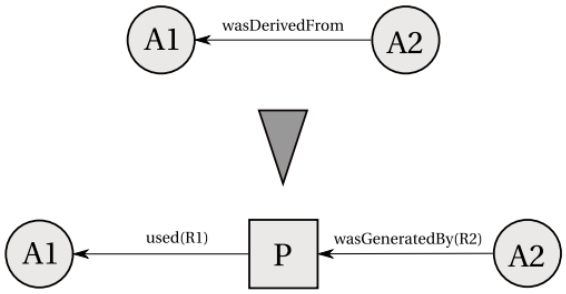


Figura 2.3. Completude: Introdução de Processo (MOREAU et al. 2011)

Na segunda regra de completude do modelo OPM apresentada na Figura 2.3, a documentação do modelo OPM afirma que a regra é unidirecional, denominada Introdução de Processo. Nesta regra uma aresta *wasDerivedFrom* esconde a presença de um processo intermediário P. Porém, a regra recíproca não se sustenta, uma vez que, sem qualquer conhecimento interno do Processo P não é possível afirmar que existe uma dependência entre A1 e A2. No entanto, em notas de rodapé da documentação do modelo OPM é sugerido que, se existir uma anotação indicando que todas as saídas de um processo do workflow são dependentes de todas as suas entradas, então a inferência inversa, isto é, a eliminação de processo, pode ser afirmada. No SciProvMiner essa confirmação se dá pelo parâmetro *AllOutputsDependentAllInputs* da instância do serviço Web instrumentalizado com o método *initialConfiguration* no workflow em questão, que quando configurado para *true*, significa que essa inferência pode ser assumida e false caso contrário. No presente trabalho essa regra será chamada de Eliminação de Processo. O detalhamento desta regra será feito no capítulo 3.

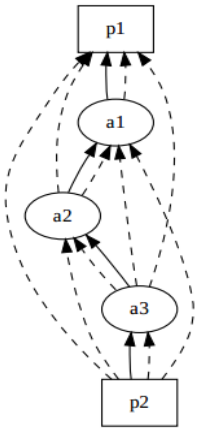
Quando usuários desejam encontrar a causa de um artefato ou um processo, eles podem não estar interessados em causas diretas, mas sim em causas indiretas, que envolvem múltiplas transações. Com a finalidade de se permitir expressar essas causas indiretas, a documentação do modelo OPM introduz quatro novos relacionamentos que são versões em múltiplos passos de relacionamentos existentes. O primeiro introduz a relação multi-passos *wasDerivedFrom*, a partir da qual outras versões são obtidas (MOREAU et al. 2011).

A aresta multi-passos *wasDerivedFrom* define que um artefato a_1 foi derivado de a_2 (possivelmente utilizando múltiplos passos), escrito como $a_1 \rightarrow^* a_2$, se a_1 “foi derivado” do artefato a_2 ou de um artefato derivado de a_2 (possivelmente em múltiplos passos). Em outras palavras, esta é a clausura transitiva da aresta *wasDerivedFrom*. Isso expressa que a_2 teve uma influência no artefato a_1.

A partir da definição da aresta multi-passos *wasDerivedFrom*, foram formuladas as seguintes arestas múltiplos passos secundárias (MOREAU et al. 2011):

- Um processo p *used* o artefato a (possivelmente usando múltiplos passos), p→*a, se p usou o artefato a ou um artefato *wasDerivedFrom* a (possivelmente utilizando múltiplos passos).
- O artefato a *wasGeneratedBy* processo p (possivelmente utilizando múltiplos passos), escrito por a→*p, se a foi um artefato ou foi derivado de um artefato (possivelmente usando múltiplos passos) que foi gerado por p.
- O processo p_1 *wasTriggeredBy* p_2 (possivelmente usando múltiplos passos), escrito por $p_1 \rightarrow^* p_2$, se p_1 usou um artefato que *wasGeneratedBy* ou *wasDerivedFrom* um artefato que foi em si gerado por p_2.

Intuitivamente, arestas multi-passos podem ser inferidas a partir de arestas de único passo, pela eliminação de artefatos que ocorrem em cadeias de dependências. Os quatro relacionamentos e inferências associadas estão ilustrados na Figura 2.4. Nesta figura, arestas planas representam dependências de único passo, enquanto as tracejadas representam dependências multi-passos. Por exemplo, a partir de p2→a3→a2 pode-se inferir p2*→a3*→a2* e p2*→a2*, pela eliminação de a3.



Arestas Único Passos (Plana)

GeneratedBy: $a_1 \rightarrow p_1$
Used: $P_2 \rightarrow a_3$
DerivedFrom: $a_3 \rightarrow a_2$, $a_2 \rightarrow a_1$

Arestas Multi-passos inferidas (tracejadas)

GeneratedBy*: $a_1 \rightarrow {}^*p_1$, $a_2 \rightarrow {}^*p_1$, $a_3 \rightarrow {}^*p_1$
Used*: $p_2 \rightarrow {}^*a_3$, $p_2 \rightarrow {}^*a_2$, $p_2 \rightarrow {}^*a_1$
DerivedFrom*: $a_3 \rightarrow {}^*a_2$, $a_3 \rightarrow {}^*a_1$, $a_2 \rightarrow {}^*a_1$
wasTriggeredBy*: $p_2 \rightarrow {}^*p_2$

Figura 2.4. Inferência: Arestas Multi-passos (Adaptada de MOREAU et al. 2011)

2.1.2.2.PROV

O PROV (BELHAJJAME et al. 2012) é um modelo de proveniência especificado pela W3C, que objetiva expressar a proveniência de dados através da descrição das entidades, atividades e agentes envolvidos em produzir, entregar ou enunciar um determinado objeto.

Assim como o OPM, o PROV utiliza um grafo para representar as informações de proveniência com três tipos de vértices e dez tipos de arestas. O primeiro tipo de vértice é a Entidade que é representada por círculo, pode ser real ou imaginária. A entidade, para o PROV, é algo conceitual, físico ou digital, ou alguma outra coisa que possua algum aspecto fixo. O segundo tipo de vértice é representado por um retângulo e é denominado Atividade. Segundo Bivar et al. (2013), uma atividade se caracteriza por ocorrer durante um período de tempo atuando sobre ou com entidades. O terceiro tipo de vértice é representado por um losango e é denominado Agente. Um agente se caracteriza por possuir certa responsabilidade por uma atividade, pelo fato de uma entidade existir ou pelas atividades de outro agente.

Da mesma forma que o OPM, as arestas do grafo PROV representam dependências causais entre seus nós e são direcionadas do efeito para a causal. As dependências causais do PROV são (BIVAR et al. 2013):

- *used*: Relaciona atividades, afirmando que uma atividade usou outra atividade.
- *wasGeneratedBy*: Relaciona entidades a atividades e indica que uma entidade foi gerada por uma atividade.
- *wasAssociatedWith*: Relaciona atividades e agentes, indicando que uma determinada atividade foi associada a um determinado agente.
- *wasAttributedTo*: Relaciona entidades e agentes e indica que uma entidade foi atribuída a um determinado agente.
- *actedOnBehalfOf*: Relaciona agentes indicando que um agente tem autoridade ou responsabilidade por outro agente.
- *wasRevisionOf*: Relaciona entidades, registrando que uma entidade foi derivada de outra com o caráter corretivo, por exemplo, de correção de erro.
- *wasDerivedFrom*: Da mesma forma que a anterior, essa dependência causal relaciona entidades, no sentido de que uma entidade foi originada da outra. Esta derivação tem o caráter evolutivo, e não corretivo como a anterior.

- *wasInformedBy*: Relaciona atividades implicando que uma atividade informada foi gerada pela atividade que a informou, porém essa atividade é desconhecida ou não é de interesse.
- *wasStartedBy*: Relaciona atividades e entidades indicando que uma atividade iniciou uma entidade. A diferença entre esta dependência causal para *wasGeneratedBy* é que a *wasGeneratedBy* cria a entidade, o que implica em dizer que ela não era existente antes dessa relação ocorrer enquanto a *wasStartedBy* inicia uma atividade já existente previamente.
- *wasEndedBy*: Da mesma forma que a aresta anterior, relaciona atividades a entidades, registrando que uma atividade finalizou uma entidade.

Da mesma forma que o OPM o PROV sempre representa os acontecimentos passados, modelando apenas a proveniência retrospectiva.

2.1.2.3.OPM x PROV

De acordo com Bivar et al. (2013), existem algumas similaridades entre os modelos OPM e PROV, o que torna possível fazer um mapeamento entre alguns conceitos chave de ambos os modelos, ligando o artefato no modelo OPM a entidade no modelo PROV, o processo no OPM a atividade no PROV, o agente no OPM ao agente no PROV. Alguns relacionamentos também são compatíveis, tais como o *used*, *wasGeneratedBy* e *wasControlledBy*. O relacionamento *wasGeneratedBy* do OPM corresponde à relação *wasAssociatedWith* no Prov, ligando atividades (processos no OPM) a agentes (em ambos os modelos).

O relacionamento *wasTriggeredBy* no OPM tem um significado diferente em comparação com o relacionamento *wasInformedBy* no PROV, uma vez que sua função é mostrar que uma atividade particular transmite alguma coisa para outra atividade, enquanto a relação *wasTriggeredBy* do OPM indica que um processo foi disparado por outro. Existe um relacionamento no PROV, o *wasStartedBy*, que é equivalente ao relacionamento *wasTriggeredBy* no OPM. No entanto, este relacionamento tem um amplo espectro porque ele pode ocorrer não somente entre duas atividades, mas também entre uma atividade e uma entidade, afirmando que a entidade é iniciada pela atividade.

No modelo PROV existem quatro relacionamentos que não foram mapeados no OPM: *wasEndedBy*, *actedOnBehalfOf* e *wasAttributedTo*. Esses últimos dois (*actedOnBehalfOf* e *WasAttributedTo*) são relações de delegação e associação entre agentes, entidades e

atividades. De acordo com Bivar et al. (2013), essas relações são importantes porque elas fornecem proveniência centrada em agentes. Este tipo de relação não acontece no OPM. Já o relacionamento *wasEndedBy* representa a finalização do processo.

Comparando os dois modelos, pode ser considerado que o modelo OPM é mais simples e mais orientado a captura de proveniência de processos, enquanto o PROV é mais amplo e permite a captura tanto de proveniência centrada em processo quanto centrada em entidade ou centrada em agente.

O modelo OPM foi escolhido para ser utilizado no SciProvMiner por ser voltado à captura de processos, o que no contexto de workflows científicos se mostra uma característica relevante; por ser, até o presente momento, o modelo de proveniência mais utilizado (BIVAR et al. 2013); e por possuir em sua documentação regras de completude e inferências bem definidas que são implementadas na ontologia que o SciProvMiner utiliza em sua camada de consulta Web semântica aos dados de proveniência. Esta última característica é de grande relevância no contexto desta dissertação, uma vez que a partir da ontologia OPMO-e, pode-se derivar conhecimento implícito nos dados de proveniência, aumentando o conhecimento do cientista acerca do experimento.

O modelo PROV também possui uma ontologia associada. No entanto, ainda não estão definidas regras de completude e inferência como na OPMO. Desta forma, para os propósitos desta dissertação, que é prover conhecimento estratégico para o cientista, a partir de informações implícitas no modelo, o atual estágio da ontologia OPMO é mais adequado.

2.2. ONTOLOGIA

O termo ontologia é originário da Filosofia. Nesse contexto, é usado como o nome de um subcampo da filosofia, denominado o estudo da natureza da existência, que é o ramo da metafísica preocupado com a identificação, em termos mais gerais, dos tipos de coisas que realmente existem, e como descrevê-los (ANTONIOU; HARMELEN 2008).

No entanto, em anos mais recentes, a ontologia tornou-se uma das muitas palavras incorporadas pela ciência da computação, onde é dado um sentido técnico específico, que é bastante diferente do original. No contexto de ciência da computação, pode-se adotar a definição de ontologia dada por Gruber (1993), onde diz que uma ontologia é uma especificação formal explícita de uma conceitualização compartilhada. Fensel (2003) analisa esta definição identificando quatro principais conceitos envolvidos: i) um modelo abstrato de

um fenômeno denominado "conceituação", ii) uma descrição matemática precisa referente à palavra "formal", iii) a precisão dos conceitos e suas relações claramente definidas são expressas pelo termo "explícito" e iv) a existência de uma concordância entre os usuários da ontologia é sugerida pelo termo "compartilhada".

Em geral, uma ontologia descreve formalmente um domínio do discurso, sendo usada para representar uma área do conhecimento. Os termos denotam conceitos importantes do domínio. Por exemplo, em um ambiente universitário, funcionários, alunos, cursos, salas de aulas e disciplinas são alguns conceitos importantes dentro deste domínio (YU 2011).

A utilização de ontologia traz como benefícios a possibilidade de compartilhamento de conhecimento sobre certos conceitos chave de um domínio, além da reutilização do conhecimento acerca de um domínio e do fornecimento de meios para codificar o conhecimento e a semântica de tal forma que máquinas possam entender, além de fazer com que processamento de máquina em larga escala possa ser realizado (YU 2011).

Em Inteligência Artificial, há uma longa tradição de desenvolvimento e utilização de linguagens de ontologias (HEBELER et al. 2009). Os fundamentos da Web semântica foram construídos sobre elas. Atualmente, as línguas mais importantes da ontologia para a Web são os seguintes:

- *Resource Description Framework*[1] (RDF) é um modelo de dados para objetos ("recursos") e relacionamento entre eles, que prevê uma semântica simples para este modelo de dados; estes modelos de dados podem ser representados em uma sintaxe XML.
- RDF Schema é uma linguagem de descrição de vocabulário para descrever propriedades e classes de recursos RDF, com uma semântica para hierarquias de generalização de tais propriedades e classes.
- *Ontology Web Language*[2] (OWL) é uma linguagem de descrição de vocabulário mais rico para descrever propriedades e classes, tais como as relações entre as classes (por exemplo, disjunção), cardinalidade (por exemplo, "exatamente um"), igualdade, tipagem

[1]Resource Description Framework (RDF), recomendação W3C de 22 Fevereiro de 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/
[2]Ontology Web Language (OWL), recomendação W3C de 10 de fevereiro de 2004. http://www.w3.org/TR/owl-features/

mais rica de propriedades, características das propriedades (por exemplo, a simetria), e classes enumeradas (YU 2011).

No contexto da Web semântica, as ontologias podem ser explicitadas através de linguagens de marcação que se relacionam segundo o modelo em cascata proposto por Berners-Lee (2001), conhecido como “bolo de noiva” (Figura 2.5). Nesse modelo, (RDF) e RDF-Schema são consideradas as fundações sobre as quais se assentam linguagens de maior abrangência semântica como OWL.

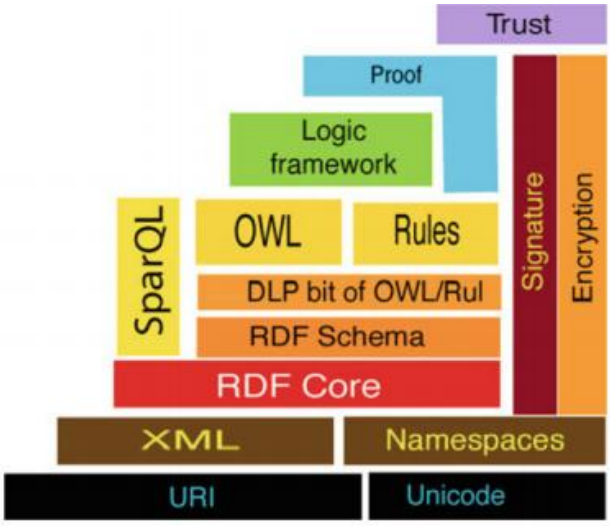


Figura 2.5. Estrutura em camadas da Web semântica (ROURE; GLOBE 2010),

O principal benefício da utilização de OWL é a capacidade para definir semântica que enriquece a informação. Uma base de conhecimento precisa aplicar um componente de inferência para interpretar a semântica e perceber a informação enriquecida. Aplicações que executam inferência são muitas vezes referenciadas como máquinas de inferência, ou raciocinadores (*reasoners*). Uma máquina de inferência é um sistema que infere novas informações com base no conteúdo de uma base de conhecimento (HEBELER et al. 2009).

Em Moreau et al. (2011) é definida uma ontologia OWL para capturar os conceitos do OPM na versão 1.1 e as inferências válidas neste modelo. Além disso, esta ontologia OWL especifica uma serialização RDF da versão 1.1 do modelo abstrato OPM. A essa ontologia é dado o nome de *Open Provenance Model Ontology* (OPMO). A ontologia OPMO utiliza o OPMV (*Open Provenance Model Vocabulary*), um vocabulário leve, para descrever os conceitos fundamentais do modelo OPM. Este vocabulário é essencialmente focado nas declarações de informações de procedência. Apesar deste vocabulário permitir que a maioria

dos conceitos OPM sejam declarados, ele não permite que inferências sejam realizadas. Este é o objetivo da ontologia OPMO: permitir completa expressividade dos conceitos OPM, e permitir inferências. No entanto, algumas regras e inferências não foram especificadas no modelo e, além disso, outras sequer foram especificadas, uma vez que tratam da junção da proveniência prospectiva e retrospectiva.

Uma das contribuições dessa dissertação é, portanto, especificar essas regras de completude, considerando inclusive a proveniência retrospetiva juntamente com a prospectiva para derivação de conhecimento implícito.

2.3. TRABALHOS RELACIONADOS

Considerando o gerenciamento de proveniência, pesquisas têm sido feitas no intuito de solucionar desafios no domínio científico. No entanto, poucos trabalhos focam na captura de proveniência utilizando um modelo padrão e no uso de funcionalidades da Web semântica. A seguir discutem-se alguns trabalhos relacionados à captura de proveniência de dados, cujo foco seja no uso de um modelo integrador e/ou uso de funcionalidades da Web semântica. Não é objetivo detalhar todos os trabalhos relacionados à proveniência, mas sim aqueles que possuem alguma similaridade com o trabalho dessa dissertação.

Em Marinho (2011), o autor propõe uma arquitetura para gerência de proveniência de dados denominada ProvManager. Tanto a arquitetura do SciProvMiner quanto a do ProvManager têm como foco a captura da proveniência em workflows orquestrados a partir de recursos heterogêneos e geograficamente distribuídos baseados na invocação de serviços Web. Porém o SciProvMiner provê uma infraestrutura baseada na Web semântica para representação e consulta aos metadados de proveniência, o que lhe confere um maior poder de expressividade para inferência de conhecimento novo. Por outro lado, o ProvManager estabelece um mecanismo de coleta automatizada de proveniência, enquanto o SciProvMiner não trabalha esse requisito. ProvManager coleta tanto proveniência prospectiva quanto retrospectiva. O SciProvMiner coleta também a proveniência prospectiva e retrospectiva, com o diferencial de propor a utilização de uma extensão do modelo OPM para a captura tanto da proveniência retrospectiva (inerente ao modelo) quanto da proveniência prospectiva. A vantagem de tal abordagem é a interoperabilidade dos dados capturados, uma vez que se está utilizando um modelo padrão. Outra característica do SciProvMiner não encontrada no

ProvManager é a capacidade deste em explorar as regras de completude e inferências válidas no modelo OPM para fornecer ao cientista conhecimento implícito.

Em Lim et al. (2010) é proposta uma extensão do modelo OPM para modelar proveniência prospectiva, além da retrospectiva já suportada no OPM nativo e um framework para coletar ambas, proveniência prospectiva e retrospectiva. O SciProvMiner também utiliza uma extensão do OPM para suportar a modelagem de proveniência prospectiva. Porém, a arquitetura proposta em Lim et al. (2010) não provê infraestrutura baseada na Web semântica - RDF, OWL, ontologias, máquinas de inferência, como é feito no SciProvMiner. Em Lim et al. (2010) também é proposta a implementação das regras de completude e inferência definidas no modelo OPM através de Views do banco de dados, no entanto apenas uma das regras de completude definidas no modelo OPM é implementada enquanto no SciProvMiner as três regras definidas no modelo são implementadas, fornecendo ao usuário maiores possibilidades de explorar o conhecimento acerca do experimento realizado. A forma de captura de proveniência prospectiva definida em Lim et al. (2010), está intimamente relacionada com a ferramenta View (LIM et al. 2010), sendo totalmente dependente desta ferramenta. A forma de captura, tanto da proveniência prospectiva quanto da retrospectiva proposta no SciProvMiner é realizada através de *Web service*, e, portanto, é independente do SGWfCs que o cientista está utilizando.

Em outro trabalho dos mesmos autores do trabalho anterior é proposto um sistema, denominado OPMProv, para integração de proveniência de workflows desenvolvidos por diferentes SGWfCs e consulta a esses dados. A integração é viável para SGWfCs que possuem mecanismos próprios para capturar a proveniência de forma compatível com o modelo OPM, pois os autores de Lim et al. (2011) propõem um algoritmo de mapeamento de dados, denominado *OPMXMLInsert*, para mapear a proveniência capturada por esses diferentes SGWfCs dentro do banco de dados relacional do OPMProv, onde as informações ficam disponíveis para serem consultadas de maneira integrada. O SciProvMiner também possui a proposta de armazenar a proveniência capturada a partir de diversos SGWfCs, no entanto a proposta do SciProvMiner é realizar a captura da proveniência, enquanto o OPMProv não captura a proveniência, apenas traduz a proveniência capturada pelo SGWfC no banco de dados relacional do OPMProv. Outra diferença do SciProvMiner para o OPMProv é que o OPMProv não captura a proveniência prospectiva, pois, de acordo com os autores, o objetivo do OPMProv é promover uma solução totalmente compatível com o modelo OPM. Os autores de Lim et al. (2011), assim como é feito no SciProvMiner, também

propõem a implementação das regras de completude e inferências definidas na documentação do modelo OPM. No entanto eles utilizam Views no banco de dados relacional para implementar as inferências, enquanto o SciProvMiner utiliza tecnologias próprias da Web semântica para isto. Além disso, da mesma forma que o trabalho anterior, eles implementam apenas uma das três regras de completude definidas no modelo OPM, enquanto o SciProvMiner implementa todas as regras de completude e inferência definidas no modelo OPM.

Em Cuevas-Vicenttin et al. (2012) os autores propõem um modelo que estende o OPM, denominado D-OPM, com aspectos específicos de workflow, pois, de acordo com eles, como o modelo OPM é definido para ser uma representação minimalística de proveniência para qualquer "coisa", ele exclui aspectos específicos de workflows científicos. O trabalho apresentado tem em comum com o SciProvMiner o fato de ambos promoverem uma extensão do modelo OPM, com o objetivo de modelara captura da proveniência prospectiva além da retrospectiva já definida no modelo OPM original. Porém, o modelo proposto neste trabalho relacionado considera aspectos adicionais que não são considerados no SciProvMiner, tais como evolução do workflow e as estruturas dos dados. Outra diferença é que em Cuevas-Vicenttin et al. (2012) os workflows devem ser especificados em uma linguagem específica criada por eles, denominada KPN *Language*, e a interoperabilidade com SGWfCs é objetivada ser alcançada através de *wrappers* que estão sendo desenvolvidos por eles para SGWfCs tais como Kepler, Taverna e Vistrails, enquanto no SciProvMiner os workflow são desenvolvidos nos próprios SGWfCs e a captura da proveniência é realizada através de instanciações de serviços web do SciProvMiner dentro do SGWfC no qual o workflow foi modelado. Outra diferença é que no trabalho exposto em Cuevas-Vicenttin et al. (2012) não são utilizados recursos web semânticos tais como ontologia com é feito no SciProvMiner.

Considerando os trabalhos acima citados, a arquitetura do SciProvMiner utiliza tecnologia Web semântica, juntamente com um mecanismo de captura de proveniência independente do SGWfC para processar as consultas de proveniência. Essas são as principais contribuições do SciProvMiner quando comparadas com as abordagens anteriores. Usando ontologia e máquinas de inferência, o SciProvMiner pode fazer inferências sobre a proveniência e, com base nessas inferências, obter importantes resultados relacionados à extração da informação além daquelas que estão explicitamente registradas na base de conhecimento.

Em Bowers et al. (2012), os autores apresentam uma abordagem que usa regras explícitas definidas pelo usuário para inferir dependência de dados dos rastros da execução do

workflow, gerando as informações de proveniência. Esta abordagem usa uma linguagem de regra de dependência que converte as regras de dependências em alto nível em consultas relacionais. Considerando o SciProvMiner, a proposta apresentada em Bowers et al. (2012) não usa um modelo de proveniência padrão como OPM ou PROV, que pode ser uma desvantagem uma vez que o usuário precisa conhecer como usar a linguagem de regra de dependência, embora os autores digam que é compatível com OPM e PROV. Outra diferença é que apenas as informações relacionadas à execução do workflow são capturadas nesta abordagem, enquanto no SciProvMiner além da proveniência retrospectiva, também é capturada a proveniência prospectiva.

Em Chebotko et al. (2010), é discutida uma abordagem para o gerenciamento de proveniência que integra tecnologias Web semânticas com SGBD. Essa abordagem usa de forma ampla tecnologias Web semânticas tais como inferência e capacidades de interoperabilidade. O SciProvMiner também adota tecnologias Web semânticas junto com armazenamento em SGBD. Ele também lida com problemas relacionados à interoperabilidade fornecendo uma ferramenta de instrumentalização que captura os dados de proveniência de maneira independente do formato de proveniência de qualquer SGWfC. No entanto, o SciProvMiner usa um modelo de proveniência padrão, o OPM.

Por fim, em Amann et al. (2013), é apresentada uma abordagem que gera dados de proveniência usando documentos XML. A proposta está relacionada ao ambiente WebLab. Esta abordagem é parecida com o SciProvMiner por considerar tecnologias Web semânticas para inferir conhecimento sobre proveniência de dados no contexto de workflows científicos, mas ela não trabalha provendo uma forma de captura de proveniência a partir de workflows modelados em SGWfCs com é feito no SciProvMiner, e sim inferindo informações de proveniência a partir de documentos XML gerados pela plataforma WebLab. Além disso, o modelo de proveniência padrão utilizado em Amann et al. (2013) é o PROV enquanto o SciProvMiner utiliza o OPM.

2.4. SCIPROV

Considerando que o SciProvMiner é uma extensão de um trabalho anterior feito pelo grupo, denominado SciProv (VALENTE 2011), é importante destacar as principais características da arquitetura anterior, no intuito de que o leitor possa identificar as características que diferem a abordagem atual SciProvMiner, desta versão anterior.

O Sciprov (*Scientific Workflow Provenance System*) especifica uma arquitetura para a coleta e gerência da proveniência no contexto de experimentos científicos distribuídos e interconectados através de uma grade computacional. Para isso, o SciProv define uma camada de interoperabilidade que interage com SGWfCs para capturar os metadados de proveniência gerados em workflows científicos. O autor considerou os seguintes requisitos como fundamentais no projeto da arquitetura (VALENTE 2011):

i. Independência em relação aos modelos de dados e processos adotados pelos SGWfC existentes;

ii. Aplicabilidade direcionada a experimentos científicos;

iii. Emprego de tecnologias da Web Semântica para representação e consulta aos dados de proveniência;

iv. Possibilidade de ajuste no mecanismo de captura para permitir o controle sobre o impacto dos processos de coleta e persistência dos dados de proveniência no desempenho da execução dos experimentos científicos;

v. Captura da proveniência utilizando o modelo OPM.

Entre as propostas de arquiteturas para o tratamento da proveniência de dados no contexto de *e-Science* disponíveis na literatura científica, o SciProv se diferenciou por apresentar um enfoque que se baseia no emprego de recursos da Web semântica e na adoção do modelo abstrato OPM, e por permitir a coleta e gerência da proveniência de dados em diversos níveis de abstração.

Para isso, o SciProv adota uma solução baseada em serviços Web para a coleta dos metadados de proveniência em razão de fatores como adequação a ambientes distribuídos e heterogêneos, uso de protocolos abertos e consensuais para a internet e interoperabilidade. Para a persistência dos metadados de proveniência coletados, o SciProv adota uma solução a partir de uma base de dados relacional em razão da confiabilidade e independência em relação a formatos de dados específicos utilizados por SGWfCs. O SciProvMiner é uma evolução da arquitetura SciProv. Neste contexto, toda a arquitetura do SciProv foi remodelada, os modelos de persistência de dados e a arquitetura baseada em *Web services* foi aprimorada e novas funcionalidades foram adicionadas, conforme será detalhado no capítulo 3.

2.5. CONSIDERAÇÕES FINAIS

Este capítulo apresentou os principais conceitos relacionados ao desenvolvimento da arquitetura SciProvMiner tais como proveniência de dados, modelos de proveniência e ontologia. Neste capítulo também foram apresentados e comparados os principais trabalhos relacionados à proposta do presente trabalho e foi apresentada a arquitetura SciProv, do qual o SciProvMiner é uma extensão.

3. SCIPROVMINER - ARQUITETURA PARA COLETA, ARMAZENAMENTO E CONSULTA DE PROVENIÊNCIA DE DADOS

Neste capítulo apresentamos a arquitetura do SciProvMiner, que conforme dito anteriormente, é uma evolução da arquitetura SciProv (Scientific Workflow Provenance System), definida em Valente (2011), com o objetivo de fornecer ao usuário mais conhecimento acerca da proveniência capturada e prover funcionalidades que aumentem o poder de análise do cientista sobre o experimento realizado, diminuindo o esforço de instrumentalização necessário para que a proveniência do workflow seja capturada como um todo. Abaixo estão listadas as principais contribuições do SciProvMiner em relação ao SciProv:

- Refinamento do mecanismo de instrumentalização de workflows científicos, com o objetivo de otimizar o desempenho do processo de coleta e persistência dos metadados de proveniência, e diminuir o trabalho do cientista na instrumentalização do workflow.
- Manutenção da aderência da arquitetura ao modelo abstrato OPM, considerando desta forma a versão mais recente 1.1, incorporando à arquitetura todas as funcionalidades disponíveis nesta versão.
- Adaptação da arquitetura para permitir a captura, gerência e consulta aos dados de proveniência prospectiva além da retrospectiva, permitindo uma completa cobertura da coleta dos dados de proveniência, tornando possível que questões relacionadas à especificação do workflow, que não podem ser respondidas utilizando apenas proveniência retrospectiva (LIM et al. 2010), possam ser respondidas pelo SciProvMiner.
- Especificação de regras de completude e inferência válidas no modelo OPM, modeladas através de ontologia, que permitem ao cientista explorar os dados de proveniência coletados de forma a obter conhecimento útil e inesperado pelo processamento de consultas a partir de máquinas de inferência, capazes de efetuar deduções sobre essas bases de conhecimento e obter resultados importantes ao extrair informações adicionais além daquelas que se encontram registradas de forma explícita nos dados de proveniência capturados.

- Extensão da ontologia OPMO a fim de comportar a modelagem da proveniência prospectiva e possibilitar as inferências das regras de completude definidas no modelo OPM.

Com a implementação das funcionalidades acima especificadas, o SciProvMiner provê uma cobertura diferenciada dos dados de proveniência, pois engloba a captura, armazenamento e consulta tanto da proveniência prospectiva quanto retrospectiva, bem como a diminuição do trabalho de instrumentalização do workflow pelo cientista, além da possiblidade de descoberta de conhecimento útil e, não óbvio, através do processamento de consultas.

3.1. ARQUITETURA DO SCIPROVMINER

Conforme dito, o SciProv propõe um modelo de proveniência de dados e processos cujo propósito consiste em interagir com os sistemas de gerenciamento de workflows científicos utilizados em um ambiente colaborativo com a finalidade de capturar e gerir as informações de linhagem geradas (VALENTE 2011). A arquitetura do SciProvMiner possui as características apresentadas pelo SciProv e estende esta arquitetura adicionando a ela a capacidade de capturar a proveniência prospectiva e a possibilidade de se utilizar as regras de completude e inferência definidas na documentação do modelo OPM.

Para incorporar tais funcionalidades, foi realizada uma extensão do modelo OPM com suporte a captura da proveniência prospectiva, e foram expandidas as classes e propriedades da ontologia OPMO para comportarem a representação do modelo OPM estendido. Com essas modificações, o SciProvMiner fornece uma maior cobertura da proveniência coletada, possibilitando que consultas relacionadas com a especificação do workflow possam ser respondidas; permitindo que máquinas de inferência possam processar novo conhecimento, a partir das novas regras implementadas na ontologia OPMO e; permitindo a diminuição do esforço do cientista na instrumentalização do workflow, visto que com as novas regras implementadas na ontologia OPMO, determinadas dependências causais entre processos e artefatos do modelo OPM passaram a serem inferidas, não precisando ser instrumentalizadas pelo cientista. Uma representação da arquitetura do SciProvMiner, em um cenário típico da aplicação, com as respectivas contribuições ressaltadas acima em destaque, é apresentada na Figura 3.1.

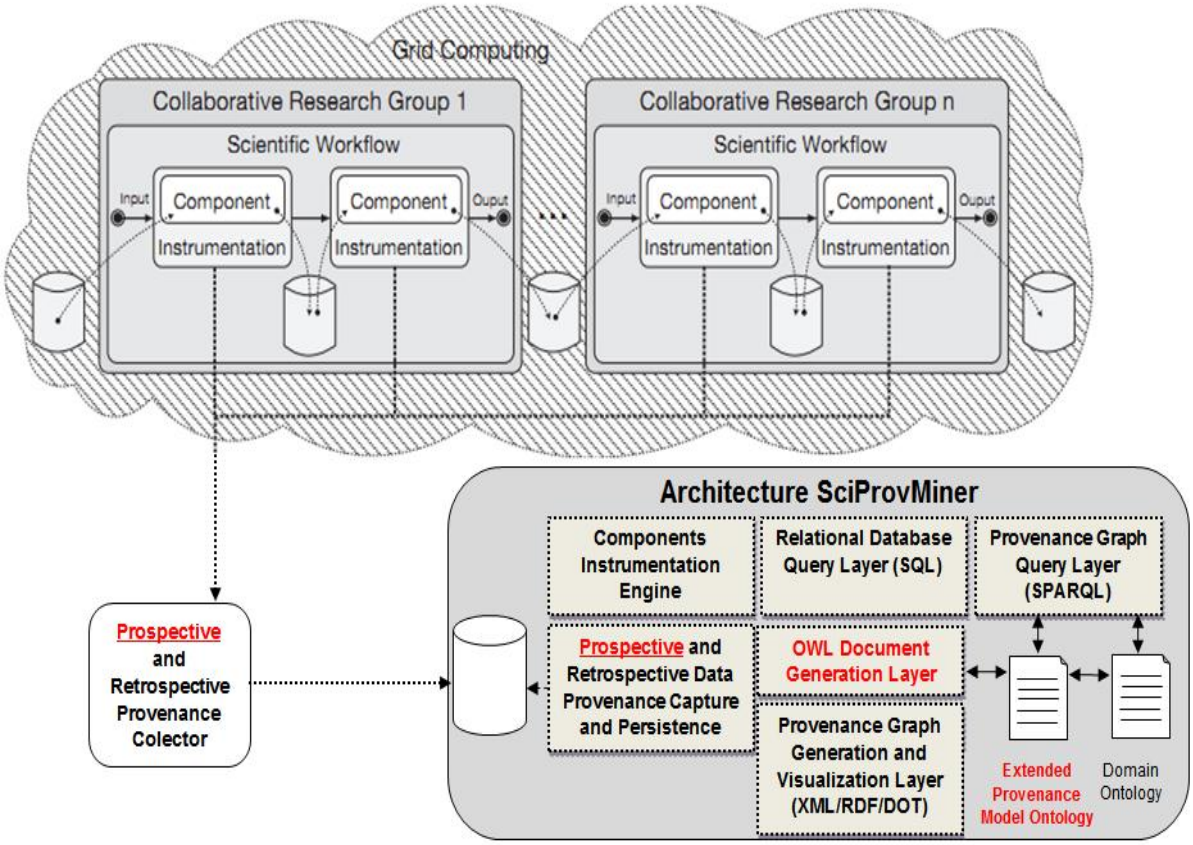


Figura 3.1. Arquitetura do SciProvMiner

Assim como o SciProv (VALENTE 2011), o SciProvMiner apresenta uma abordagem para o gerenciamento da proveniência de dados de forma independente de SGWfCs. Desta forma, os pesquisadores podem modelar workflows científicos a partir de SGWfCs distintos (como Kepler, Taverna e Vistrails) e cuja execução requer o acesso a repositórios heterogêneos (dados relacionais, semiestruturados, etc.) e distribuídos em uma grade computacional. Neste contexto, um mecanismo de instrumentalização é implementado a partir de tecnologia de serviços Web e configurado manualmente para cada componente cuja proveniência deve ser coletada (Figura 3.1). Neste cenário, o SciProvMiner especifica um mecanismo de instrumentalização para coleta da proveniência dos diversos componentes dos workflows científicos que irão conduzir o experimento colaborativo. Este mecanismo de instrumentalização tem como objetivo coletar informações geradas durante o processo de execução do workflow e enviar esses metadados para um repositório de proveniência gerenciado pelo SciProvMiner. Como diferencial do SciProvMiner em relação ao SciProv, o mecanismo de instrumentalização captura, no momento de execução do workflow, além da proveniência retrospectiva, a proveniência prospectiva. Outro diferencial nesta camada está

no fato do SciProvMiner tornar mais intuitivo o processo de instrumentalização, uma vez que para cada dependência causal que o cientista queira instrumentalizar, o *Web service* informa quais são os dados que devem ser fornecidos. As informações de proveniência capturadas, tanto retrospectivas quanto prospectivas, são persistidas em um banco de dados relacional. O SciProvMiner, a exemplo do SciProv, contém uma "camada de consulta à base de dados relacional" (*Relational Database Query Layer* na Figura 3.1) que fornece uma interface com o usuário para a realização de consultas formuladas em SQL. No entanto, o SciProvMiner amplia o espectro de consultas que podem ser respondidas em relação ao SciProv, pois torna possível ao usuário fazer consultas que dizem respeito à especificação do workflow, visto que armazena dados de proveniência prospectiva além da retrospectiva.

A "camada de geração e visualização do grafo de proveniência" (*Provenance Graph Generation and Visualization Layer* na Figura 3.1), foi implementada a partir das especificações do SciProv (VALENTE 2011). As informações de proveniência retrospectiva persistidas no banco de dados relacional, armazenadas de acordo com as especificações do modelo OPM são utilizadas como base para se obter a representação em memória do grafo de proveniência relacionada a cada execução de um workflow científico. Esta camada permite também que seja construída uma representação visual do grafo de proveniência retrospectiva gerado. O processo se dá a partir da serialização do grafo de proveniência em uma linguagem de marcação. O arquivo gerado converte o grafo em uma linguagem específica de descrição de gráficos, que produz saídas em um formato padrão de visualização.

A "camada de geração de documento OWL" (*OWL Document Generation Layer* na Figura 3.1) foi introduzida na arquitetura do SciProvMiner com o objetivo de, a partir dos dados de proveniência prospectiva e retrospectiva contidos na base de dados relacional, construir o documento OWL baseado na ontologia OPMO-e, que represente as informações de proveniência prospectiva e retrospectiva do workflow relacionado, de forma que na "camada de consulta ao grafo de proveniência" (*Provenance Query Graph Layer* na Figura 3.1), as consultas à ontologia OPMO-e, formuladas a partir da linguagem de consulta SPARQL, possam ser realizadas e máquinas de inferência possam ser aplicadas sobre esse documento a fim de efetuar deduções sobre os metadados de proveniência.

A captura da proveniência prospectiva permite ao SciProvMiner a possibilidade de modelar na ontologia OPMO-e as regras de completude definidas no modelo OPM em Moreau et al. (2011), regras estas que se propõem a encontrar componentes do modelo que não foram informadas de maneira explícita pelo usuário (artefato ou processo, dependendo da

regra) e que, se descobertas, podem aumentar o conhecimento do cientista acerca do experimento realizado. Assim, foram estendidas as classes e propriedades da ontologia OPMO para comportarem a representação do modelo OPM estendido, que captura informações de proveniência prospectiva e retrospectiva, com o objetivo de aumentar o poder de processamento semântico pelas máquinas de inferência. Além disso, esta funcionalidade de captura da proveniência prospectiva confere ao SciProvMiner a possibilidade de otimização da instrumentalização do workflow pelo cientista. Como exemplo pode ser citada a instrumentalização da dependência causal *wasGeneratedBy*, uma vez que esta pode ser inferida pela ontologia OPMO-e, com base em regras definidas por cadeia de propriedades que levam em consideração informações de proveniência prospectiva em conjunto com informações de proveniência retrospectiva.

Assim, considerando a importância da captura de todas as informações relacionadas à proveniência, e considerando que o modelo OPM, proposto em Moreau et al. (2011), prevê apenas a captura da proveniência retrospectiva, foi utilizada uma extensão do modelo OPM baseado no modelo proposto em Lim et al. (2011) que captura além da proveniência retrospectiva a proveniência prospectiva, para projetar o modelo de proveniência do SciProvMiner. O modelo de proveniência do SciProvMiner é especificado através do diagrama Entidade-Relacionamento apresentado no Anexo A.

A partir da implementação da modelagem da proveniência prospectiva é possível coletar informações da especificação do workflow, tais como sua descrição, as tarefas que fazem parte do workflow, subtarefas de uma tarefa, o sujeito que desempenhou uma tarefa bem como as portas de entrada e saída de uma tarefa, onde a porta de saída de uma tarefa pode ser conectada a porta de entrada de outra tarefa, caracterizando assim o fluxo de dados.

Considerando a captura dos dados de proveniência, o SciProvMiner utiliza uma abordagem para a captura de forma independente do Sistema Gerenciamento de Workflow Científico (SGWfC). Tendo isto por foco, foi desenvolvido um mecanismo de instrumentalização que deve ser utilizado pelos cientistas a fim de que estes venham a configurar o workflow de forma a tornar possível que as informações de proveniência deste workflow possam ser capturadas.

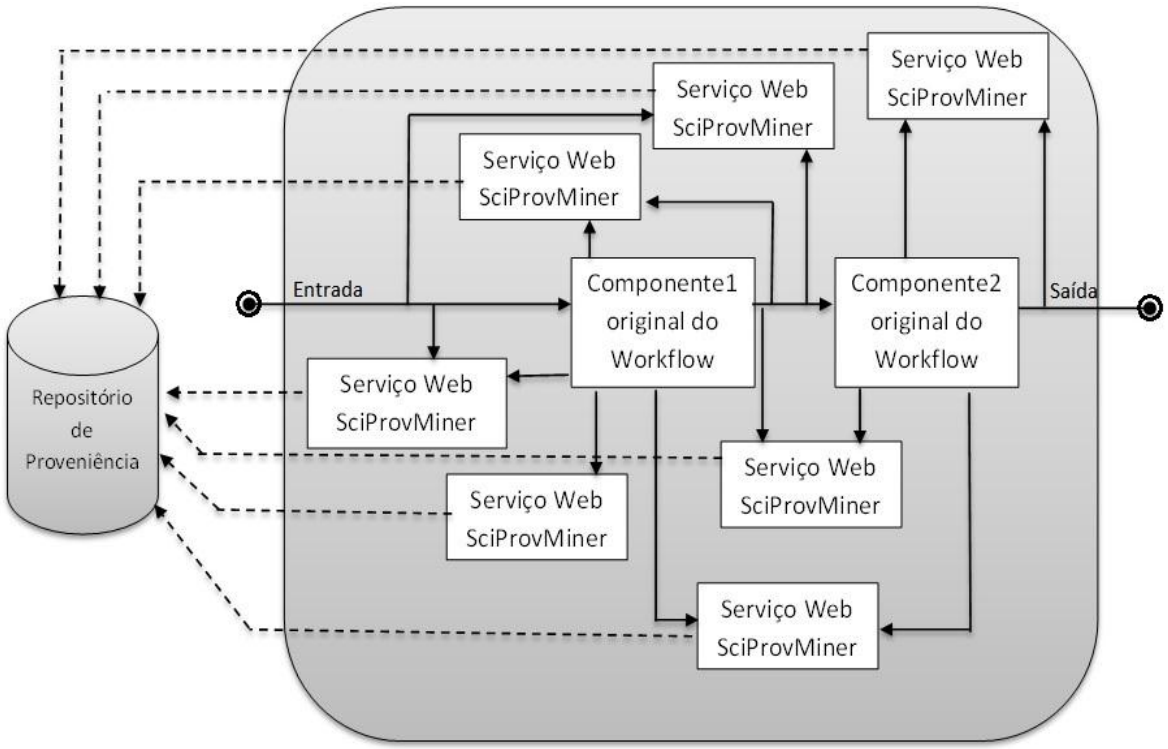


Figura 3.2. Mecanismo de Instrumentalização do SciProvMiner

O modelo de instrumentalização do SciProvMiner herda do SciProv a característica de adotar um único serviço Web para processar a captura da proveniência dos componentes do workflow, como mostrado na Figura 3.2. Porém, se diferem substancialmente no fato de que no SciProv cada instância do serviço Web captura um único componente do workflow, enquanto no SciProvMiner uma instância do serviço Web captura múltiplos componentes do workflow. Por exemplo, no SciProv, uma instância do serviço Web é instrumentalizada para a captura de um artefato, outra instância é instrumentalizada para a captura do processo e uma outra instância para a captura da dependência causal entre esse processo e esse artefato. Já no SciProvMiner, como apresentado na Figura 3.2, uma única instância do serviço Web pode capturar a dependência causal e as entidades (artefatos, processos e agentes) que se relacionam nesta dependência causal, bem como as informações de proveniência prospectiva que dizem respeito às entidades participantes daquela dependência causal. Para isto, o SciProvMiner disponibiliza, através de métodos do serviço Web, as opções disponíveis ao usuário, segundo as dependências causais do modelo OPM, para a captura da proveniência dos componentes do workflow que estão sendo instrumentalizados. São disponibilizados seis métodos de instrumentalização, sendo que cinco desses são utilizados para instrumentalizar os componentes do workflow, i.e., *wasDerivedFrom*, *wasGeneratedBy*, *wasControlledBy*, *used*,

wasTriggeredBy, e o sexto deles, o *initialConfiguration*, é utilizado para parametrizar as configurações do workflow em geral, orquestrando a captura da proveniência do workflow como um todo.

Este último método permite o controle da captura da proveniência do workflow como um todo, informando a cada instância de *Web service* instrumentalizado para a captura de proveniência, a qual workflow e a qual execução do workflow está relacionada aquela captura. Como apresentado na Figura 3.3, cada um dos serviços Web instrumentalizados estão conectados ao serviço Web de método *initialConfiguration*, que faz a orquestração da capturada da proveniência do workflow como um todo. Para cada workflow deve haver um único serviço Web de método *initialConfiguration* instrumentalizado. As demais instâncias do serviço Web devem ser instanciadas com os métodos necessários de forma a garantir a captura da proveniência prospectiva e retrospectiva de todos os elementos do workflow.

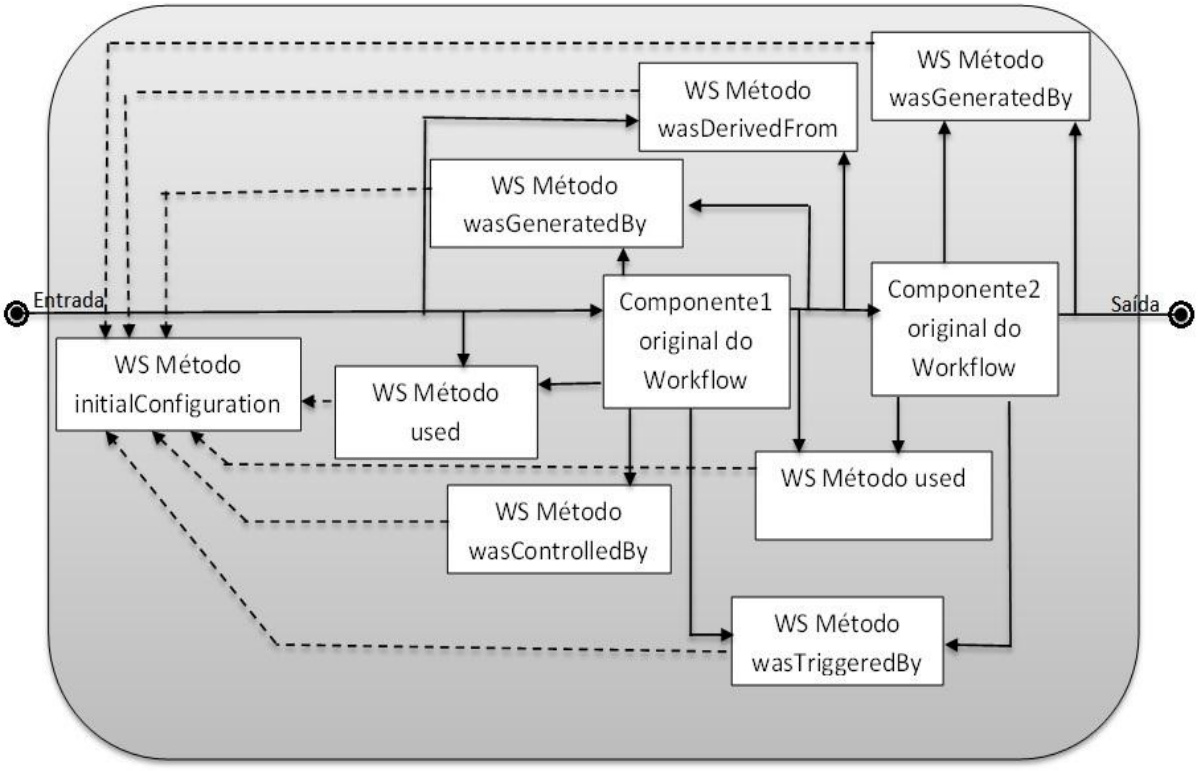


Figura 3.3. Orquestração da Captura de Proveniência pelo SciProvMiner

Outra característica importante da arquitetura do SciProv é a possibilidade de utilização das regras de completude e inferências válidas no modelo OPM, apresentadas na seção 2.1, sobre os dados de proveniência coletados pelo SciProvMiner, com o objetivo de enriquecer o conhecimento do cientista acerca da proveniência capturada, provendo informações que não foram explicitamente informadas pelo usuário. Para incorporar tais regras ao SciProvMiner,

foi realizada uma extensão do modelo OPM com suporte a captura da proveniência prospectiva, e foram expandidas as classes e propriedades da ontologia OPMO para comportarem a representação do modelo OPM estendido. As novas regras implementadas na ontologia OPMO, conforme já dito, aumentam a capacidade das máquinas de inferência em processar conhecimentos, além de permitirem a diminuição do esforço do cientista no momento da instrumentalização do workflow. Para isso, foi feita uma análise de quais seriam as alterações necessárias na arquitetura do SciProvMiner para que este cobrisse todas elas. Dentro deste cenário, foi realizada uma análise da ontologia OPMO disponível e foi detectado que com alguns ajustes da ontologia, todas as inferências em múltiplos passos poderiam ser capturadas pela ontologia, bem como a regra de completude que diz que uma aresta *wasTriggeredBy* pode ser obtida a partir da existência das arestas *used* e *wasGeneratedBy*.

No entanto, a regra de completude que esconde a introdução de processo e a regra de completude que esconde a introdução de artefato não são passíveis de serem capturadas utilizando apenas dados de proveniência retrospectiva, visto que apenas com estas informações não é possível saber qual seria o processo ou artefato a ser incluído. Para a cobertura dessas regras foram criadas na ontologia propriedades a partir do conceito de cadeia de propriedades (*property chain*), que define uma propriedade em termos de outras, utilizando informações de proveniência prospectiva e retrospectiva. Desta forma o SciProvMiner passou a ab*ranger* todas as regras de completude e inferência válidas no modelo OPM, fornecendo ao usuário ferramentas que aumentam o seu conhecimento acerca do experimento realizado. Os detalhes da implementação das regras na ontologia OPMO são apresentados na seção 3.2.2..

3.2. IMPLEMENTAÇÃO DO SCIPROVMINER

Esta seção detalha aspectos importantes relacionados à implementação da arquitetura do SciProvMiner com ênfase no emprego de softwares livre e de código aberto e considerando o objetivo da arquitetura de processar informações em um ambiente colaborativo de pesquisa que é distribuído e interconectado através de uma grade computacional. A página inicial do SciProvMiner é ilustrada na Figura 3.4.

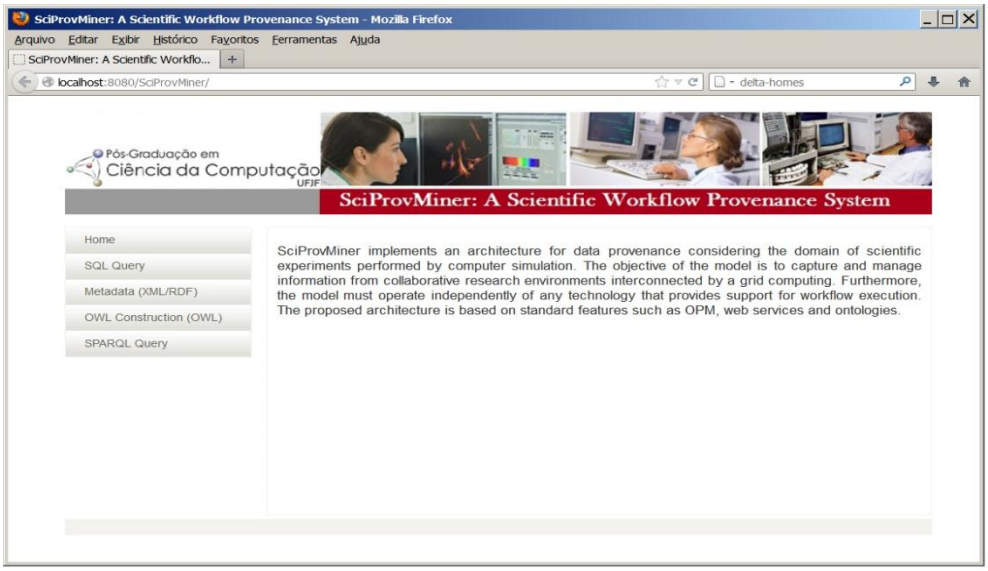


Figura 3.4. Página Inicial do SciProv

3.2.1. Implementação do Mecanismo de Captura da Proveniência

Como apresentado anteriormente, o mecanismo de captura da proveniência do SciProvMiner é implementado a partir da tecnologia de serviços Web, que torna a captura da proveniência independente do SGWfC utilizado pelo cientista. No mecanismo de coleta do SciProvMiner, uma única instância do serviço Web captura múltiplos componentes do workflow. Desta forma, cada instância do serviço Web inserida no workflow captura vários dados de proveniência ao mesmo tempo, tais como qual é a dependência causal entre as entidades do modelo OPM, de acordo com o método selecionado, as informações das entidades envolvidas na dependência causal, bem como as informações de proveniência prospectiva que dizem respeito às entidades participantes daquela dependência causal. As informações do *account*[3]em que as entidades do modelo capturadas estão inseridas e as anotações das entidades OPM relacionadas na dependência causal também podem ser informadas.

A instrumentalização de captura de proveniência do SciProvMiner foi elaborada sob um conjunto de regras de negócio que definem quais informações de proveniência cada método do serviço Web irá capturar e também, para cada método, quais são as informações que devem ser fornecidas pelo usuário, de forma a capturar as informações de proveniência previstas para aquele método. Essas informações são solicitadas através de parâmetros.

[3] Um account segundo o modelo OPM é uma descrição do grafo de proveniência em um determinado nível de detalhamento. Diferentes accounts podem coexistir em um mesmo grafo.

O Serviço Web utilizado pelo SciProvMiner disponibiliza seis métodos de instrumentalização, sendo cinco desses utilizados para instrumentalizar os componentes do workflow, segundo as dependências causais que eles estão capturando no workflow, que são *wasDerivedFrom, wasGeneratedBy, wasControlledBy, used, wasTriggeredBy* e um para orquestração do workflow como um todo, denominado *initialConfiguration*. Cada um dos métodos do serviço Web exige que um conjunto distinto de parâmetros seja configurado, para que a captura da proveniência prospectiva e retrospectiva seja realizada. Como exemplo, a instrumentalização do serviço Web para a captura da dependência causal *used* entre um artefato A e um processo P segundo o modelo OPM, solicita ao usuário as informações do artefato A e do processo P, a *role* sobre a qual o artefato A foi usado pelo processo P, bem como qual a Tarefa T do workflow ao qual o Processo P está relacionado, e por qual porta deste componente o artefato A entrou, e de qual componente e porta o artefato A foi originado. Ainda deve ser informada a qual *account* essas entidades estão relacionadas, bem como as anotações que se deseja fazer sobre as entidades do modelo OPM capturadas. Desta maneira a SciProvMiner possibilita que tanto informações de proveniência retrospectiva quanto prospectiva sejam capturadas pelo mecanismo de instrumentalização. No APÊNDICE B cada um desses métodos é detalhado, incluindo os parâmetros necessários e a importância dos mesmos no contexto da instrumentalização.

Um exemplo de workflow instrumentalizado no SGWfC Kepler é apresentado na Figura 3.5. Este workflow, denominado *SimpleAddition,* recebe duas constantes como entrada de uma tarefa de adição, faz a operação de adição das entradas e o resultado desta operação é recebido por uma tarefa que tem a função de retornar o valor absoluto do dado entrado. Neste exemplo, o workflow encontra-se plenamente instrumentalizado, de forma que a instanciação dos serviços Web com seus respectivos métodos garantem que a captura da proveniência prospectiva e retrospectiva seja realizada para todos os elementos do workflow, quando este for executado. Para sua completa instrumentalização foram necessárias três instâncias de serviço Web com o método *Used,* uma para cada vez que um artefato é usado por um processo, três instâncias com o método *wasDerivedFrom,* uma para cada artefato que é derivado de outro, duas com o método *wasTriggeredBy* e outras duas com o método *wasGeneratedBy*, ambos os métodos instrumentalizados para cada um dos processos presentes no workflow e uma com o método *initialConfiguration*.

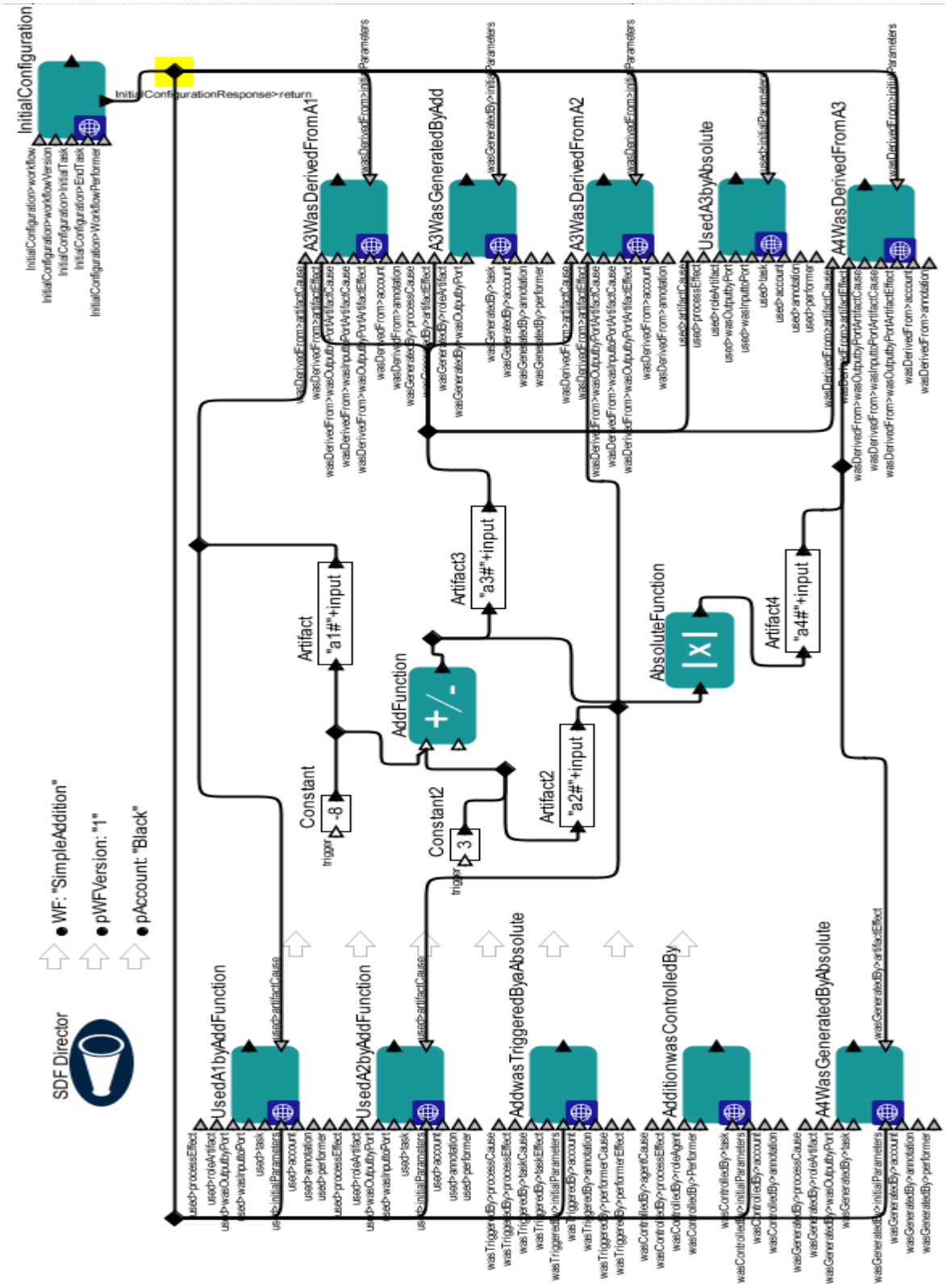


Figura 3.5. Exemplo de workflow Instrumentalizado

Como a abordagem utilizada pelo SciProvMiner para captura da proveniência de dados do workflow tem a característica de ser independente do SGWfC e é realizada a partir de

serviços Web, a captura da proveniência prospectiva, que diz respeito à especificação do workflow, também é realizada em tempo de execução. Desta forma, caso aconteça algum erro durante a execução do workflow, a captura dos dados referente à especificação daquele workflow pode ficar comprometida. Com o objetivo de ser possível descobrir se um workflow teve sua especificação capturada como um todo, ou se ocorreu um erro durante a captura da proveniência, o workflow alvo da coleta de proveniência deve ter uma tarefa final, que seja executada uma única vez a cada vez que o workflow é rodado, para que o SciProvMiner possa capturar essa informação e saber que, se chegou a colher os dados de proveniência desta atividade, o workflow completou a sua execução. Esta então, é outra regra de negócio considerada pelo SciProvMiner que o cientista precisa adaptar o workflow para que seja cumprida, caso o workflow trabalhado não atenda a esta necessidade do SciProvMiner. Esta informação é solicitada no SciProvMiner através do método *initialConfiguration*, no parâmetro *EndTask*.

Também devido à independência em relação ao SGWfC da solução proposta pelo SciProvMiner para a coleta de proveniência, quando a especificação do workflow é alterada em relação à última vez em que o workflow instrumentalizado para a captura da proveniência tiver sido executado, o cientista precisa informar essa alteração através do parâmetro denominado *version*, no método *initialConfiguration*. A manutenção desta informação pelo cientista é de grande importância, visto que, caso a captura da proveniência prospectiva de alguma versão do workflow não tenha sido realizada com êxito, e exista na base de dados uma captura completa da especificação daquele workflow na mesma versão, esta especificação completa pode ser associada à execução realizada com erro deste workflow na mesma versão para que o usuário possa ter resposta a questões como a nona questão do *Third Provenance Challenge*, que questiona quais passos deixam de ser realizados quando ocorre um erro no workflow.

Conforme dito, todo workflow instrumentalizado precisa ter uma instância do serviço Web configurado com o método *initialConfiguration*, como apresentado na Figura 3.5, responsável por gerenciar a execução da captura da proveniência do workflow como um todo, informando para as outras instâncias de serviço Web instrumentalizadas qual é o workflow e a qual execução do workflow se refere a linhagem que está sendo capturada, de forma a garantir que, se a proveniência de diversos workflows estiver sendo capturadas simultaneamente, a coleta das informações de proveniência de cada workflow será consistida independentemente, e, se

um mesmo workflow for executado diversas vezes, cada execução terá a sua captura de proveniência identificada e consistente, sem misturar os dados de execuções diferentes.

É importante também destacar, que o método *used* possui os parâmetros *wasOutputByPort* e *wasInputToPort* (APÊNDICE B), e nesses parâmetros estão contidas as informações de qual componente do workflow e porta de saída o artefato que está sendo usado pelo processo na dependência causal *used* está vindo, e por qual porta de entrada da tarefa o artefato foi consumido, tarefa esta associada ao processo que é o efeito da dependência causal. Com isso, é possível capturar a informação sobre o fluxo de dados do workflow, que se dá pela conexão de portas das tarefas do workflow. Esta informação é muito importante por fazer parte da especificação do workflow, e também porque ela é utilizada para a construção das regras de diversas propriedades que foram incluídas na ontologia OPMO com a finalidade de aumentar o poder das máquinas de inferência em produzir conhecimento novo.

Para a captura da proveniência retrospectiva, a instrumentalização do workflow vai depender do grau de granularidade, ou seja, do nível de detalhamento que o cientista deseja ter na captura dos dados proveniência. Por exemplo, se o cientista desejar capturar apenas as dependências causais entre processos, dada pela aresta *wasTriggeredBy* segundo o modelo OPM, ele pode apenas instrumentalizar o workflow com instâncias de serviço Web de método *wasTriggeredBy* do SciProvMiner. No entanto, se o cientista desejar obter um maior detalhamento da captura da proveniência, o que confere mais informações acerca da proveniência, para cada parâmetro de entrada do workflow, deve ser especificada uma instância do serviço Web do SciProvMiner com o método *used* (APÊNDICE B). Além disso, para cada dado gerado pelo workflow, deve ser instrumentalizado um serviço Web com método *wasGeneratedBy*, para cada dado que foi gerado a partir de outro, deve ser instrumentalizado um serviço Web com o método *wasDerivedFrom*; para cada agente que controla ou catalisa um processo, deve ser instrumentalizado um serviço Web com o método *wasGeneratedBy*; e para capturar cada dependência causal entre processos deve ser instrumentalizado um serviço Web com o método *wasTriggeredBy*.

Na captura da proveniência prospectiva, para que a especificação do workflow seja plenamente capturada, é importante garantir que todos os dados de entrada, atividades e portas das atividades do workflow pelas quais os parâmetros se comunicam com as atividades, tenham sido instrumentalizados. Sabendo que todos os métodos do serviço Web do SciProvMiner que capturam artefato também capturam as portas e as atividades do workflow, e que todos os métodos do serviço Web do SciProvMiner que capturam processos também

capturam as atividades e os *performers*, e que o método *used* do serviço Web do SciProvMiner captura o fluxo de dados entre atividades do workflow, o cientista pode trabalhar e criar sua própria maneira de instrumentalizar o workflow de forma a capturar a especificação deste como um todo.

Porém, para facilitar o processo de instrumentalização do cientista, foram definidos passos básicos que garantam que toda a especificação do workflow seja instrumentalizada pelos serviços Web do SciProvMiner. É recomendável que se instrumentalize uma instância do serviço Web com o método *used* para cada parâmetro de cada atividade do workflow. Desta forma fica garantido que todos os dados de entrada, todas as atividades e o fluxo de dados pelo workflow serão capturados. No entanto, para os dados que foram gerados, mas não foram consumidos por outras atividades do workflow, deve ser definida uma instância do serviço Web com o método *wasGeneratedBy* para cada um desses casos, visto que estes não são capturados pelas instâncias do serviço Web com o método *used*.

3.2.2. Extensão da ontologia OPMO

Uma das principais características do SciProvMiner está no emprego de tecnologias da Web Semântica, que lhe confere a possibilidade de processar consultas a partir de máquinas de inferência, capazes de efetuar deduções sobre essas bases de conhecimento e obter resultados importantes ao extrair informações adicionais além daquelas que se encontram registradas de forma explícita nos grafos de proveniência.

Em Moreau et al. (2011) é definida uma ontologia para capturar os conceitos da versão 1.1 do modelo OPM, denominada *Open Provenance Model Ontology* (OPMO). A Figura 3.6 ilustra a ontologia OPMO original desenhada no *plugin* ProvViz da ferramenta Protégé na versão 4.02 e o APÊNDICE C detalha as principais classes e restrições.

Na figura 3.6, as classes estão representadas por círculos, sendo que os círculos de cor laranja representam as classes equivalentes e aqueles de cor amarela representam as classes definidas. As propriedades que conectam indivíduos das classes estão representadas nas arestas. A partir desse conjunto de propriedades de objeto são modelados os relacionamentos entre os indivíduos das classes da ontologia OPMO original.

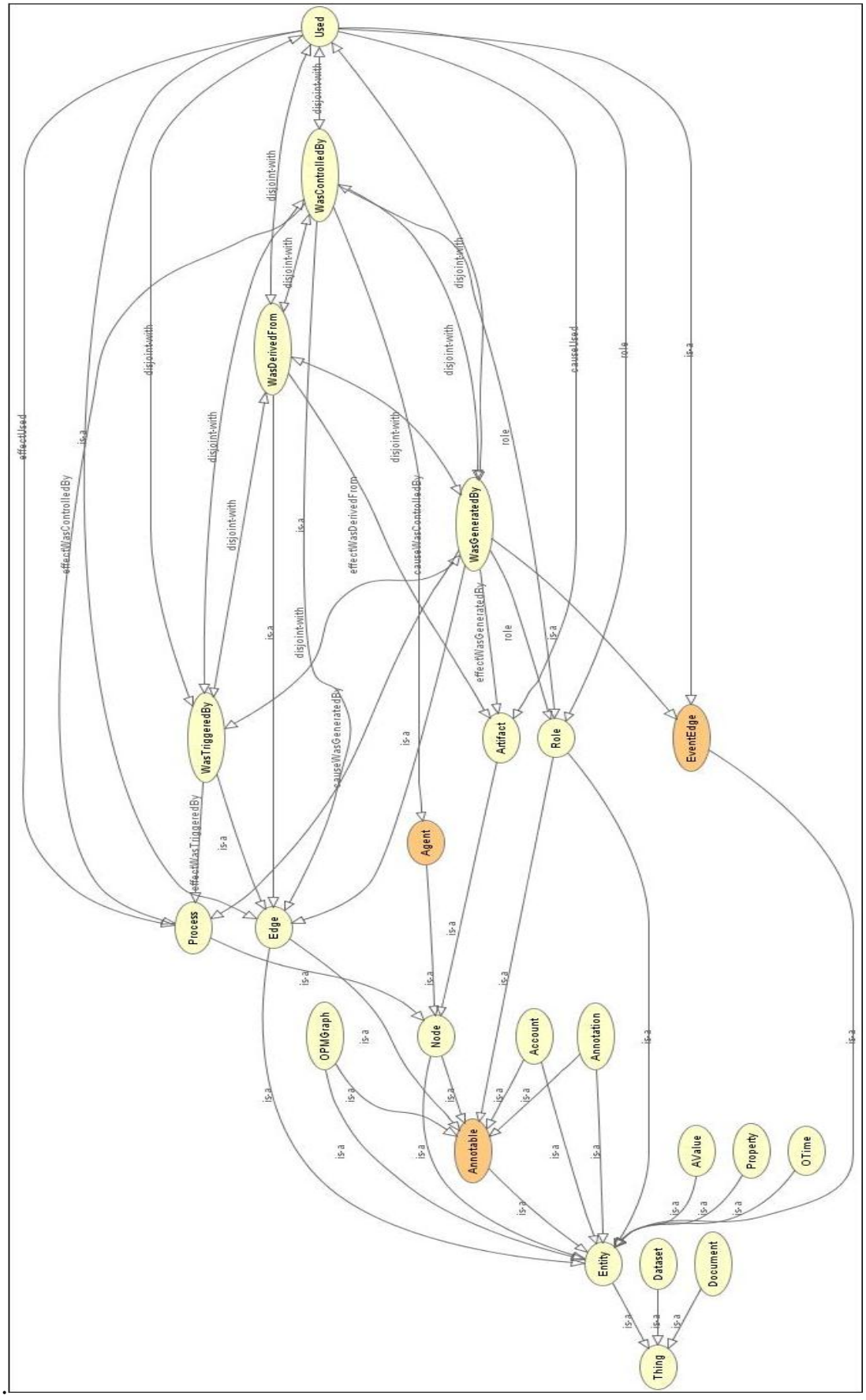


Figura 3.6. Ontologia OPMO original

3.2.2.1.Criação de novas propriedades na OPMO

Considerando as restrições expressas na ontologia OPMO original (APÊNDICE C) pode ser observado que cada dependência causal segundo o modelo OPM entre um nó e uma aresta é modelada através de duas propriedades, uma representando a causa da dependência causal e outra representando o efeito. Conforme o exemplo da Figura 3.7, para modelar a dependência causal *used*, entre o Artefato instanciado como "*Artifact_a1_value-8_Fire0*" e o processo P instanciado como "*ProcessAdd_Fire0"* (*ProcessAdd_Fire0usedArtifact_a1_value-8_Fire0*), é necessário instanciar um indivíduo da classe *Used*, denominado no exemplo "*Used*_add_a1_operator1_Fire0" e acrescentar a este indivíduo i) o relacionamento com a propriedade *causeUsed*, afirmando que "*Used_add_a1_operator1_Fire0causeUsedArtifact_a1_value-8_Fire0*", e ii) o relacionamento com a propriedade *effectUsed* através da afirmação "*Used*_add_a1_operator1_Fire0 *effectUsed Used*_add_a1_operator1_Fire0". O relacionamento com a propriedade *role* para esta dependência causal é obrigatório segundo a definição do modelo OPM em Moreau et al. (2011), e é modelado com as restrições da classe *WasGeneratedBy* através da restrição "role some role", e por isso também é necessária ser afirmada para a aresta *Used*_add_a1_operator1_Fire0, conforme exemplo da Figura 3.7.

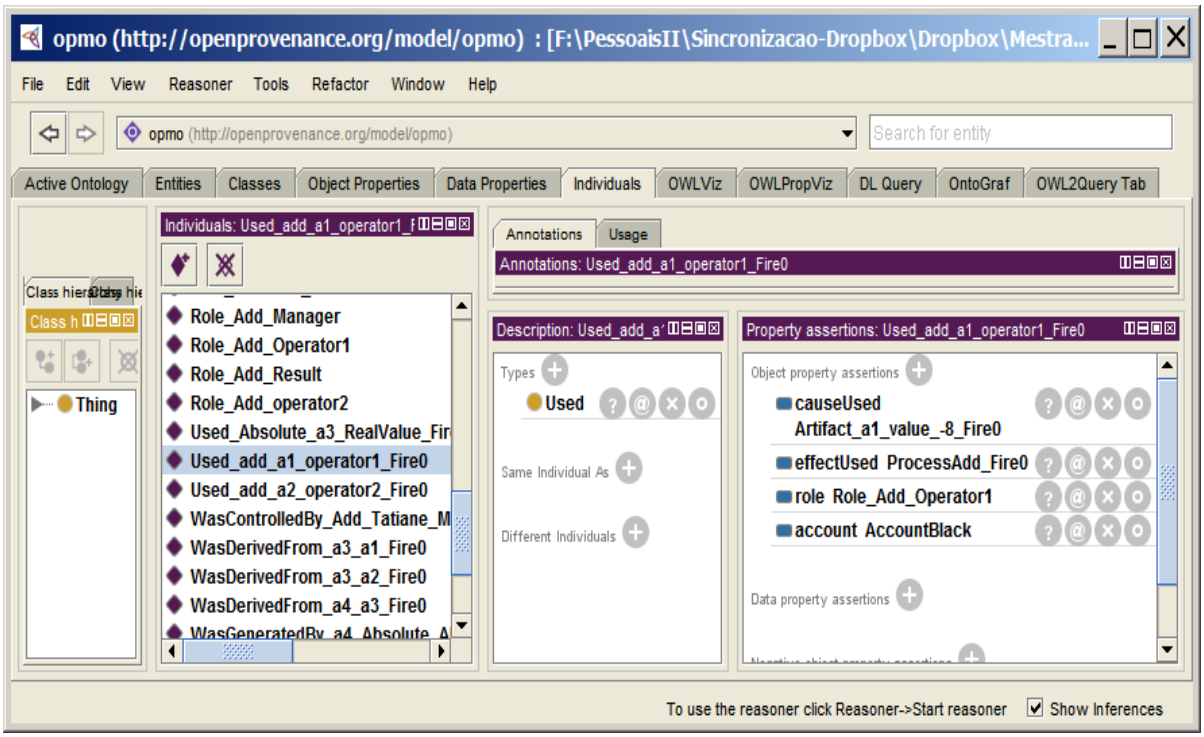


Figura 3.7. Exemplo de instanciação de um indivíduo da classe *used* e seus relacionamentos com outros indivíduos da ontologia OPMO através de propriedades de objetos.

Com o objetivo de modelar o relacionamento direto entre as entidades do modelo OPM relacionadas nas dependências causais, do tipo (A *used* P), sem a necessidade de informar explicitamente este relacionamento, visto que ele está implícito através de outras propriedades já definidas na ontologia OPMO, e que um dos principais benefícios de se ter uma ontologia é que ela é apta a ajudar a encontrar fatos implícitos, especialmente aqueles que não estão tão perceptíveis para o usuário (YU 2011), é utilizado o recurso introduzido na OWL 2 denominado cadeia de propriedades (*property chain*s) que fornece uma maneira de definir uma propriedade em termos de uma cadeia de propriedades que conectam recursos.

As seguintes propriedades apresentadas na Tabela 3.1definidas a partir de cadeia de propriedades foram inseridas na ontologia OPMO com o objetivo de inferir o relaciomento entre as entidades do modelo OPM e as dependências causais de forma direta.

Tabela 3.1:*Object properties* construídas a partir de *property chain* na ontologia OPMO original

Nome	***Domain***	***Range***	***Property chain***
used	*Process*	*Artifact*	*effectUsedInverse o causeUsed subproperty of used*
wasDerivedFrom	*Artifact*	*Artifact*	effect*WasDerivedFrom*Inverse o cause*WasDerivedFrom* subproperty of *wasDerivedFrom*
wasGeneratedBy	*Artifact*	*Process*	*effectWasGeneratedByInverse o causeWasGeneratedBy subproperty of wasGeneratedBy*
wasTriggeredBy	*Process*	*Process*	effect*WasTriggeredBy*Inverse o cause*WasTriggeredBy* subproperty of *wasTriggeredBy*

Como a cadeia de propriedades é formada pela conexão de propriedades, o domínio da primeira propriedade da cadeia tem que ser o mesmo domínio da propriedade que está sendo formada. O domínio de uma propriedade que está conectada a outra tem que ser da mesma classe do *range* da propriedade que a antecede na cadeia, sendo que a última propriedade da cadeia tem que ter por *range* a mesma classe que é o *range* da propriedade que está sendo criada.

A Figura 3.8 ilustra a formação da propriedade *used* através da cadeia de propriedade "*effectUsedInverse o causeUsed*". Nesta figura, a propriedade que está sendo criada tem por domínio a classe *Process* e por *range* a classe *Artifact*. A primeira propriedade da cadeia é a *effectUsedInverse* que tem por domínio *Process*, que é o mesmo domínio da propriedade que está sendo formada, e por *range* a classe *Used*. A segunda e última propriedade da cadeia é a *causeUsed*, que tem por domínio a classe *Used*, ou seja a mesma classe que a propriedade que

a antecede tem por *range*, e tem por *range* a classe *Artifact*, que é a classe do *range* da propriedade *used* que está sendo modelada.

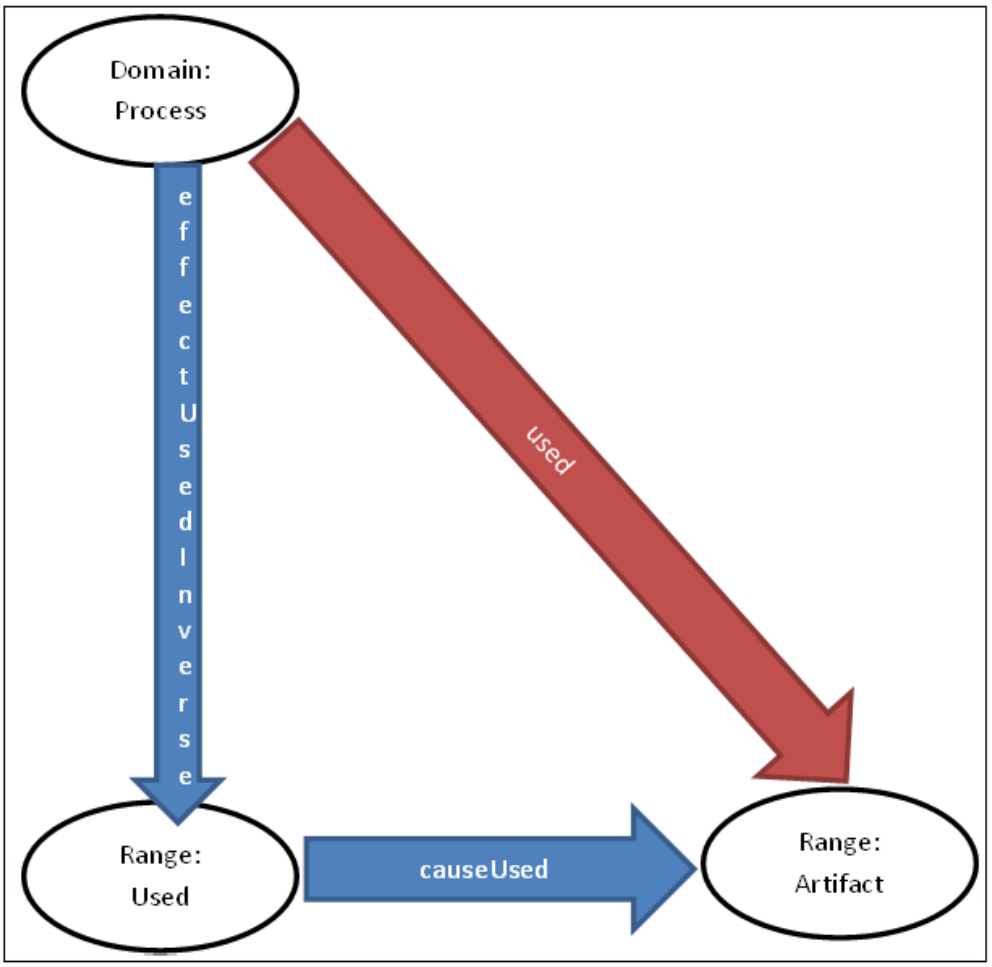


Figura 3.8. Ilustração da *property chain* criada para a definição da propriedade *used*

Cadeia de propriedades é uma característica muito útil introduzida na OWL 2. Ela pode conectar a quantidade de recursos que forem necessários para a definição de uma propriedade. Porém, uma importante restrição no uso de *property chain* é que elas podem ser utilizadas somente como parte de um relacionamento do tipo sub-propriedade e só podem aparecer na posição de sub-propriedades de tal relacionamento (HEBELER et al. 2009). Por isso, nas propriedades definidas na tabela 3.1 todas elas aparecem com o relacionamento "*subpropertyOf*" da propriedade que está sendo criada. Como consequência elas não podem ser utilizadas em nenhum outro relacionamento de propriedades tal como relacionamento inverso, ou equivalente, ou disjunto.

A Figura 3.9 mostra que, quando a máquina de inferência é aplicada sobre a ontologia, é inferido o conhecimento do relacionamento entre o indivíduo *ProcessAdd_Fire0* da classe *Process* e o indivíduo *Artifact_a1_value-8_Fire0* da classe*Artifact* na propriedade *used*.

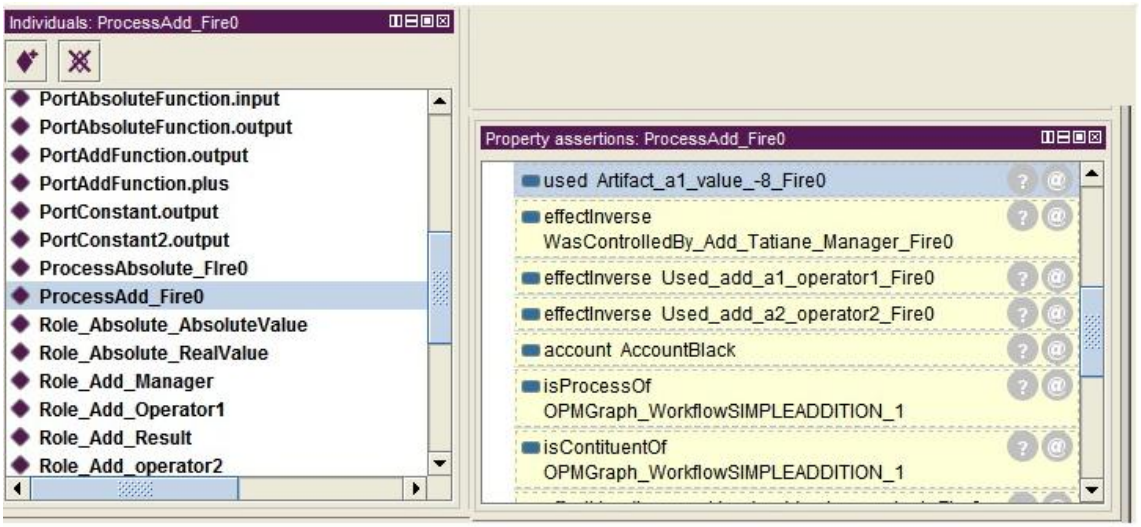


Figura 3.9. Confirmação da inferência da propriedade *used* entre indivíduos da classe *Process* e *Artifact* na ferramenta Protégé 4.2

Como a ontologia OPMO original modela apenas os conceitos de proveniência retrospectiva e o SciProvMiner trabalha também com a captura da proveniência prospectiva, foi realizada uma extensão da ontologia OPMO para que esta viesse a capturar os conceitos de proveniência prospectiva. Além disso, a partir de uma análise detalhada da Ontologia OPMO original, pôde-se constatar que esta não captura todas as inferências e regras de completude definidas na documentação do modelo OPM. Assim, foram implementadas as modificações necessárias nesta ontologia para garantir que todas as regras de completude e inferências válidas no modelo OPM fossem contempladas na ontologia OPMO-e de forma a enriquecer o conhecimento do cientista a respeito do experimento que está sendo modelado, dando-lhe maior subsídio para tomada de decisão em relação ao experimento.

Detalhamos a seguir as alterações na ontologia OPMO para modelar a proveniência prospectiva e seus relacionamentos com a proveniência retrospectiva. As classes que representam a proveniência prospectiva foram adicionadas como subclasse de *ProspectiveEntity*, como ilustrado na Figura 3.10. As classes modeladas foram *WFModel*, *Workflow*, *Performer*, *Component*, tendo *Task* como subclasse, e *Port*, tendo como subclasses *DestinationPort* para representar a porta que é conectada à outra como porta destino, sendo que uma porta destino sempre é uma porta de entrada de um componente, por isso a

equivalência com a classe *InputPort* e *SourcePort* que representa a porta que é conectada a outra como porta de onde o artefato está vindo, ou seja, a porta de saída de um componente, por isso a equivalência desta classe com a classe *OutputPort*.

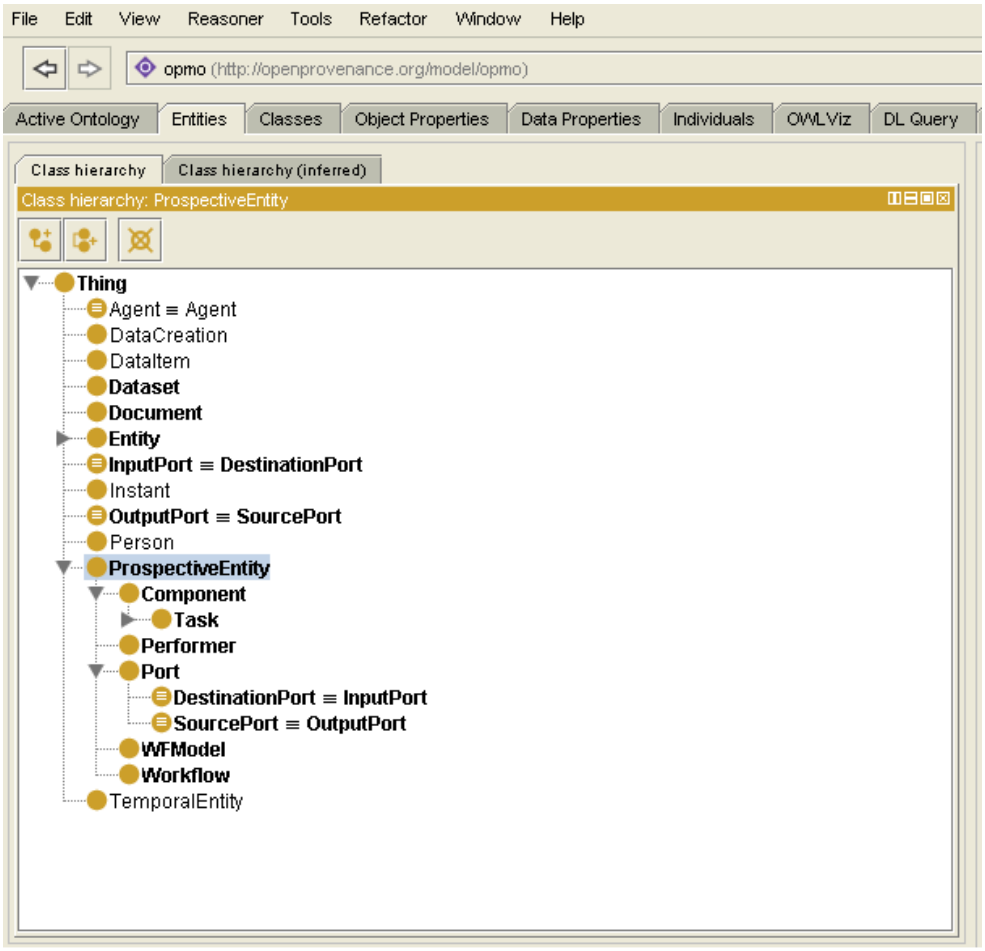


Figura 3.10. Inclusão das Classes referentes a proveniência prospectiva na ontologia OPMO

Algumas *object properties* foram criadas para relacionar indivíduos da proveniência prospectiva, a fim de modelar o relacionamento entre indivíduos desta natureza. A tabela 3.2 apresenta as *object properties* adicionadas para este fim. A propriedade *wfModelHasWorkflow* modela o relacionamento *WFModelHasWorkflow* do modelo E-R (APÊNDICE A) do SciProvMiner. Esta propriedade tem por domínio a classe *WFModel* e como *range* a classe *Workflow*. Ela tem a característica de ser inversa funcional e tem como inversa a propriedade *WorkflowIsModelOf*. A propriedade *hasComponent* representa o relacionamento entre os indivíduos das classes *Workflow* e *Component*, estando a primeira na posição de domínio e a segunda na posição de *range*, e possui a característica de ser inversa funcional. Esta propriedade modela o relacionamento *PartOf* entre *Workflow* e *Component* no esquema E-R do SciProvMiner e possui como inversa a propriedade *isComponentOf*. A propriedade

hasParent relaciona indivíduos da classe *Task* para representar que o indivíduo desta classe que está no domínio da propriedade tem o relacionamento de pai com o indivíduo desta mesma classe que está no *range* da propriedade. Esta propriedade modela o relacionamento *contains* do modelo E-R do SciProvMiner e tem por características ser funcional, assimétrica e irreflexiva, além de possuir como propriedade inversa a propriedade *hasChild*. A propriedade *performs* relaciona indivíduos da classe *Performer* com indivíduos da classe *Task* através da sub-propriedade *performsTask* e com indivíduos da classe Workflow através da subpropriedade *performsWorkflow*. Estas sub-propriedades possuem como inversas as propriedades *taskHasPerformer* e *workflowHasPerformer* respectivamente e modelam os relacionamentos *hasPerformer* e *performers* do esquema E-R. A propriedade *IsConnectedTo*, que modela o relacionamento de mesmo nome no diagrama E-R do SciProvMiner, relaciona indivíduos da classe *Port*, sendo que a subpropriedade *likeDestinationPort* relaciona a conexão da porta destino à porta origem, e subpropriedade *likeSourcePort* o inverso. Essas duas últimas propriedades possuem a característica de serem irreflexivas.

Tabela 3.2: *Object properties* adicionadas à ontologia OPMO para relacionar indivíduos da proveniência prospectiva

Nome	***Domain***	***Range***	**Características**	**Inversa**
wfModelhasWorkflow	*WFModel*	Workflow	Inversa Funcional	WorkflowIsModelOf
hasComponent	Workflow	Component	Inversa Funcional	isComponentOf
hasParent	*Task*	*Task*	Funcional, Assimétrica, Irreflexiva	hasChild
Performs	Performer			hasPerformer
performsWorkflow	Performer	Workflow		WorkflowHas Performer
performsTask	Performer	*Task*		TaskHas Performer
isConnectedTo	*Port*	*Port*		
likeDestinationPort	Destination *Port*	SoucePort	Irreflexiva	likeSourcePort
hasPort	Component	*Port*		isPortOf
componentHasInputPort	Component	InputPort		isInputPortOf
componentHasOutputPort	Component	OutputPort		IsOutputPortOf

A propriedade *hasPort*, que modela o relacionamento *PortOf* do esquema E-R, possui a classe *Component* como *domain* e *Port* como *range*, e objetiva relacionar um indivíduo do tipo *Component* com um indivíduo do tipo *Port*, afirmando que o indivíduo da classe *Port* possui a propriedade *hasPort* com o valor que é um indivíduo da classe *Port*. A inversa dessa propriedade é a propriedade *isPortOf*. A propriedade *hasPort* possui as sub-propriedades *componentHasInputPort* e *componentHasOutputPort*, e suas propriedades inversas são *isInputPortOf* e *isOutputPortOf* respectivamente.

Tabela 3.3: *Object properties* adicionadas à ontologia OPMO para relacionar indivíduos de proveniência prospectiva com indivíduos de proveniência retrospectiva.

Nome	***Domain***	***Range***	**Características**	**Inversa**
processHasTask	*Process*	*Task*	Funcional	isTaskOf
agentHasPerformer	*Agent*	Performer	Funcional	isPerformerOf
hasWorkflow	OPMGraph	Worflow	Funcional	isWorkflowOf
wasOutputByPort	*Artifact*	OutputPort	Funcional	*wasOutputByPort*Inverse
wasInputToPort	*Artifact*	Inputport		*wasInputToPort*Inverse

Também foi necessária a inclusão de propriedades na ontologia OPMO que modelassem o relacionamento entre os indivíduos da proveniência prospectiva com indivíduos da proveniência retrospectiva. A Tabela 3.3 ilustra as propriedades adicionadas com este fim. As propriedades *processHasTask*, *agentHasPerformer* e *hasWorkflow* modelam os relacionamento *InstanceOf* do esquema E-R do SciProvMiner entre as entidades *Process* e *Task*, *Agent* e *Performer*, *OPMGraph* e *Workflow* respectivamente, tendo como *domain* as classes de proveniência retrospectiva e como *range* as classes de proveniência prospectiva relacionadas nas propriedades. Elas possuem a característica de serem funcionais. As propriedades inversas de cada uma dessas citadas são respectivamente *isTaskOf*, *isPerformerOf* e *isWorkflowOf*. As propriedades *wasOutputByPort* e *wasInputToPort* modelam os relacionamentos *OutputToPort* e *InputToPort* do modelo E-R do SciProvMIner, e relacionam indivíduos da classe *Artifact* com indivíduos da classe *OutputPort* na propriedade *wasOutputByPort* e relacionam indivíduos da classe *Artifact* com indivíduos da classe *InputPort* na propriedade *wasInputToPort*. As inversas dessas propriedades são respectivamente *wasOutputByPortInverse* e *wasInputToPortInverse*.

Devido à característica das cadeias de propriedades formarem propriedades a partir de outras, e pelo fato do *range* de uma propriedade que terá outra conectada a ela ter que ser do mesmo tipo do *domain* da propriedade que está sendo conectada, foi necessária a criação de algumas propriedades inversas daquelas já definidas na ontologia OPMO original, para ser possível criar as cadeias de propriedades. A Tabela 3.4 apresenta todas as propriedades que foram adicionadas a ontologia OPMO que são inversas de propriedades que constam na ontologia OPMO original.

As propriedades adicionadas foram isContituentOf inversa de hasContituent, isContituentAgentOf inversa de hasAgent subpropriedade de isContituentOf, isAccountOf inversa de hasAccount subpropriedade de isContituentOf, isArtifactOf inversa de hasArtifact e subpropriedade de isContituentOf, isDependencyOf inversa de hasDependency e subpropriedade de isContituentOf e isProcessOf inversa de hasProcess e subpropriedade de isContituentOf.

Tabela 3.4: *Object properties* adicionadas à ontologia OPMO para relacionar indivíduos de proveniência retrospectiva.

Nome	*Domain*	*Range*	Características	Inversa
isContituentOf		OPMGraph		hasContituent
isContituentAgentOf	*Agent*	OPMGraph		hasAgent
isAccountOf	Account	OPMGraph		hasAccount
isArtifactOf	*Artifact*	OPMGraph		hasArtifact
isDependencyOf	Edge	OPMGraph		hasDependency
isProcessOf	*Process*	OPMGraph		hasProcess

Com base nas propriedades de objeto criadas, foram adicionadas restrições a algumas classes da ontologia OPMO, de forma a garantir uma melhor representação das informações. A Tabela 3.5 ilustra as classes, com suas superclasses e as restrições adicionadas a algumas dessas classes. A classe *ProspectiveEntity*, possui como subclasses *WFModel*, *Workflow*, *Component*, *Performer* e *Port*, que são disjuntas entre si e são classes primitivas. A classe *Task* é subclasse de *Component* e também é uma classe definida. As classes primitivas *SourcePort* e *DestinationPort* são subclasses de *Port* e são equivalentes às classes *OutputPort* e *InputPort* respectivamente. Elas não são disjuntas entre si, ou seja, uma porta pode ser classificada como *SourcePort* e *DestinationPort* simultaneamente.

Tabela 3.5: Classes adicionadas à ontologia OPMO.

Nome	**Superclasse**	**Restrições adicionadas**
ProspectiveEntity	Thing	
WFModel	ProspectiveEntity	
Workflow	ProspectiveEntity	workflowIsModelOf some *WFModel* workflowHasPerformer some Performer
Component	ProspectiveEntity	isComponentOfsome Workflow
Task	ProspectiveEntity →Component	
Performer	ProspectiveEntity	
Port	ProspectiveEntity	isPortOf some Component
SourcePort	ProspectiveEntity →*Port*	isOutputPortOf some Component
DestinationPort	ProspectiveEntity →*Port*	isInputPortOf some Component
NoUserInput	Entity → Node →*Artifact*	*Artifact* and (*wasGeneratedBy* some *Process*)

A classe *NoUserInput*, que é uma subclasse de *Artifact*, que faz parte da representação da proveniência retrospectiva é uma classe definida, ou seja, os indivíduos que fazem parte desta classe são inferidos pela máquina de inferência a partir da restrição *Artifact* and (*wasGeneratedBy some Process*). Esta classe infere os artefatos que não foram fornecidos por indivíduos (os parâmetros, por exemplo), ou seja, infere os artefatos que foram gerados pelas tarefas do workflow. Na tentativa de inferir os artefatos que foram fornecidos por indivíduos, como sendo o complemento dos artefatos gerados pelas tarefas do workflow poderia ser definida uma classe *UserInput* sob a restrição *Artifact and (not* (*NoUsersInput*)). No entanto, devido ao "Raciocínio de Mundo Aberto" (*Open World Reasoning*) que é utilizado pelas máquinas de inferência, onde é considerado que não se pode assumir que alguma coisa não existe até que seja explicitamente afirmado que ela não existe, pois o conhecimento pode simplesmente ainda não ter sido adicionado à base de conhecimento (HEBELER 2009), essa classe *UserInput* não pode ser definida sob esta restrição.

A Figura 3.11 apresenta as classes apresentadas na tabela 3.5 e as propriedades apresentadas nas Tabelas 3.2 e 3.3.

Para melhorar a definição das classes *Process* e *Agent* na ontologia OPMO-e, que considera a proveniência prospectiva, foram adicionadas a essas classes primitivas as restrições definidas na tabela 3.6.

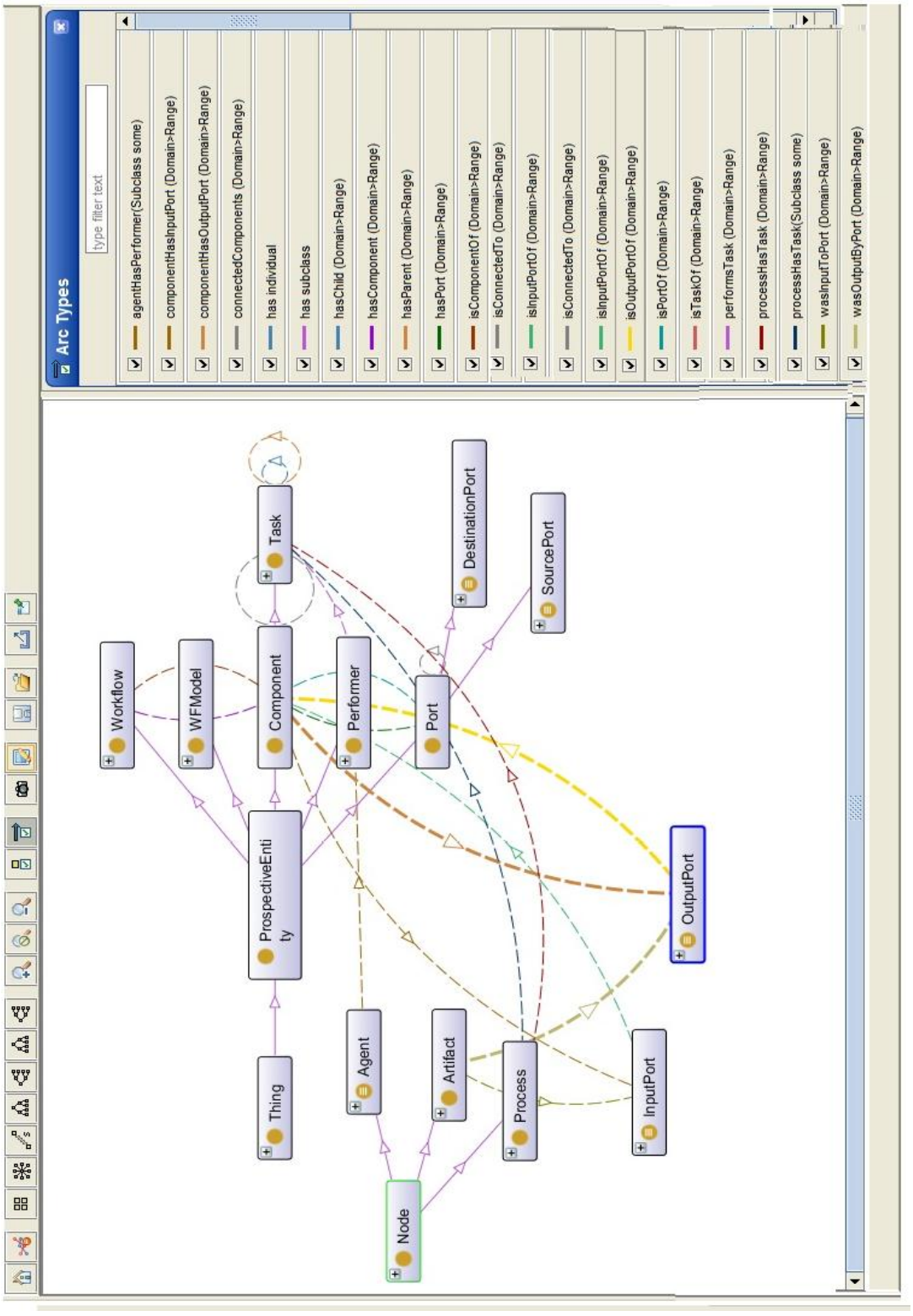


Figura 3.11. Classes da proveniência prospectiva adicionadas à ontologia OPMO e os relacionamentos entre essas classes através das propriedades apresentadas nas Tabelas 3.2 e 3.3.

Tabela 3.6: Restrições adicionadas a classes já existentes na ontologia OPMO original.

Nome	Superclasse	Restrições adicionadas
Process	Entity → Node →*Process*	processHasTask some *Task*
Agent	Entity → Node →*Agent*	agentHasPerformer some Performer

As restrições adicionadas às classes *Process* e *Agent* se referem ao relacionamento *InstanceOf*, do esquema E-R do APÊNDICE A, entre essas entidades (*Process* e *Agent*) que modelam a proveniência retrospectiva e as entidades da proveniência prospectiva *Task* e *Performer*, respectivamente, e afirmam que um indivíduo da classe *Process* é uma instância de execução de um indivíduo da classe *Task* e que um indivíduo da classe *Agent* tem associado a ele um indivíduo da classe *Performer*.

3.2.2.2. Implementação das regras de completude e inferência definidas no modelo OPM na ontologia OPMO-e

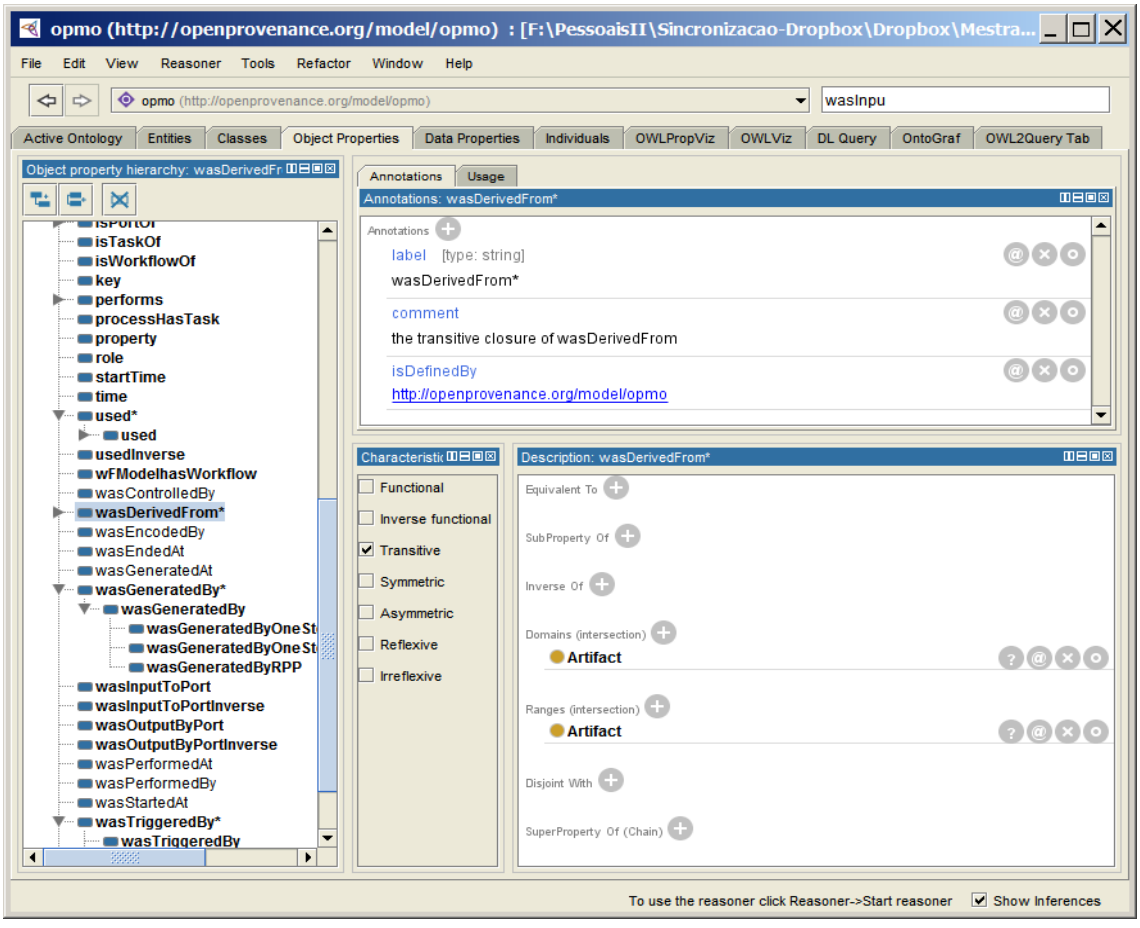


Figura 3.12. Propriedade *wasDerivedFrom implementada na ontologia OPMO original, ilustrada na ferramenta Protégé 4.2.**

A utilização de ontologias permite que o SciProvMiner realize inferências válidas no modelo OPM utilizando tecnologia da Web semântica específica para este fim, sem a necessidade de serem especificadas consultas em SQL como é feito em Lim et al. (2011). O grande problema em se utilizar SQL para a realização destas consultas é que as mesmas podem ser custosas para o SGBD, devido ao fato de modelos relacionais normalizados serem destinados ao armazenamento de dados e não a processamentos analíticos e também por essas consultas serem de natureza complexa e poderem envolver grande volume de dados.

Assim, considerando as inferências em múltiplos passos definidas na documentação do modelo OPM, podemos citar que a primeira delas denominada "*WasDerivedFrom*",* cuja definição descrita em Moreau et al. (2011) diz que um artefato a1 foi derivado de a2 (a1 *wasDerivedFrom* a2), possivelmente em múltiplos passos, escrito como a1 →* a2, se a1 foi derivado do artefato a2 ou de um artefato que foi derivado de a2 (possivelmente em múltiplos passos) já estava declarada na ontologia OPMO original, através da propriedade *wasDerivedFrom⋆*, que tem por domínio e *range* a classe *Artifact* e tem a característica de ser Transitiva, no entanto não havia sido definida a regra para ela em termos de cadeia de propriedades, e, portanto, a ontologia não estava preparada para realizar nenhuma das inferências em múltiplos passos definidas na documentação do modelo OPM (Moreau et al. 2011), visto que todas as inferências em múltiplos passos dependem da propriedade *WasDerivedFrom**. Foi então inserida a cadeia de propriedade *"wasDerivedFrom o wasDerivedFrom*" na propriedade *WasDerivedFrom.*

A inferência em múltiplos passos "p *used artirfact* a" p →* a, que afirma que um processo p usou o artefato a (possivelmente usando múltiplos passos), se p usou um artefato que era "a" ou que foi derivado (*wasDerivedFrom*) do artefato a (possivelmente usando múltiplos passos) também já está implementada na ontologia OPMO original, através da propriedade *used**, que tem por domínio a classe *Process*, por *range* a classe *Artifact* e é definida através da *property chain* "*used ∘ wasDerivedFrom⋆ SubPropertyOf used⋆*".

A inferência em múltiplos passos "a *wasGeneratedBy process* p", a →* p, que diz que o artefato a foi gerado pelo processo p (a *wasGeneratedBy* p), possivelmente utilizando múltiplos passos, se a foi um artefato gerado por p ou foi derivado de um artefato (*wasDerivedFrom*) que foi gerado por p, também já está implementada na ontologia OPMO, através da propriedade *wasGeneratedBy⋆* tendo por domínio a classe *Artifact*, por *Range* a

classe *Processe* sendo definida através da cadeia de propriedades "*wasDerivedFrom** ∘ *wasGeneratedBySubPropertyOfwasGeneratedBy**".

Já a inferência em múltiplos passos "p1 *wasTriggeredBy p2",* p1 →* p2, diz que o processo p1 foi disparado pelo processo p2 (*wasTriggeredBy*), se p1 usou um artefato que foi gerado (*wasGeneratedBy*) por p2 (possivelmente usando múltiplos passos), ou p1 foi derivado de um artefato (possivelmente utilizando múltiplos passos) que foi gerado por p2. Esta inferência em múltiplos passos não está implementada na ontologia OPMO atualmente.

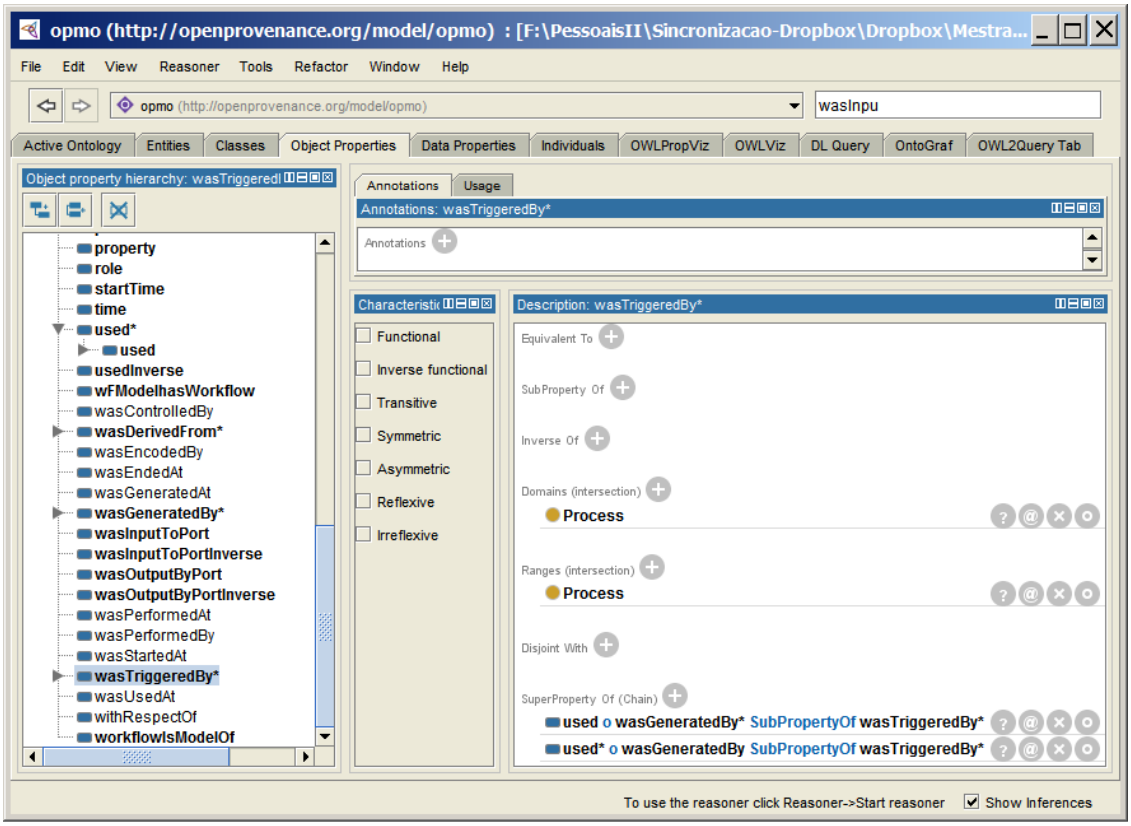


Figura 3.13. Propriedade *wastriggeredby implementada na ontologia OPMO-e, ilustrada na ferramenta Protégé 4.2.**

Com o objetivo de defini-la, inserimos uma *object property* chamada *wasTriggeredBy** tendo a classe *Process* como domínio e *range*. Para definir a propriedade, utilizamos duas *property chains*, sendo a primeira definida como "*used** ∘ *wasGeneratedBy subproperty of wasTriggeredBy*"* para contemplar a situação de p_1 ter usado um artefato que foi gerado por p_2, e a segunda definida como "*used** ∘ *wasGeneratedBy subproperty of wasTriggeredBy*"* para cobrir a situação p_1 ter sido derivado de um artefato (possivelmente utilizando múltiplos passos) que foi gerado por p_2. A Figura 3.13 ilustra a implementação desta propriedade na ontologia OPMO-e na ferramenta Protégé.

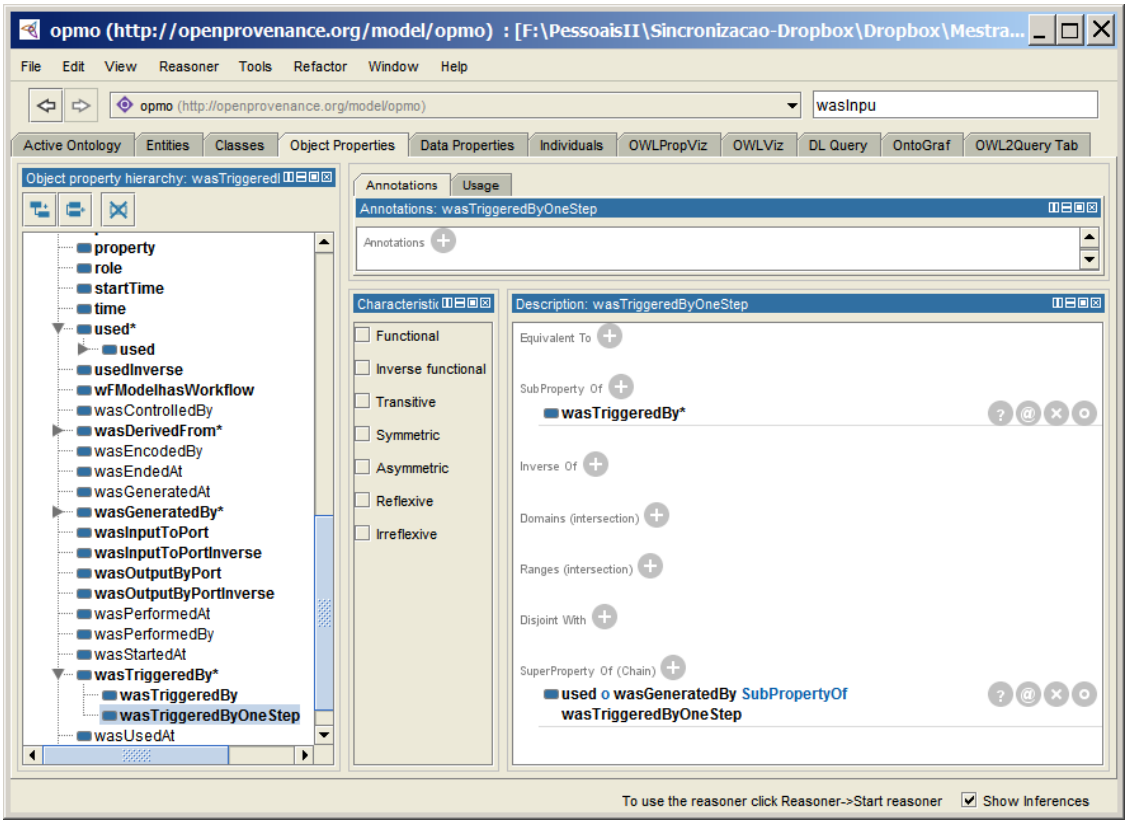


Figura 3.14. Propriedade *wasTriggeredByOneStep* implementada na ontologia OPMO-e, ilustrada na ferramenta Protégé 4.2.

Com relação às regras de completude definidas na documentação do modelo OPM (MOREAU et al. 2011), a regra que diz respeito à eliminação do artefato A, mostrada na Figura 2.2, onde é afirmado que uma aresta *wasTriggeredBy* pode ser obtida a partir da existência das arestas *used* e *wasGeneratedBy*, não está definida na ontologia OPMO. Como o objetivo de defini-la, inserimos uma *object property* denominada *wasTriggeredByOneStep* como sub-propriedade de *wasTriggeredBy**, definida na ontologia OPM original, com a classe *Processo* como Domínio e *Range*. Esta propriedade foi definida a partir da cadeia de propriedades "*used* ○ *wasGeneratedBy subPropertyOf wasTriggeredByOneStep*". A Figura 3.14 ilustra a implementação desta propriedade na ontologia OPMO-e na ferramenta Protégé.

Apesar de a regra inversa da regra de completude denominada Introdução de Processo ilustrada na Figura 2.3 não ser definida explicitamente na documentação do modelo OPM, conforme explicitado na seção 2.1.2.1, a documentação do modelo OPM considera a regra inversa válida, ou seja, a eliminação do processo, caso exista alguma anotação relacionada àquele workflow afirmando que todas as saídas dos processos são dependentes de todas as

suas entradas (MOREAU et al. 2011). No SciProvMiner o usuário configura esta informação através do parâmetro *AllOutputsDependentAllInputs* da instância de serviço Web para instrumentalização do workflow configurado com o método *initialConfiguration*, configurando o valor como "true" caso essa informação seja verdadeira e "false" caso contrário.

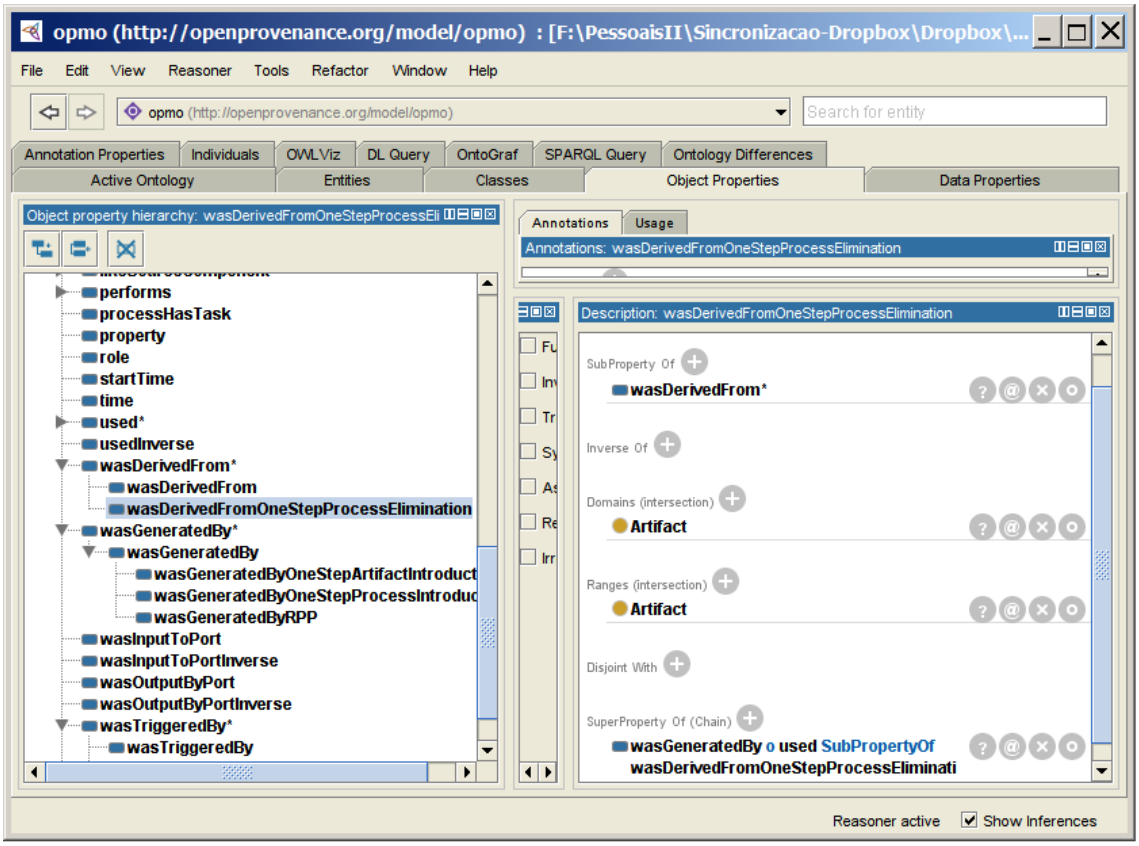


Figura 3.15. Propriedade *wasDerivedFromOneStepProcessElimination* implementada na ontologia OPMO-e, ilustrada na ferramenta Protégé 4.2.

Caso seja configurado como verdadeira esta afirmação de que todas as saídas de um processo são dependentes de suas entradas para aquele workflow, o SciProvMiner utiliza a ontologia OPMO-e onde foi definida a propriedade *wasDerivedFromOneStep-ProcessElimination*, subpropriedade de *wasDerivedFrom**, sob a cadeia de propriedades "*wasGeneratedBy* o *used*", tendo a classe *Artifact* como domínio e *range*, e adiciona a cadeia de propriedades "*wasDerivedFromOneStepProcessIntroduction o wasDerivedFromOneStep-ProcessElimination*" à propriedade *wasDerivedFrom**, para que a nova propriedade seja considerada na inferência em múltiplos passos realizada pela propriedade *wasDerivedFrom**. Caso esta afirmação seja falsa, é utilizada uma versão da ontologia OPMO-e onde não está

definida a propriedade *wasDerivedFromOneStepProcessElimination*. A Figura 3.15 ilustra a implementação desta propriedade na ontologia OPMO-e na ferramenta Protégé.

Para capturar as demais regras de completude definidas na documentação do modelo OPM, foi necessário encontrar meios para inferir conhecimento que não está expresso de forma explícita na base de conhecimento do SciProvMiner, pois estas regras são consideradas transformações com perda de informação, que necessitam que peças do modelo desconhecidas sejam encontradas (artefato ou processo, dependendo da regra) para que elas possam ser implementadas.

Foram estudadas técnicas de mineração de dados tais como regras de associação, com o objetivo de garantir que o SciProvMiner cobrisse todas as regras de completude do modelo OPM. Apesar de esses mecanismos serem utilizados para inferir conhecimento não explicito na base de dados, essas inferências possuem certo grau de incerteza, devido à natureza probabilística dessas técnicas. Buscando então outros meios, foi descoberto que através das informações de proveniência prospectiva relacionadas às retrospectivas disponíveis na base de conhecimento do SciProvMiner, modeladas através de propriedades de cadeias na ontologia OPMO-e, se tornaria possível a recuperação dessas informações desconhecidas do modelo.

A regra de completude de introdução de artefato, ilustrada na Figura 2.2, afirma que a introdução de um artefato permite estabelecer que uma aresta *wasTriggeredBy* está escondendo a existência de algum artefato A usado por P2 (A *used* P2) e gerada por P1 (A *wasGeneratedBy* P1). O algoritmo desenvolvido para encontrar este artefato A segue os seguintes passos:

- É realizada a verificação de qual porta de saída (porta output na Figura 3.16) da tarefa T1, que se relaciona com o processo P1, está conectada a porta de entrada (porta input na Figura 3.16) da tarefa T2 relacionada ao processo P2, sendo que o processo P1 e P2 se relacionam na dependência causal *wasTriggeredBy* (P2 *wasTriggeredBy* P1).
- Tendo encontrado a porta output, é realizada a procura do artefato A que possui esta porta como *wasOutputByPort*. Este é o artefato procurado.

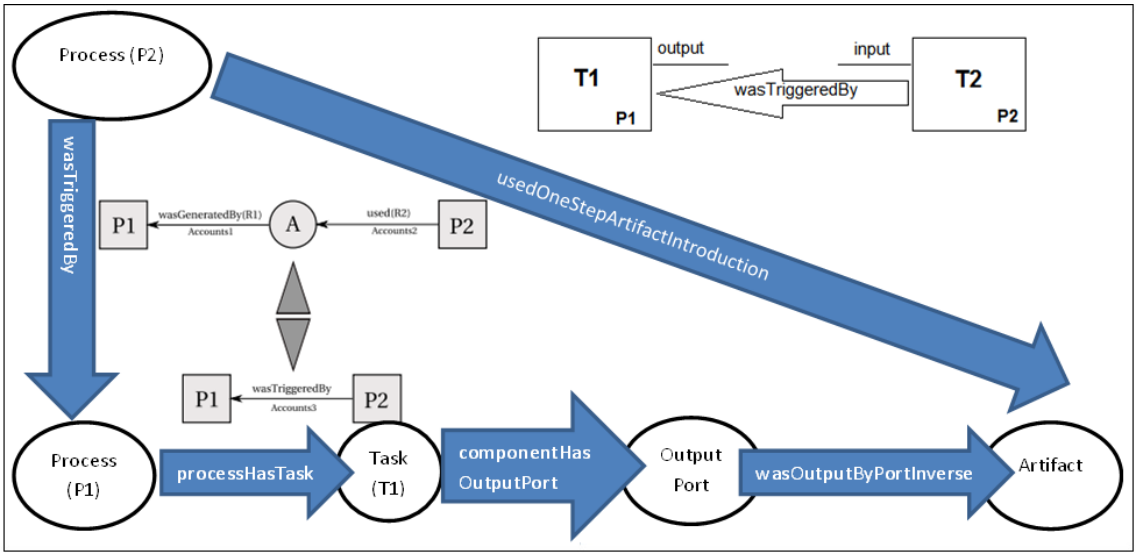


Figura 3.16. Ilustração da cadeia de propriedades para formação da propriedade *usedOneStepArtifactIntroduction*.

Duas propriedades foram criadas para esta regra de completude, uma para definir a dependência causal *used* entre o processo P2 e o artefato A (P2 *used* A), denominada *usedOneStepArtifactIntroduction*, subpropriedade de *used*, e outra para definir a dependência causal *wasGeneratedBy* entre o artefato A e o processo P1 (A *wasGeneratedBy* P1), denominada *wasGeneratedByOneStepArtifactIntroduction*, subpropriedade de *wasGeneratedBy*. Ambas foram criadas em termos de cadeia de propriedades. A Figura 3.16 ilustra a *property chain* construída para a criação da propriedade *usedOneStepArtifactIntroduction*.

A propriedade *usedOneStepArtifactIntroduction* possui como domínio a classe *Process* e como *range* a classe *Artifact*, assim como a propriedade *used* da qual ela deriva. Esta propriedade é definida pela cadeia de propriedades "*wasTriggeredBy o processHasTask o componentHasOutputPort o wasOutputByPortInverse*", onde a propriedade *wasTriggeredBy* é também definida por cadeia de propriedades "*effectWasTriggeredByInverse o causeWasTriggeredBy*" na ontologia OPMO original, e as demais propriedades que fazem parte da cadeia são propriedades simples introduzidas na ontologia OPMO-e apresentadas na seção anterior. A Figura 3.17 ilustra a implementação desta propriedade na ontologia OPMO-e na ferramenta Protégé.

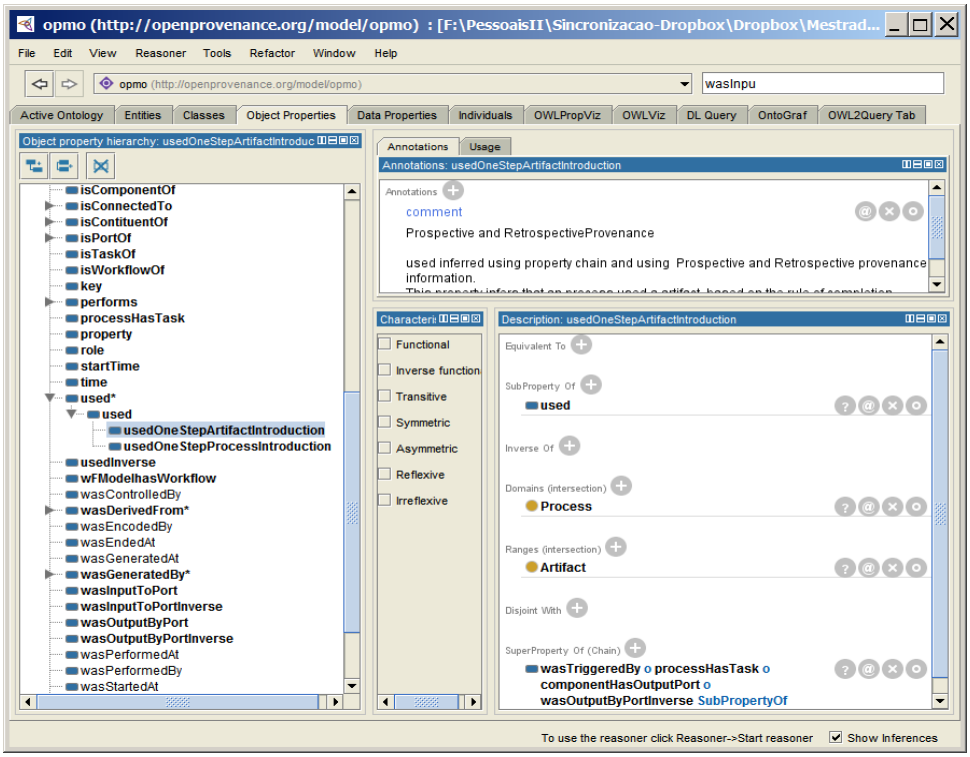


Figura 3.17. Propriedade *usedOneStepArtifactIntroduction* implementada na ontologia OPMO-e, ilustrada na ferramenta Protégé 4.2.

A cadeia de propriedades utilizada para a criação da propriedade *wasGeneratedByOneStepProcessIntroduction* é apresentada na Figura 3.18.

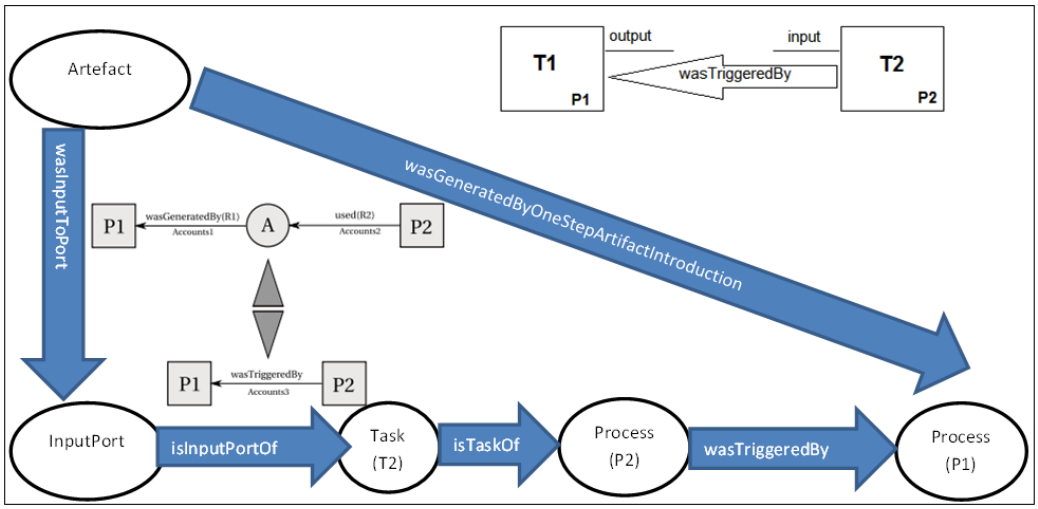


Figura 3.18. Ilustração da *cadeia de propriedades para formação* da propriedade *wasGeneratedByOneStepArtifactIntroduction*.

Esta propriedade possui como domínio a classe *Artifact* e como *range* a classe *Process*, assim como a propriedade *wasGeneratedBy* da qual ela deriva. Esta propriedade é definida

pela cadeia de propriedades "*wasInputToPort o isInputPortOf o isTaskOf o wasTriggeredBy*", onde as propriedades *wasInputToPort*, *isInputPortOf*, *isTaskOf* são propriedades simples definidas na seção anterior e a propriedade *wasTriggeredBy* é uma definida por cadeia de propriedades na ontologia OPMO original, como mostrado anteriormente. A Figura 3.19 ilustra a implementação desta propriedade na ontologia OPMO-e na ferramenta Protégé.

Para ab*ranger* a regra de completude unidirecional denomina Introdução de Processo, ilustrada na Figura 2.3, que afirma que uma aresta *wasDerivedFrom* esconde a presença de um processo intermediário P que usa o artefato A1 e gera o artefato A2, o seguinte algoritmo foi criado: dado que se conhece quem é o artefato A2, e que este artefato tem a propriedade *wasOutputByPort* que relaciona o artefato A2 a porta output da tarefa T pela qual A2 é disponibilizado para ser consumido, busca-se o processo P associado à tarefa T que possui a porta output pelo qual o artefato foi gerado. P é o processo procurado. Tendo descoberto o processo P, pode-se inferir que A2 foi gerado por P (A2 *wasGeneratedBy* P).

Neste cenário, pode ser afirmado que, se o artefato A2 foi derivado do artefato A1 (A2 *wasDerivedFrom* A1), e se o artefato A2 foi gerado por P (A2 *wasGeneratedBy* P), pode ser inferido que o processo P usou o artefato A1 (P *used* A1), segundo a regra de completude de introdução de Processo.

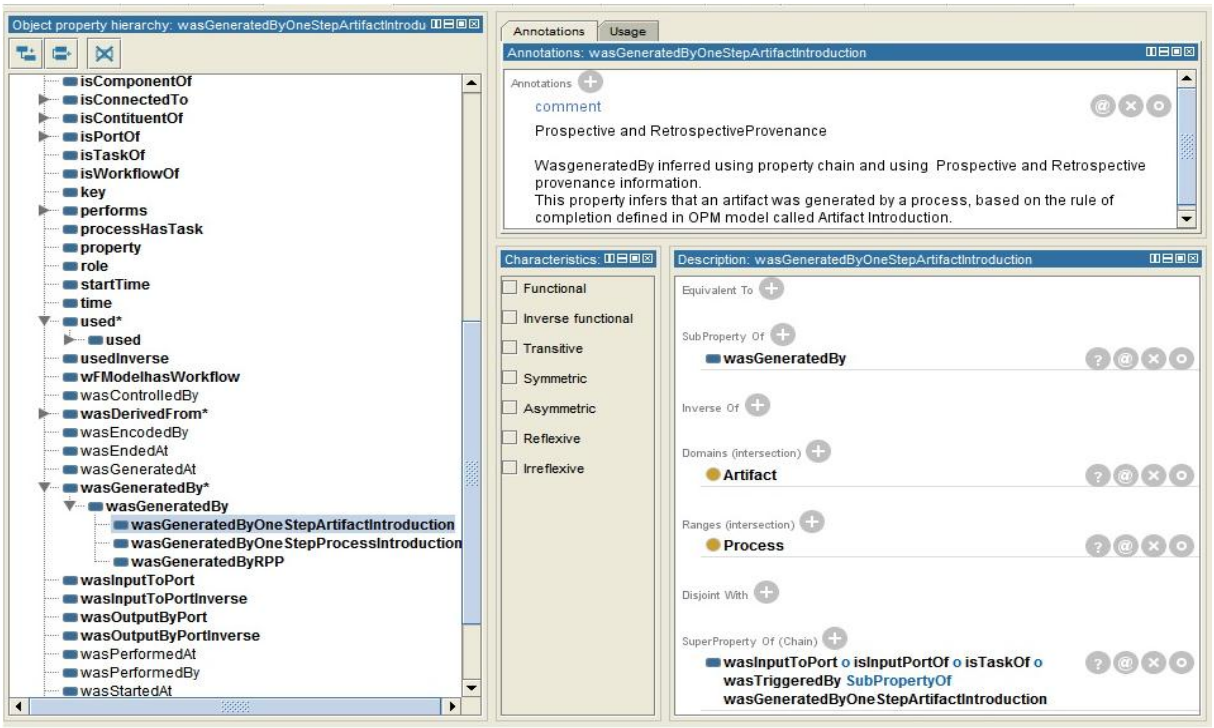


Figura 3.19. Propriedade *wasGeneratedByOneStepArtifactIntroduction* implementada na ontologia OPMO-e, ilustrada na ferramenta Protégé 4.2.

Outras duas propriedades foram criadas para esta regra de completude, uma para definir a dependência causal *used* entre o processo P e o artefato A1 (P *used* A1), denominada *usedOneStepProcessIntroduction*, subpropriedade de *used*, e outra para definir a dependência causal *wasGeneratedBy* entre o artefato A2 e o processo P (A2 *wasGeneratedBy* P), denominada *wasGeneratedByOneStepProcessIntroduction*, subpropriedade de *wasGeneratedBy*. Ambas foram criadas em termos de cadeia de propriedades. A Figura 3.20 ilustra a *property chain* construída para a criação da propriedade *usedOneStepProcessIntroduction*.

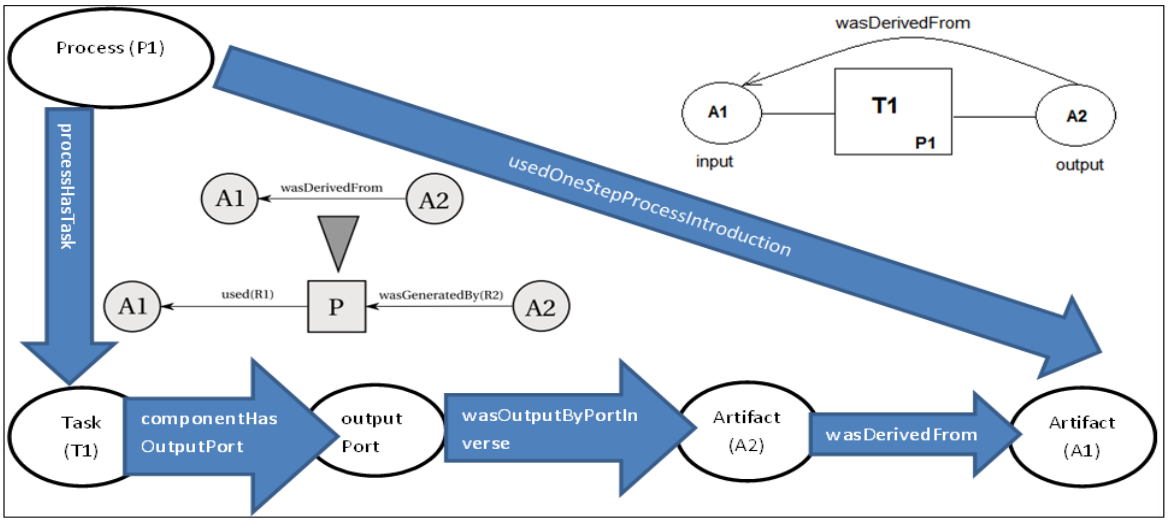


Figura 3.20. Ilustração da cadeia de propriedades para formação da propriedade *usedOneStepProcessIntroduction*.

A propriedade *usedOneStepProcessIntroduction* possui como domínio a classe *Process* e como *range* a classe *Artifact*, assim como a propriedade *used* da qual ela é subpropriedade. Esta propriedade é definida pela cadeia de propriedades "*processHasTask o componentHasOutputPort o wasOutputByPortInverse o wasDerivedFrom*", onde a propriedade *wasDerivedFrom* é também definida por cadeia de propriedades "*effectWasDerivedFromInverse o causeWasDerivedFrom*" na ontologia OPMO original, e as demais propriedades que fazem parte da cadeia são propriedades simples introduzidas na ontologia OPMO-e que já foram apresentadas na seção anterior. A Figura 3.21 ilustra a implementação desta propriedade na ontologia OPMO-e, na ferramenta Protégé.

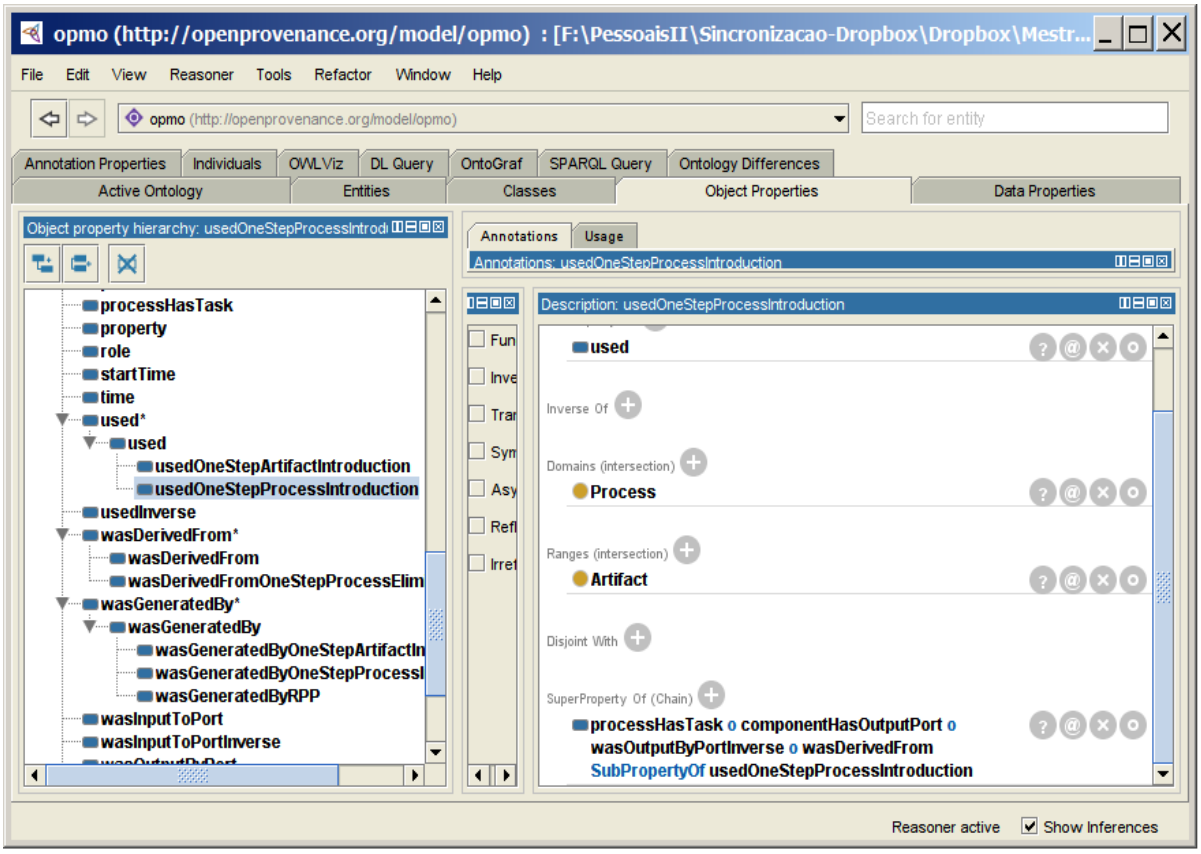


Figura 3.21. Propriedade *usedOneStepProcessIntroduction* implementada na ontologia OPMO-e, ilustrada na ferramenta Protégé 4.2.

A cadeia de propriedades utilizada para a criação da propriedade *wasGeneratedByOneStepProcessIntroduction* é apresentada na Figura 3.22.

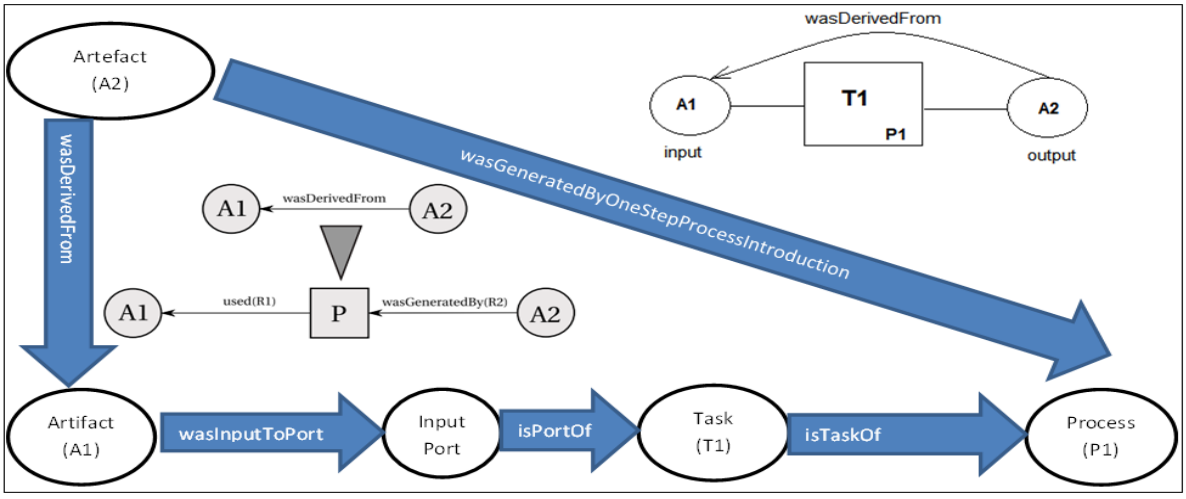


Figura 3.22. Ilustração da cadeia de propriedades para formação da propriedade *wasGeneratedByOneStepProcessIntroduction*.

A propriedade *wasGeneratedByOneStepProcessIntroduction* possui como domínio a classe *Artifact* e como *range* a classe *Process*, assim como a propriedade *wasGeneratedBy* da qual ela é subpropriedade. Esta propriedade é definida pela cadeia de propriedades "*wasDerivedFrom o wasInputToPort o isPortOf o isTaskOf*", onde a propriedade *wasDerivedFrom* é também definida por cadeia de propriedades na ontologia OPMO original, como mencionado anteriormente, e as demais propriedades foram introduzidas na ontologia OPMO-e apresentadas na seção anterior. A Figura 3.23 ilustra a implementação desta propriedade na ontologia OPMO-e, na ferramenta Protégé.

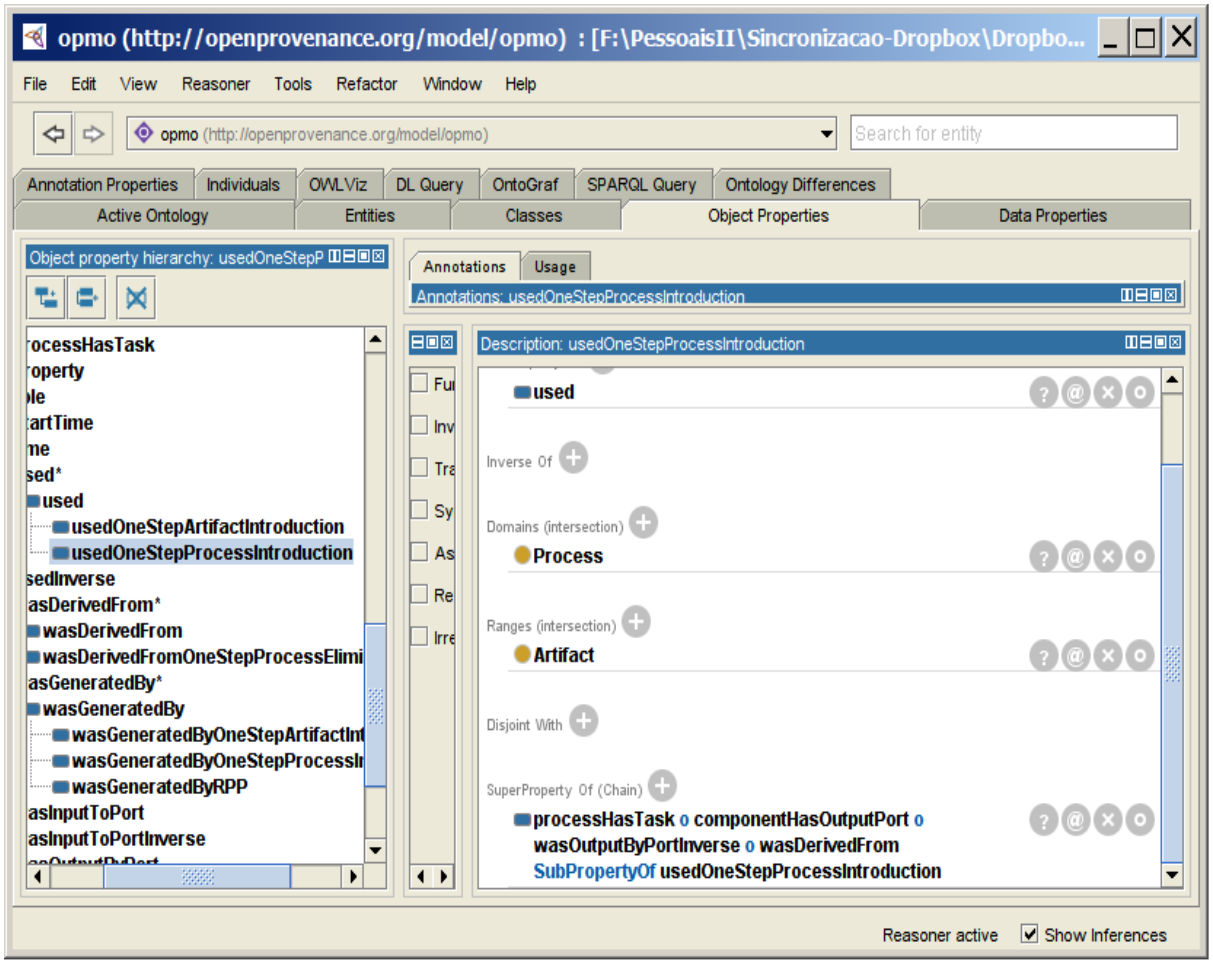


Figura 3.23. Propriedade *usedOneStepProcessIntroduction* implementada na ontologia OPMO-e, ilustrada na ferramenta Protégé 4.2.

Na ontologia OPMO original foram implementadas as propriedades *used*, *wasDerivedFrom*, *wasTriggeredBy* e *wasGeneratedBy* em termos de cadeia de propriedades, como pode ser visto na Tabela 3.1. Porém, a propriedade *wasControlledBy* não foi implementada em termos de cadeia de propriedades para ser inferido o relacionamento direto entre o Agente e o Processo que fazem parte do relacionamento. Então foi realizada a

implementação desta propriedade em termos de cadeia de propriedades na ontologia OPMO-e, para que este conhecimento também possa ser inferido e não precise ser explicitamente declarado. A cadeia de propriedade usada foi "*effectWasControlledByInverse o causeWasControlledBy*" conforme ilustra a Figura 3.24.

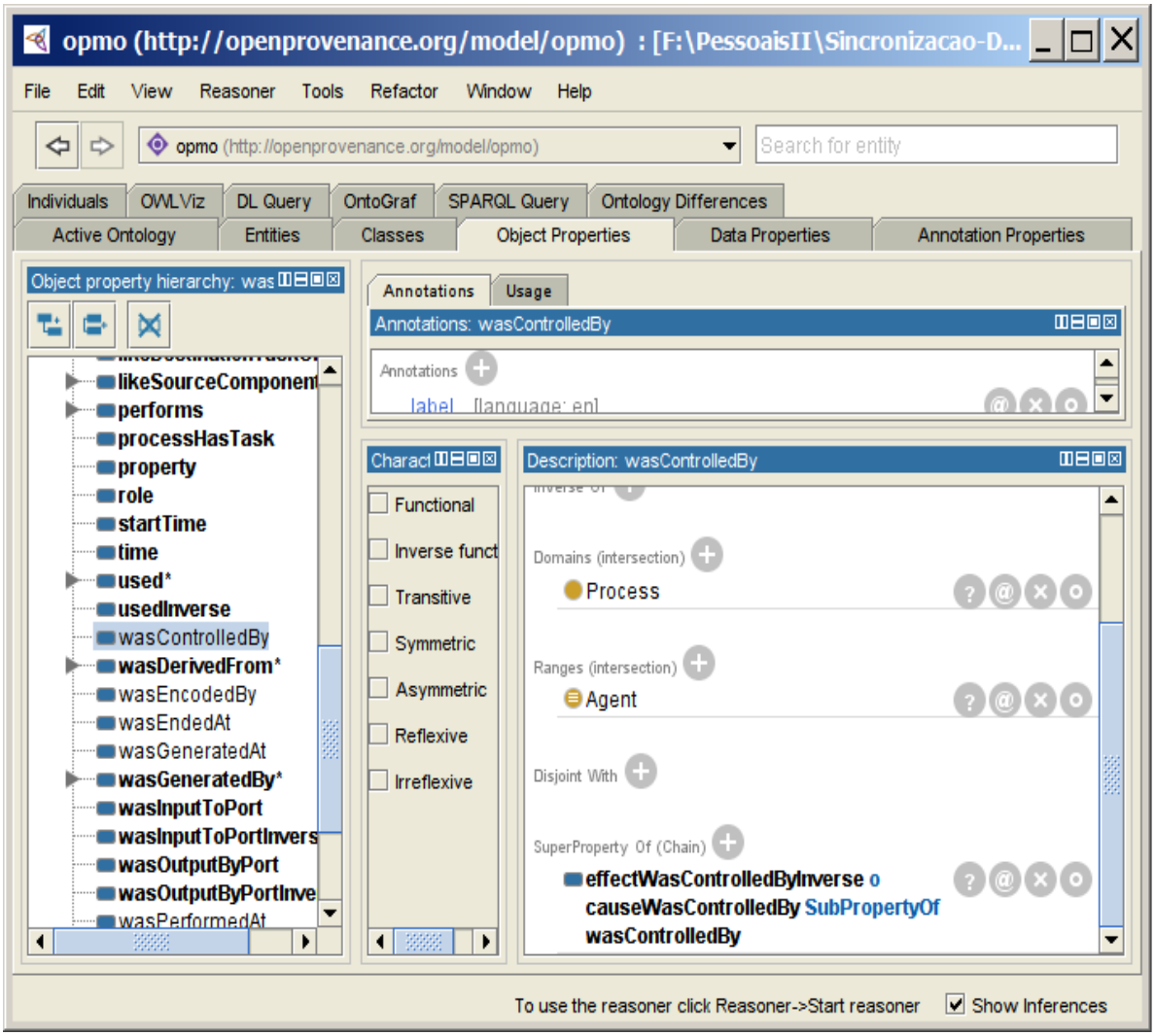


Figura 3.24. Propriedade *wasGeneratedBy* implementada através de *property chain* na ontologia OPMO-e, ilustrada na ferramenta Protégé 4.2.

3.2.2.3.Implementação de otimização do mecanismo de instrumentalização

Com o objetivo de diminuir o trabalho do cientista no momento da instrumentalização do workflow, que é um trabalho manual e dispendioso, foi estudado e identificado que é possível sublimar a instrumentalização da dependência causal *wasGeneratedBy* de tal forma que esta dependência causal só será utilizada no workflow quando se desejar capturar parte da

proveniência prospectiva do workflow utilizando uma instância de serviço Web configurada com o método *wasGeneratedBy*.

A característica de se capturar a proveniência prospectiva e de ser modelada na ontologia OPMO-e foi fundamental para tornar possível a sublimação desta dependência causal, visto que para a sua inferência foi adicionada uma propriedade na ontologia OPMO denominada *wasGeneratedbyRPP*, subpropriedade de *wasGeneratedBy*, que considera em sua formação informações de proveniência prospectiva relacionada a informações de proveniência retrospectiva. A Figura 3.25 ilustra a cadeia de propriedades sob a qual a propriedade *wasGeneratedbyRPP* foi formada. A idéia por trás desta regra é que se um artefato A foi disponibilizado para ser utilizado por outros processos através de uma porta de saída P de uma tarefa T, e sendo o processo P a representação da execução da tarefa T, então pode ser inferido que o artefato A foi gerado pelo processo P.

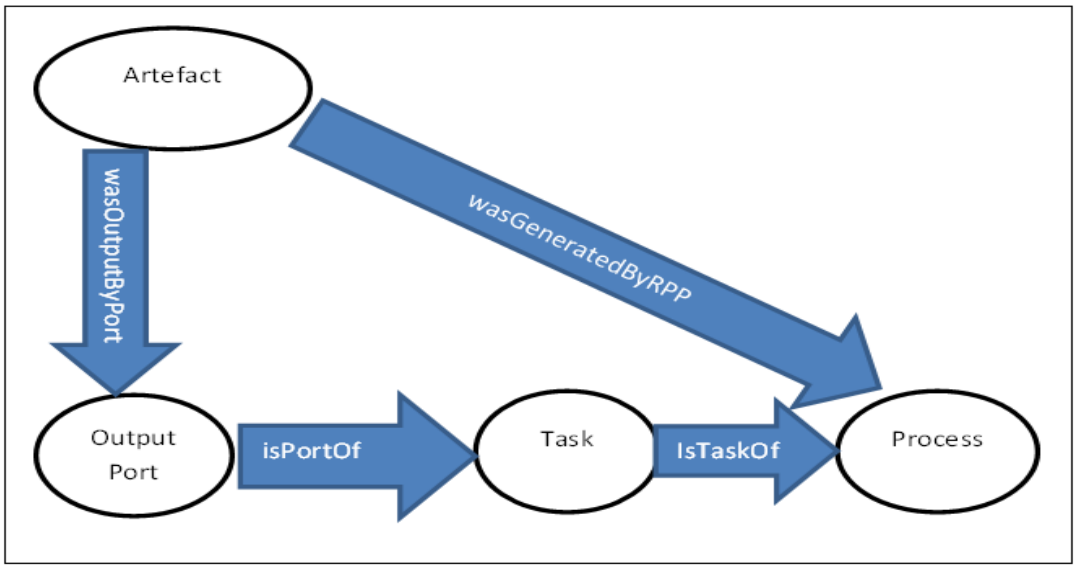


Figura 3.25. Ilustração da cadeia de propriedades para formação da propriedade *wasGeneratedByRPP*.

A propriedade *wasGeneratedByRPP* é responsável por inferir o conhecimento a respeito da dependência causal *wasGeneratedBy* entre o artefato e o processo envolvidos na dependência não declarada explicitamente pela otimização na instrumentalização do workflow realizada. Foi dado o nome de *wasGeneratedByRPP* a esta propriedade, onde RPP é abreviatura de *Retrospective and Prospective Provenance*, pelo fato dela ser formada por propriedades que usam informações de proveniência prospectiva com informações de proveniência retrospectiva na sua formação, conforme ilustrado na Figura 3.25. A Figura 3.26 exibe a implementação desta propriedade na ontologia OPMO-e.

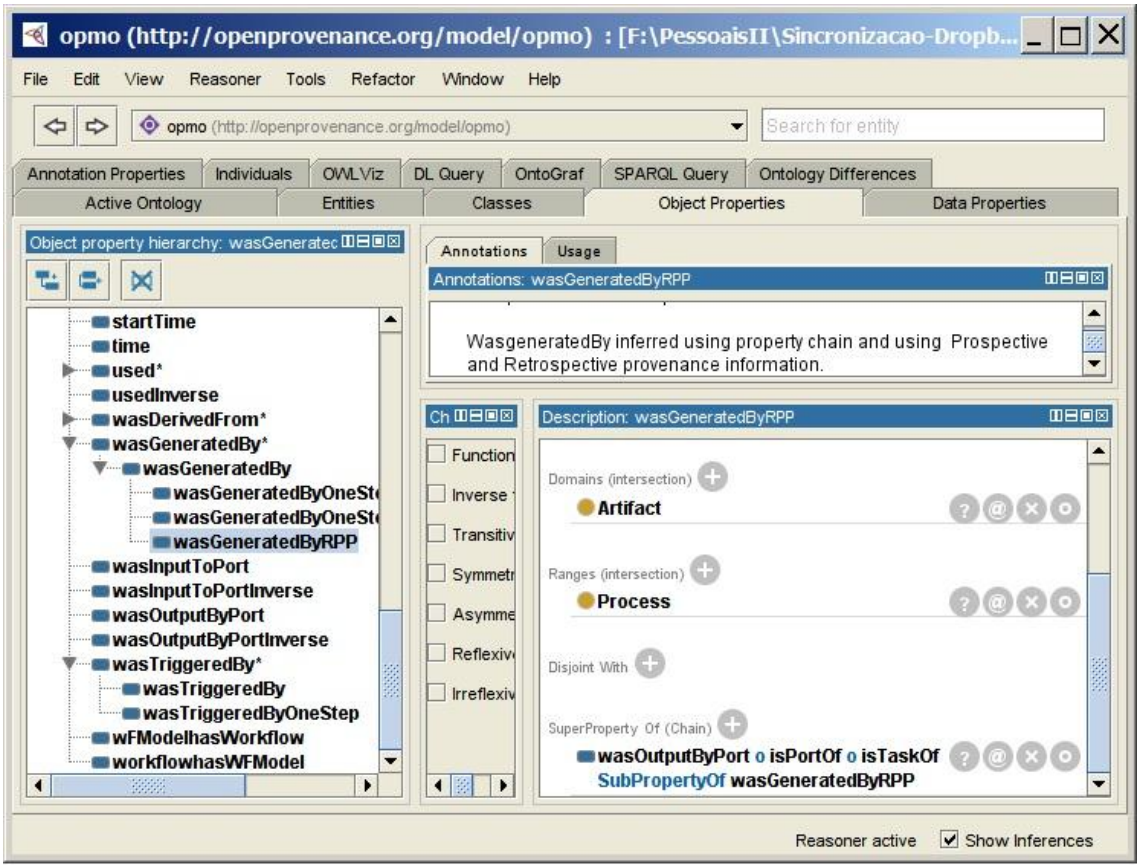


Figura 3.26. Propriedade *wasGeneratedByRPP* implementada através de *property chain* na ontologia OPMO-e.

3.2.2.4. Enriquecimento da ontologia OPMO por adição de propriedades com poder de inferência

Para enriquecimento da ontologia, foram construídas outras propriedades definidas a partir de cadeia de propriedades. Uma delas é a propriedade *likeSourceComponent*, cuja função é dizer que o indivíduo Component que está no domínio da propriedade se conecta com o indivíduo Component que está no *range* da propriedade com a função de ser o componente antecessor na conexão entre esses componentes. Esta propriedade foi definida por meio da cadeia de propriedades, "*componentHasOutputPort o likeSourcePort o isInputPortOf*", possuindo *Component* como domínio e *range*. A Figura 3.27 ilustra a cadeia de propriedades que forma a propriedade *likeSourceComponent* e a Figura 3.28 mostra a implementação desta propriedade na ontologia OPMO-e, na ferramenta Protégé.

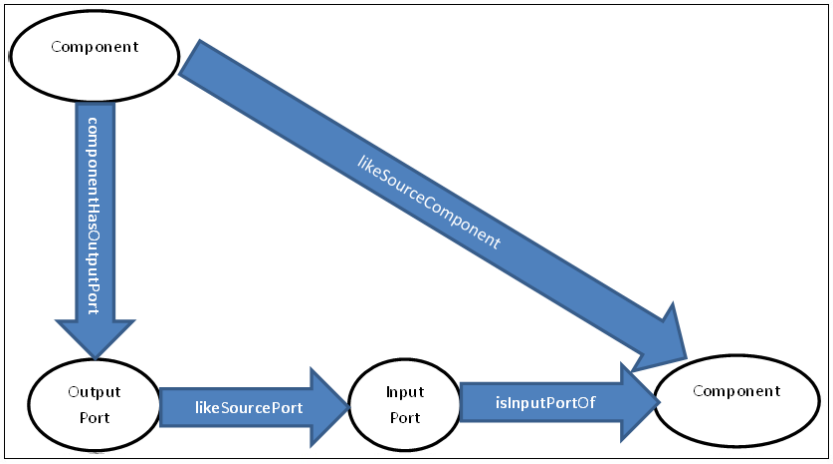


Figura 3.27. Ilustração da cadeia de propriedades para formação da propriedade *likeSourceComponent*.

Com o objetivo obter a cadeia de componentes antecessores de um componente foi implementada a propriedade *likeSourceComponent** da qual a propriedade *likeSourceComponent* é subpropriedade como pode ser visto na Figura 3.28. Esta propriedade foi formada tendo a classe *Component* como domínio e *range* da propriedade, tendo como *property chain* a expressão "*likeSourceComponent o likeSourceComponent*" e tendo a característica de ser transitiva, o que lhe confere o poder de formar a cadeia de componentes antecessores de um determinado componente.

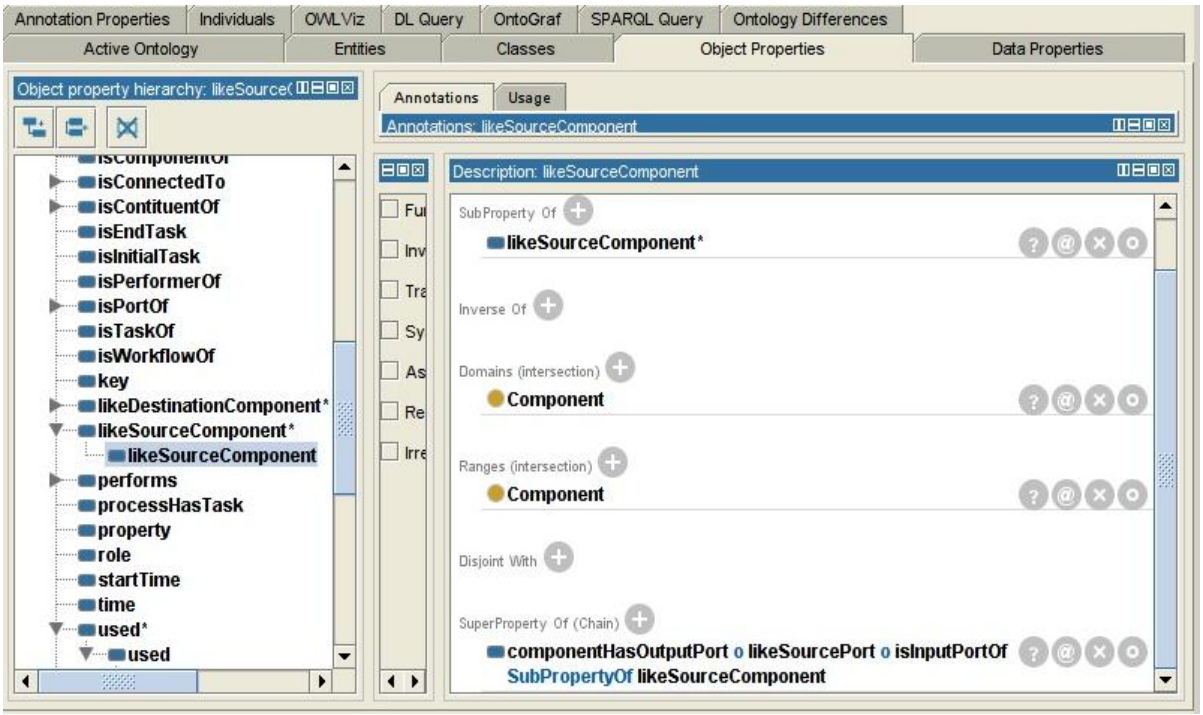


Figura 3.28. Propriedade *likeSourceComponent* implementada no Protégé 4.2

Como uma propriedade baseada em cadeia de propriedades não pode ter a característica de ser inversa foi criada também a propriedade *likeDestinationComponent*, já mencionado anteriormente, tendo como domínio e *range* a classe *Component*, e foi definida a partir da cadeia de propriedades "*componentHasInputPort o likeDestinationPort o isPortOf*". A Figura 3.29 ilustra a formação da cadeia de propriedades de *likeDestinationComponent*.

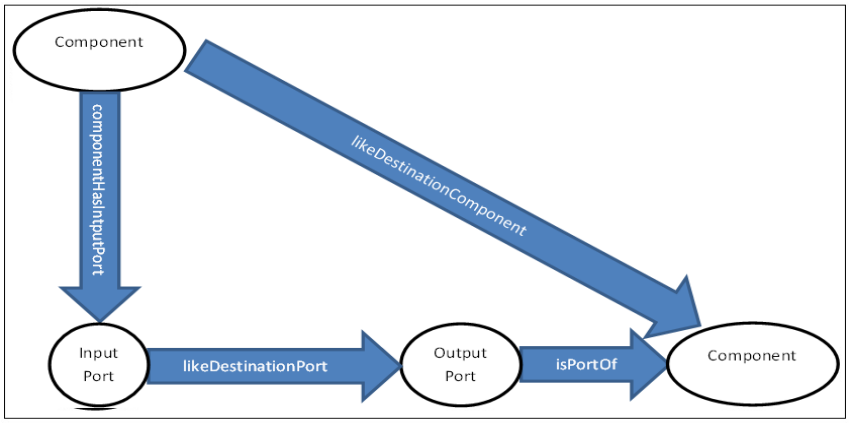


Figura 3.29. Ilustração da cadeia de propriedades para formação da propriedade *likeDestinationComponent*.

A Figura 3.30 apresenta a implementação desta propriedade na ontologia OPMO-e, na ferramenta Protégé.

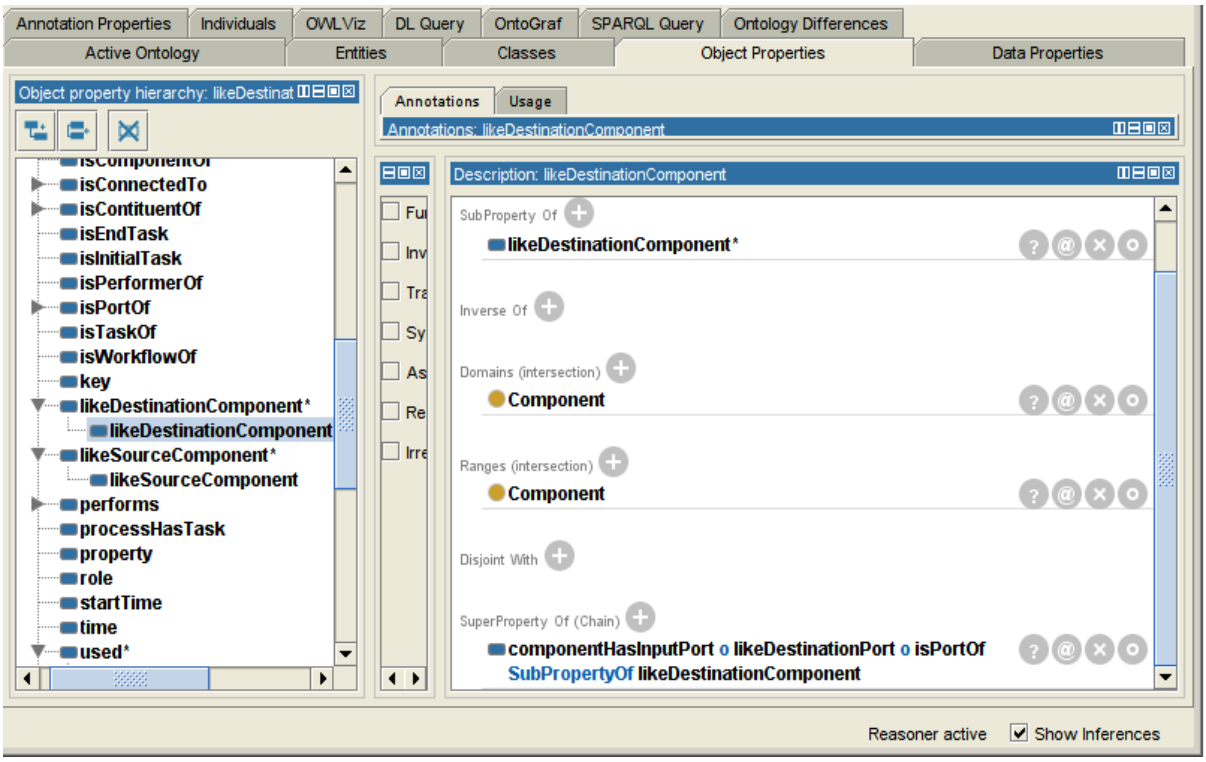


Figura 3.30. Propriedade *likeDestinationComponent* implementada no Protégé 4.2

Com o objetivo de obter a cadeia de componentes sucessores de um componente foi implementada a propriedade *likeDestinationComponent** da qual a propriedade *likeDestinationComponent* é subpropriedade como pode ser visto na Figura 3.30. Esta propriedade foi formada tendo a classe Component como domínio e *range* da propriedade, tendo como *property chain* a expressão "*likeDestinationComponent o likeDestinationComponent*" e tendo a característica de ser transitiva, o que lhe confere o poder de formar a cadeia de componentes sucessores de um determinado componente.

Também foi criada a propriedade *connectedComponents* que possui como domínio e *range* a classe *Component*, e é definida a partir da cadeia de propriedades "*hasPort o isConnectedTo o isPortOf*", como ilustrado na Figura 3.31. Esta propriedade infere qual componente está conectado a qual outro componente.

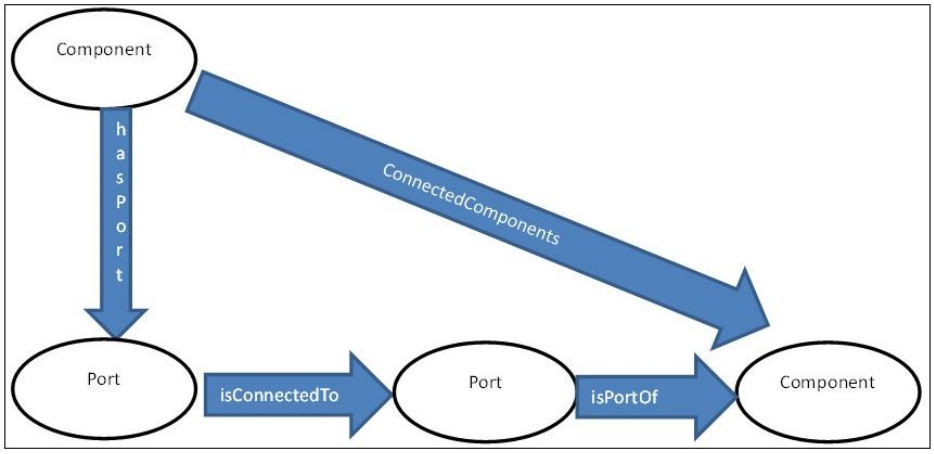


Figura 3.31. Ilustração da cadeia de propriedades para formação da propriedade *connectedComponents*.

A Figura 3.32 apresenta a implementação desta propriedade na ontologia OPMO-e, na ferramenta Protégé.

Segundo Yu (2011), um dos principais benefícios de se construir ontologia é que ela pode ajudar a encontrar fatos implícitos nas informações inseridas na base de conhecimento, principalmente aqueles que não estão tão aparentes para o usuário. Através da exploração do conceito de cadeia de propriedades inserida na OWL2, o SciProvMiner tentou aprimorar a ontologia OPMO, criando para o usuário novas formas de explorar o conhecimento explícito e implícito na base de conhecimento. Essa extensão permite que seja possível inferir conhecimento novo e útil para o usuário, como está previsto na documentação do modelo OPM, onde é dito ser esperado que algoritmos inteligentes possam explorar o modelo de

dados do OPM para fornecer ao usuário novas e poderosas funcionalidades (MOREAU et al. 2011).

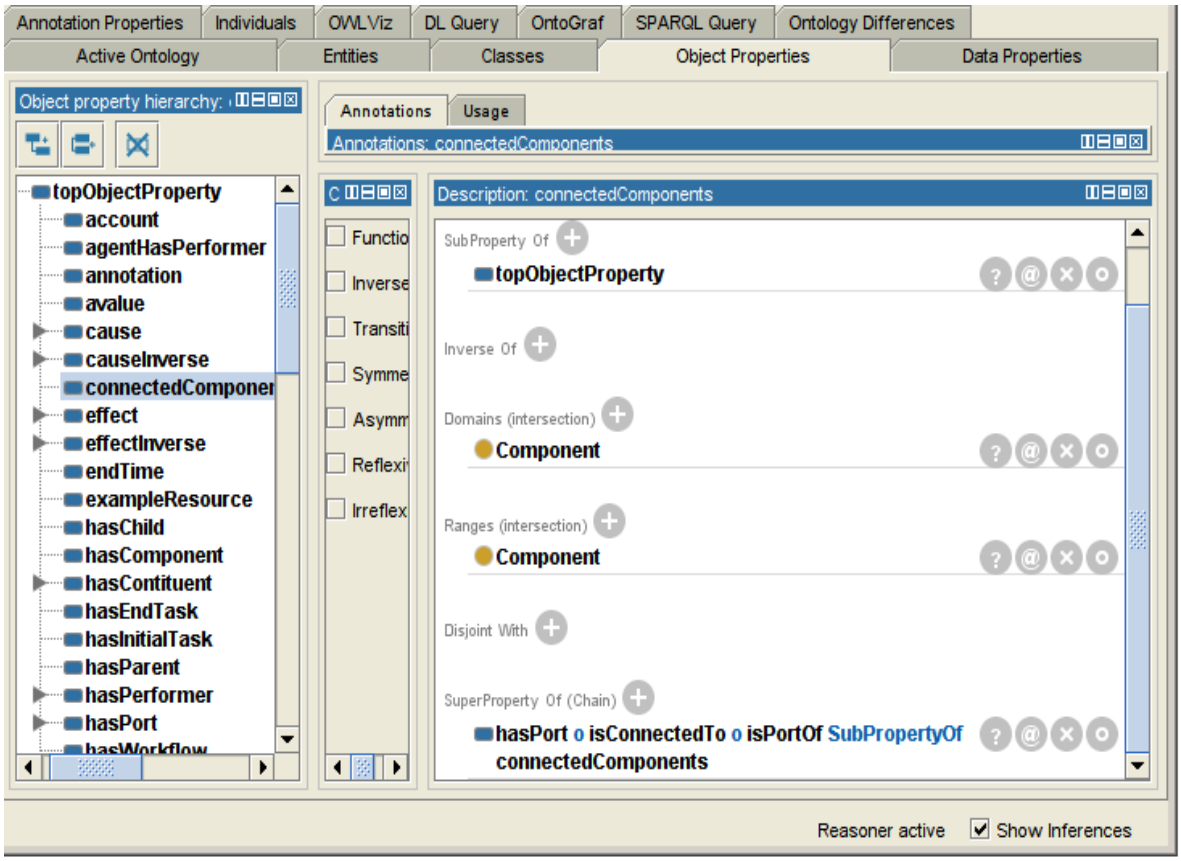


Figura 3.32. Propriedade *connectedComponents* implementada no Protégé 4.2

3.3. CONSIDERAÇÕES FINAIS

Este capítulo apresentou a arquitetura SciProvMiner, onde foi detalhada cada uma de suas camadas, explicitando as contribuições do SciProvMiner sobre a arquitetura SciProv. A implementação do SciProvMiner também foi detalhada neste capítulo, mostrando como foram utilizadas tecnologias de serviços Web para o desenvolvimento da ferramenta de captura de proveniência prospectiva e retrospectiva de dados, de forma a garantir a independência da ferramenta de coleta do SciProvMiner em relação ao SGWfC utilizado pelo cientista e como a ontologia OPMO foi estendida de forma a enriquecer semanticamente o conhecimento do cientista a respeito do experimento realizado.

4. PROVA DE CONCEITO

Com o intuito de avaliar a viabilidade da aplicação da abordagem proposta neste trabalho, uma prova de conceito (*Proof of Concept*- PoC) (CALDIERA; ROMBACH 2004) foi executada no contexto de utilização da arquitetura SciProvMiner para captura de proveniência de dois tipos de workflows, a saber, o *SimpleMathOperations*, que é um workflow de operações matemáticas, desenvolvido no Kepler, construído com o objetivo de ser o workflow piloto de testes para as funcionalidades implementadas no SciProvMiner, e o *Load*, que é um workflow apresentado pela equipe de SDSC[4], uma das dezesseis equipes participantes do *Third Provenance Challenge*, também desenvolvido no Kepler. Uma vez que a adequabilidade da arquitetura do SciProv para a coleta e gerência da proveniência de dados e processos no contexto de experimentos científicos distribuídos já foi validada em Valente (2011) e que o SciProvMiner estende esta arquitetura, o objetivo do presente capítulo é avaliar os benefícios adicionados à arquitetura SciProv propostos neste trabalho.

Desta forma, a seção 4.1 apresenta em detalhes como as funcionalidades do SciProvMiner foram utilizadas no contexto do workflow *SimpleMathOperations*, e a Seção 4.2 apresenta a captura e consulta aos dados de proveniência do workflow PC3 realizada pelo SciProvMiner e avalia quais os benefícios obtidos pela utilização desta abordagem em um workflow bastante utilizado pela comunidade de pesquisa em proveniência.

Os experimentos foram executados em um computador configurado com processador Intel Core I7, CPU 2.10 GHz cache 2MB, memória RAM 8 GB, memória de vídeo compartilhada, disco rígido de 1 TB e sistema operacional Microsoft Windows 7 Professional 64bits versão 2009 service pack 1.

4.1. WORKFLOW *SIMPLEMATHOPERATIONS*

Como primeira prova de conceito, o SciProvMiner foi utilizado na captura de proveniência de um workflow da área de matemática. Esta primeira prova de conceito tem por objetivo apresentar todas as funcionalidades de captura de proveniência disponibilizadas pelo SciProvMiner e demonstrar que com o uso de novas regras ontológicas e a combinação da captura de proveniência

[4] San Diego Supercomputer Center

prospectiva e retrospectiva em conjunto, a instrumentalização do workflow pôde ser substancialmente otimizada e a descoberta de novas informações de proveniência, a partir do processamento de inferências sobre a ontologia utilizada, é reforçada.

O workflow *SimpleMathOperations* é formado por três atividades e dois parâmetros de entrada fornecidos pelo usuário. A primeira tarefa *AddFunction*, recebe dois parâmetros numéricos como entrada e fornece como saída a soma dos valores passados nos parâmetros. A segunda tarefa, denominada *AbsoluteFunction* recebe o parâmetro vindo da tarefa anterior e retorna o valor *absolute* do valor de entrada. A terceira atividade, de nome *ExpFunction*, recebe o parâmetro advindo da tarefa anterior e calcula o valor exponencial do parâmetro de entrada. A Figura 4.1 ilustra a execução do workflow *SimpleMathOperations* desenvolvida do SGWfC Kepler, para os parâmetros de entrada -8 e 2.

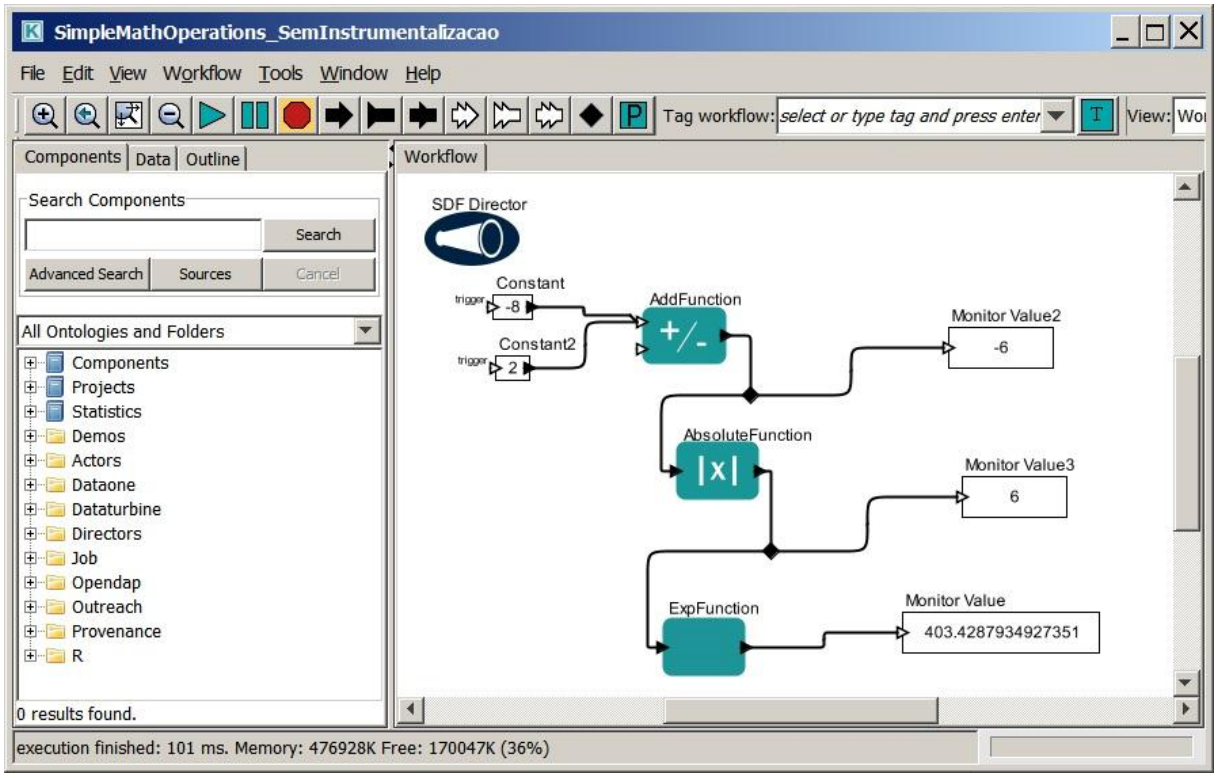


Figura 4.1. Execução do workflow *SimpleMathOperations* no SGWfC Kepler para os parâmetros de entrada -8 e 2.

Para uma instrumentalização plena do workflow *SimpleMathOperations*, garantindo uma cobertura completa das informações de proveniência prospectiva e retrospectiva que o workflow pode capturar, sem utilização de nenhum grau de otimização, seriam adicionados dezessete instâncias do serviço Web do SciprovMiner, como mostrado na Figura 4.2.

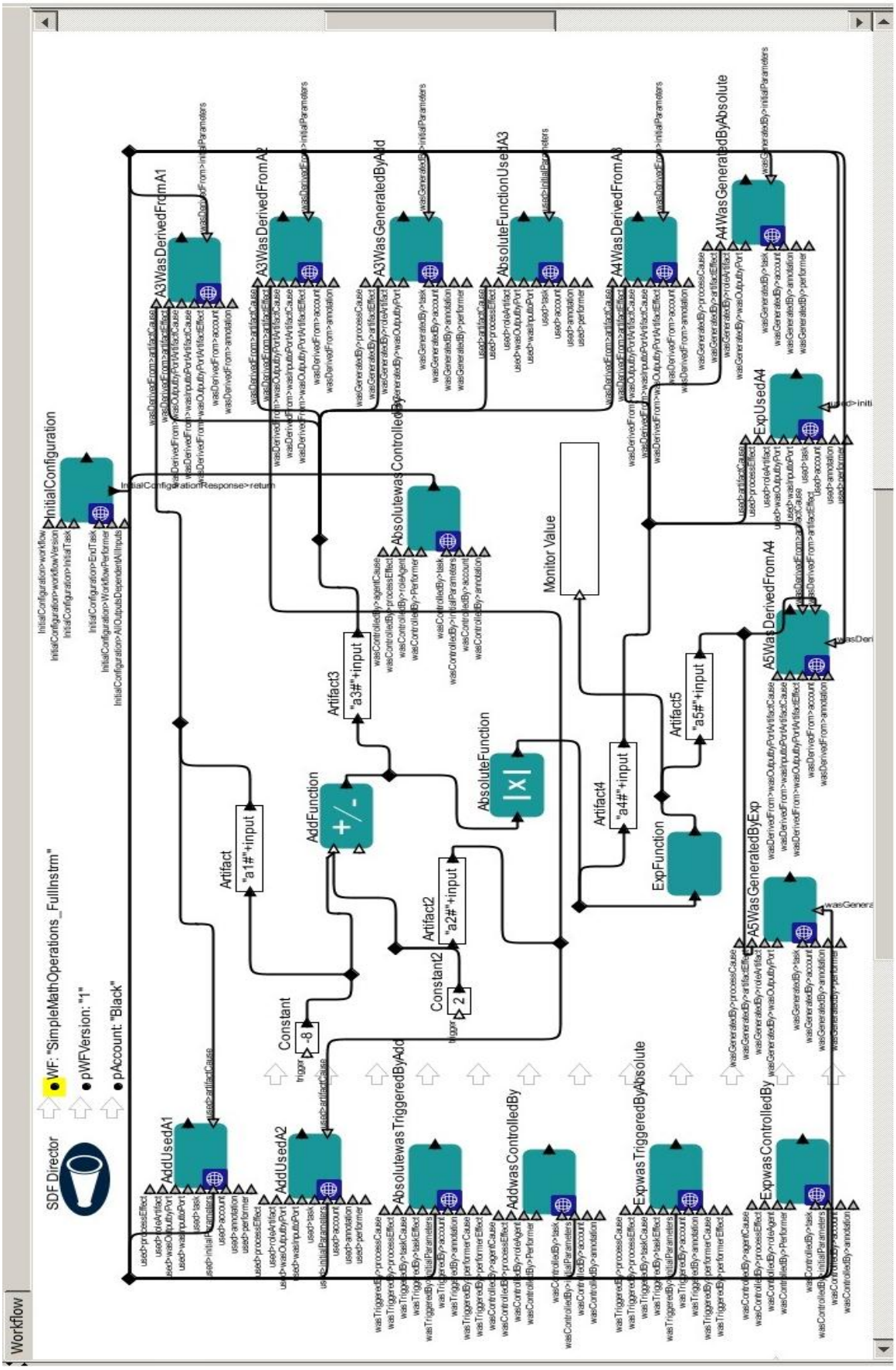


Figura 4.2. Instrumentalização completa do workflow *SimpleMathOperations*, sem utilização de nenhum grau de otimização.

Assim, na Figura 4.2, foram inseridas dezessete instâncias de serviço Web do SciProvMiner, sendo uma instância com o método *InitialConfiguration*, quatro instâncias com o método *used* para registrar que os processos usam cada um dos artefatos consumidos pelas atividades do workflow, três instâncias com o método *wasGeneratedBy*, para cada um dos artefatos gerados pela execução das atividades do workflow, quatro instâncias com o método *wasDerivedFrom*, considerando que cada artefato gerado por um processo depende de todos os seus artefatos de entrada, três instâncias com o método *wasControledBy*, associado a cada processo que é a execução de uma tarefa do workflow, e duas instâncias de *wasTriggeredBy*, permitidas pelos três processos, que estão associados às três atividades do workflow.

Após executar o workflow e ser processada a captura da proveniência pela execução dos serviços Web intrumentalizados, é possível gerar o grafo de causalidade da proveniência retrospectiva capturada, segundo o modelo OPM. Para isso, é necessário na interface inicial do SciProvMiner selecionar a opção Metadata (XML/RDF), escolher a execução do workflow para o qual se deseja realizar a operação e apertar o botão submit para que a serialização do grafo seja realizada. A Figura 4.3 ilustra esta operação.

Figura 4.3. Interface gráfica do SciProvMinerpara seleção do workflow a ser representado namemória

Após o sistema realizar a serialização do grafo de proveniência retrospectiva, segundo o modelo OPM, este fica disponível para ser exibido em PDF para o usuário. A Figura 4.4 exibe o grafo de proveniência gerado para o workflow *SimpleMathOperations-*

_FullInstrumentalization, onde os processos estão representados dentro de retângulos, os artefatos dentro de círculos, o agente dentro de octógonos, e as dependências causais são representadas pelas arestas, conforme o modelo OPM. Os rótulos das arestas tracejadas representam as roles segundo o modelo OPM.

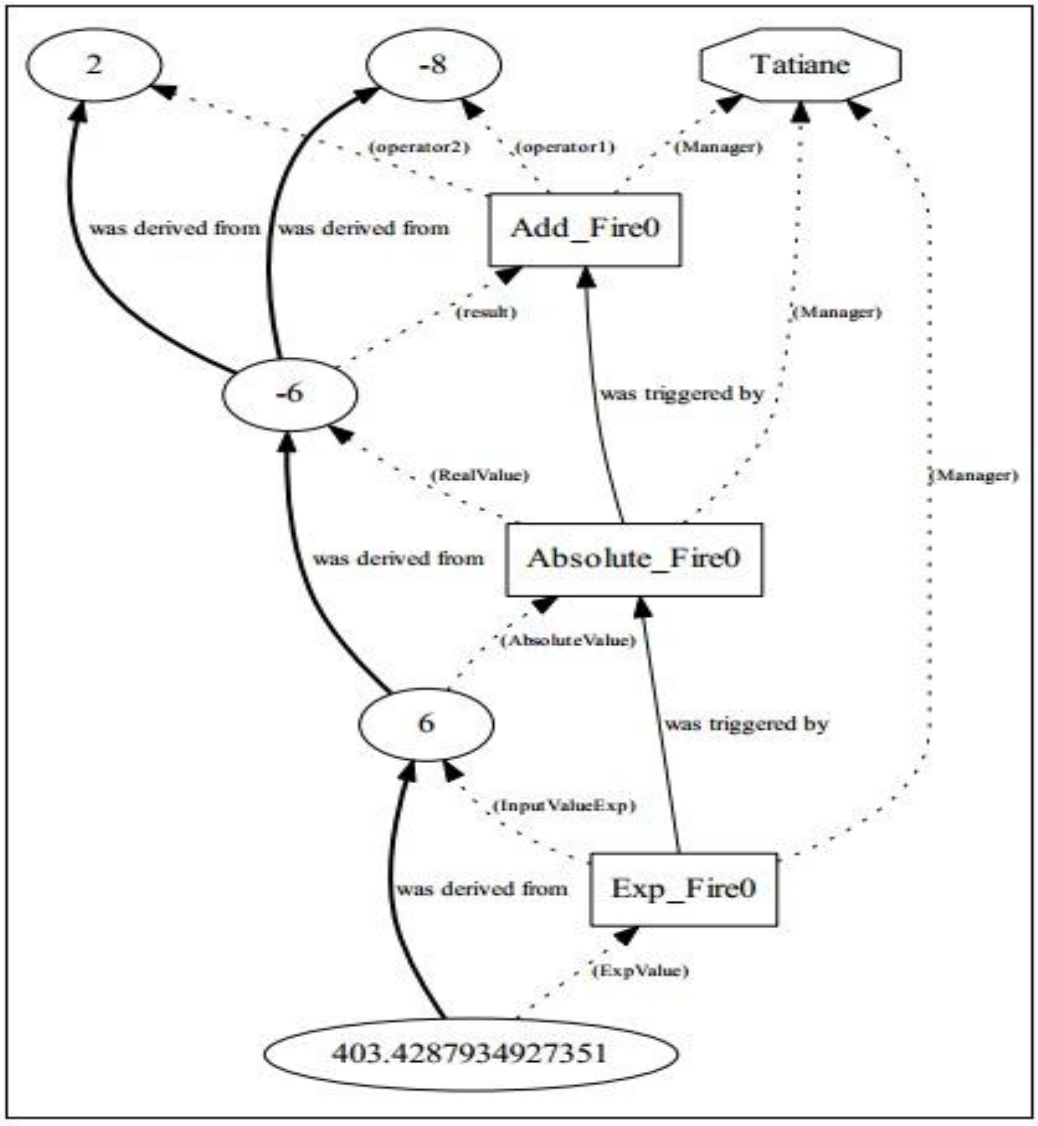


Figura 4.4. Representação visual do grafo de proveniência retrospectiva do workflow *SimpleMathOperations_FullInstrumentalization*

Uma vez que as informações de proveniência coletadas são armazenadas em uma base de dados relacional, a arquitetura do SciProvMiner oferece uma interface com o usuário para a consulta aos dados de proveniência a partir da linguagem padrão SQL (Figura 4.5).

A consulta solicitada na Figura 4.5 tem por finalidade recuperar as dependências causais do tipo *wasDerivedFrom* ocorridas no workflow *SimpleMathOperations_-*

FullInstrumentalization de versão 1. O resultado da consulta é apresentado na Figura 4.6, onde é retornado o *OPMGraphId*, indicando em qual execução do SciProvMiner ocorreu aquela dependência causal, o *artifactLabel* e *artifactValue* do artefato de causa e efeito respectivamente.

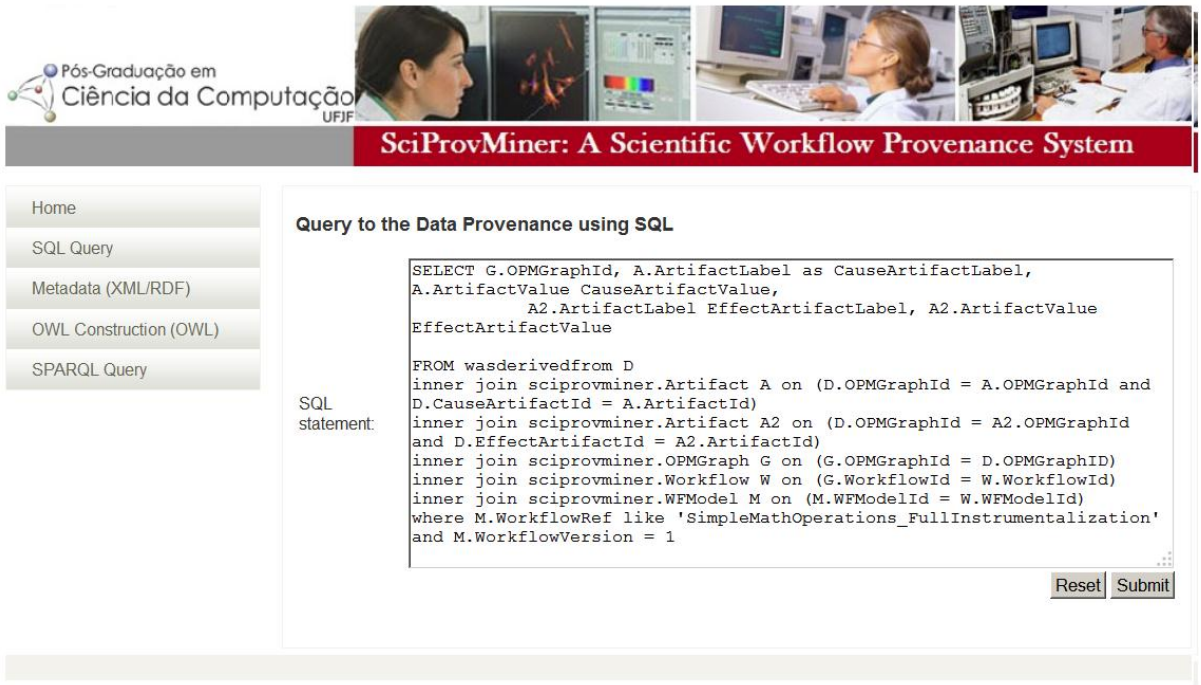


Figura 4.5. Interface gráfica do SciProvMiner para consulta SQL

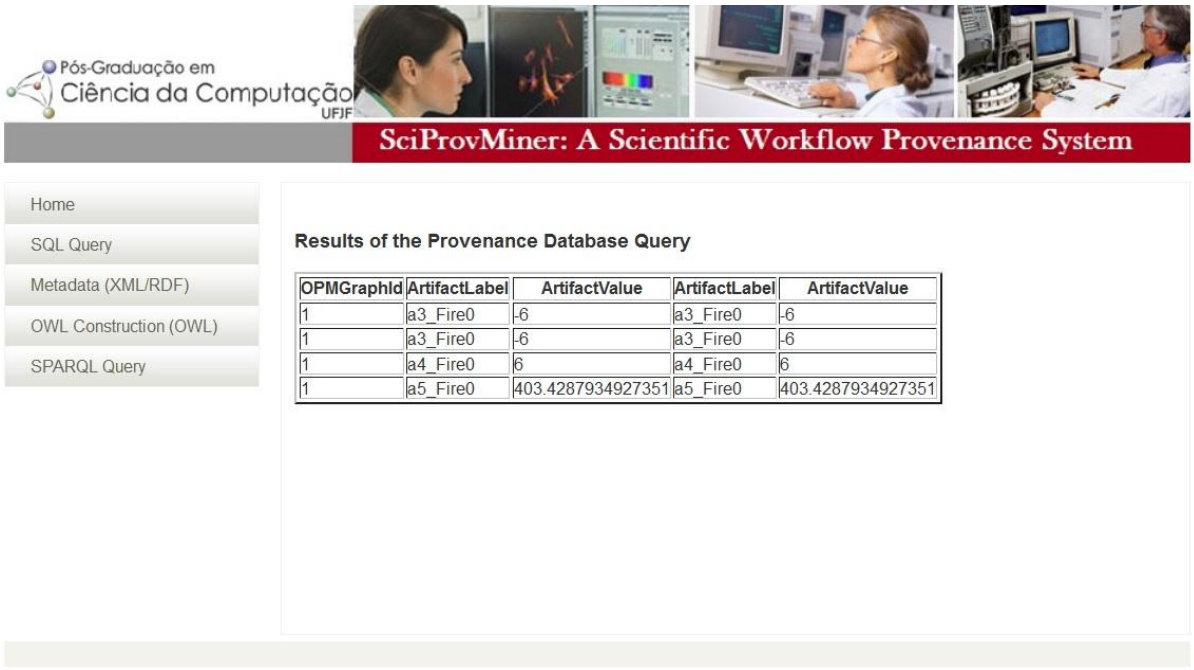


Figura 4.6. Interface gráfica do SciProvMiner resultado de consulta SQL

Considerando que a proposta para o desenvolvimento da arquitetura do SciProvMiner norteia-se pelo emprego de tecnologias Web semânticas relevantes e atuais, a arquiteturaimplementada utiliza a máquina de inferência do arcabouço Pellet (SIRIN et al. 2007), que provê suportepara perfis em OWL 2 — incluindo OWL 2 EL na versão 2.2.2.

A possibilidade de processar inferências desde OWL-DL até OWL 2 EL através doreasoner Pellet confere ao SciProvMiner expressividade em consultas SPARQLao grafo de proveniência modelado a partir de tecnologias Web semântica. O SciProvMiner oferece uma interface para que seja realizada a construção do arquivo OWL que modela a proveniência capturada de um determinado workflow na ontologia OPMO-e (Figura 4.7).

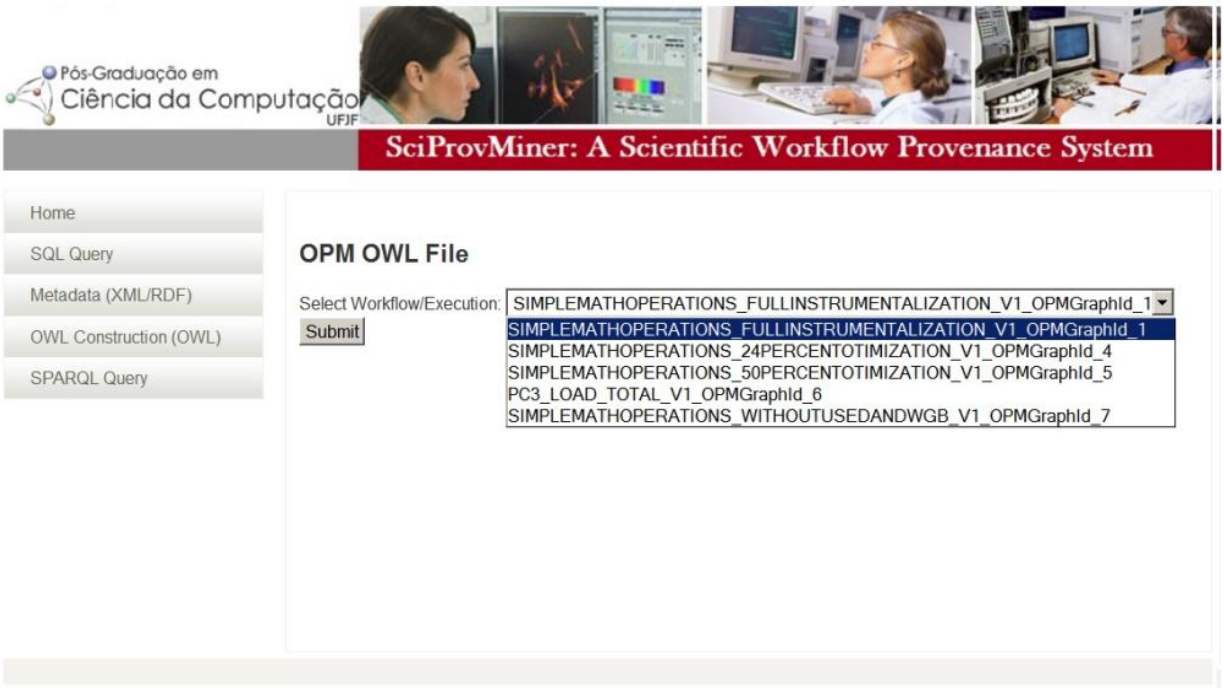


Figura 4.7. Interface gráfica do SciProvMiner para construção do arquivo OWL

Após o usuário selecionar o botão Submit da Figura 4.7 o sistema realiza a criação do arquivo OWL com base na ontologia OPMO-e, adicionando a esta ontologia os indivíduos que estão na base de dados relacional do SciProvMiner, que se referem ao workflow selecionado. A partir daí é possível realizar consultas ao grafo de proveniência gerado, e ser beneficidado do poder de expressividade da máquina de inferência, obtendo informações além daquelas explicitamente informadas pelos usuários.

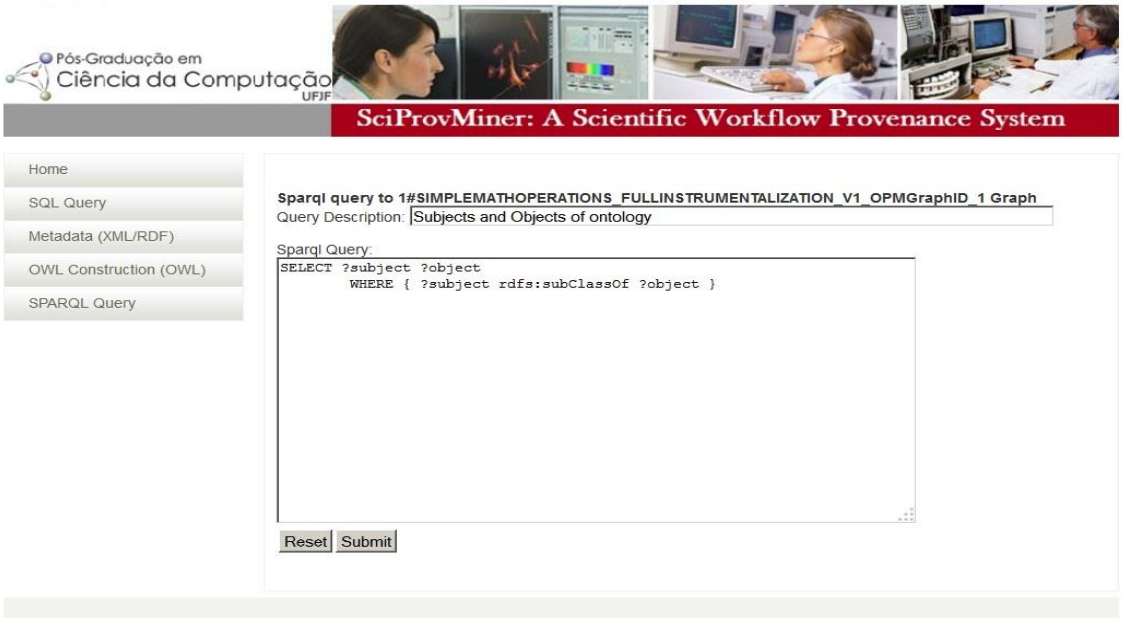


Figura 4.8. Interface gráfica do SciProvMiner para consultas SPARQL

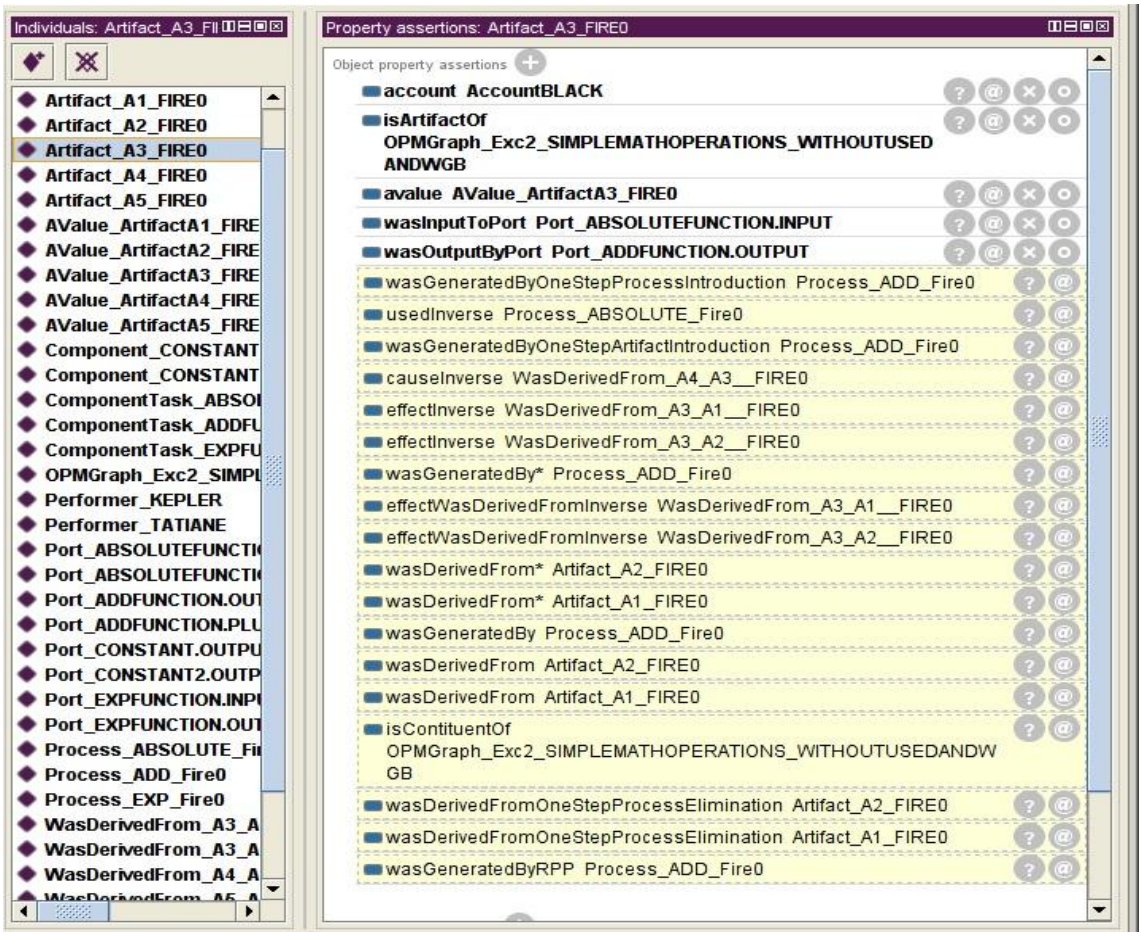


Figura 4.9. Interface gráfica do Protégé com a ontologia OPMO-e e os indivíduos do workflow *SimpleMathOperations_FullInstrumentalization*

O SciProvMiner oferece duas opções para o usuário poder explorar as informações obtidas a partir do grafo de proveniência gerado, a saber, através da utilização da interface para realizar consultas em SPARQL disponível no sistema do SciProvMiner (Figura 4.8), onde as inferências são realizadas utilizando o reasoner Pellet, ou utilizando a ferramenta Protégé, que é um editor de ontologia open source (DENTLER et al. 2011) onde pode ser aberto o arquivo OWL gerado pelo SciProvMiner, e o usuário pode fazer uso das funcionalidades disponíveis nesta ferramenta. Como a ontologia OPMO-eusa recursos da OWL 2.0, a versão da ferramenta Protégé tem que ser a partir da 4.02.

A Figura 4.9 exibe os indivíduos do workflow *SimpleMathOperations-_FullInstrumentalization* na ontologia OPMO-e, que foi gerado pela funcionalidade *OWL Construction* do sistema SciProvMiner. Os indivíduos se encontram do lado esquerdo da figura. Do lado direito em negrito se encontram as afirmações feitas explicitamente sobre o indivíduo selecionado, enquanto as inferências sobre este indivíduo se encontram destacadas em amarelo.

Considerando que a ferramenta Protégé tem interface amigável, e que o perfil de usuário do SciProvMiner são cientistas que não têm necessariamente o domínio da linguagem SPARQL, é interessante a disponibilização desta opção de utilização para o usuário, pois torna possível que ele possa extrair conhecimento útil de forma mais intuitiva. Para apresentar os resultados deste trabalho será utilizada esta ferramenta em sua versão 4.3 com o objetivo de facilitar a análise das inferências obtidas a partir das regras implementadas na ontologia OPMO-e.

Após ser aberto o arquivo OWL gerado para o workflow *SimpleMathOperations-_FullInstrumentalization* no Protégé 4.3, foram executadas várias máquinas de inferência sobre a ontologia tais como FaCT++[5], Hermit[6], Pellet[7] e RacerPro[8] (HEBELER et al. 2009), sendo que a que obteve melhor resultado em relação a tempo de processamento e inferências foi a Pellet.

Como pode ser visto na figura 4.9 muitas inferências foram feitas a respeito dos indivíduos da ontologia, tendo a máquina de inferência levado em consideração as regras e restrições

[5] http://owl.man.ac.uk/factplusplus
[6] http://kaon2.semanticweb.org
[7] http://pellet.owldl.com
[8] http://agraph.franz.com/racer

contidas na ontologia e as informações inseridas explicitamente sobre os indivíduos para gerar as inferências. Por questões de limitação de espaço serão analisados os indivíduos das classes que possuem alguma inferência relavante a ser analisada.

Para a ontologia relacionada ao workflow *SimpleMathOperations_FullInstrumentalization*, todas as regras ontológicas foram corretamente inferidas. Os indivíduos da classe definida *NoUsersInput* também foram inferidos corretamente. Na Figura 4.10 pode ser vista a inferência de todos os artefatos que não foram informados pelo usuário, inferidos como indivíduos da classe *NoUsersInput*. Para se tornar membro da classe *NoUsersInput* o indivíduo tem que ser da classe *Artifact* e tem que se relacionar com algum indivíduo da classe *Process* através da propriedade *wasGeneratedBy*. Conforme pode ser observado na Figura 4.10 os indivíduos *Artifact_A3_FIRE0, Artifact_A4_FIRE0* e *Artifact_A5_FIRE0* atenderam aos requisitos e foram inferidos nesta classe. Estas inferências são úteis ao usuário que está analisando o resultado do experimento científico por informá-lo quais foram os artefatos gerados por processos e não informados explicitamente no workflow.

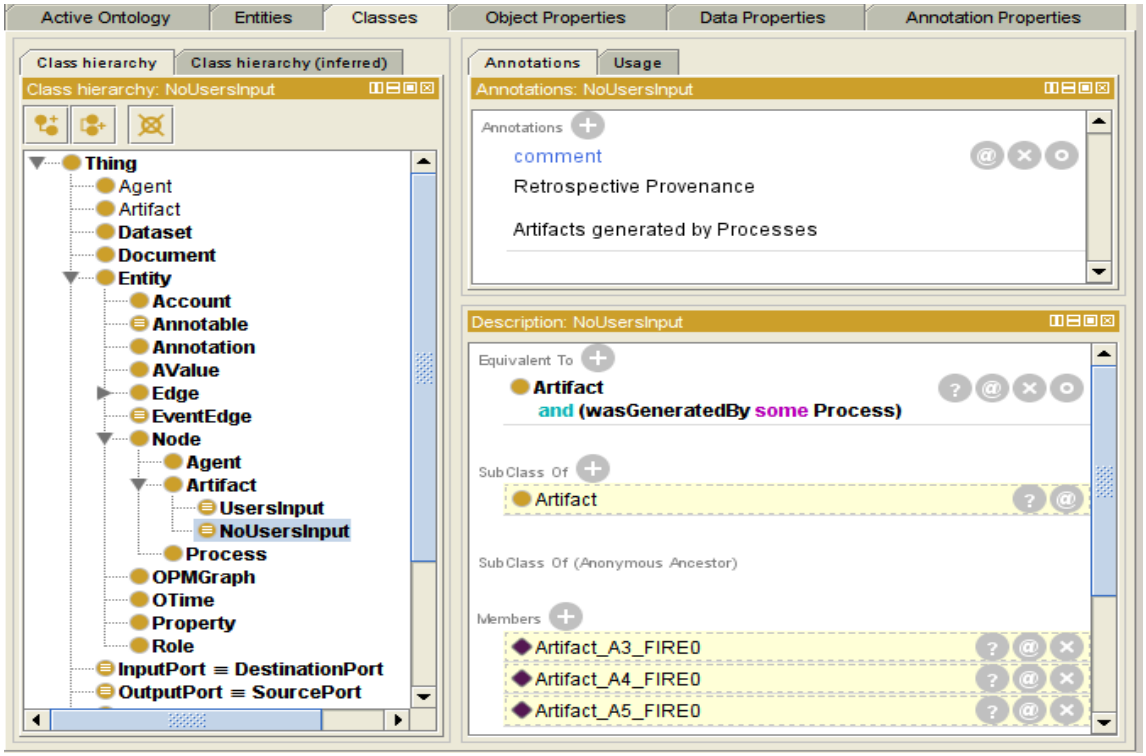


Figura 4.10. Inferência de indivíduos na classe definida NoUsersInput

A regra de completude Introdução de Processo (Figura 2.3) definida na documentação do modelo OPM (MOREAUet al. 2011) e implementada no SciProvMiner através de regra ontológicatambém está sendo corretamente inferida na ontologia OPMO-ecom os indivíduos

deste workflow. Um exemplo pode ser visto na Figura 4.11, onde para o artefato *Artifact_A3_FIRE0* existe a inferência "*Artifact_A3_FIRE0 wasGeneratedByOneStep-ProcessIntroductionProcess_ADD_ Fire0*" e para o processo *Process_ADD_Fire0"* existe a inferência "*Process_ADD_Fire0usedOneStepProcessIntroductionArtifact_A1_FIRE0"* e a inferência "*Process_ADD_ Fire0usedOneStepProcessIntroductionArtifact_A2_FIRE0"*.

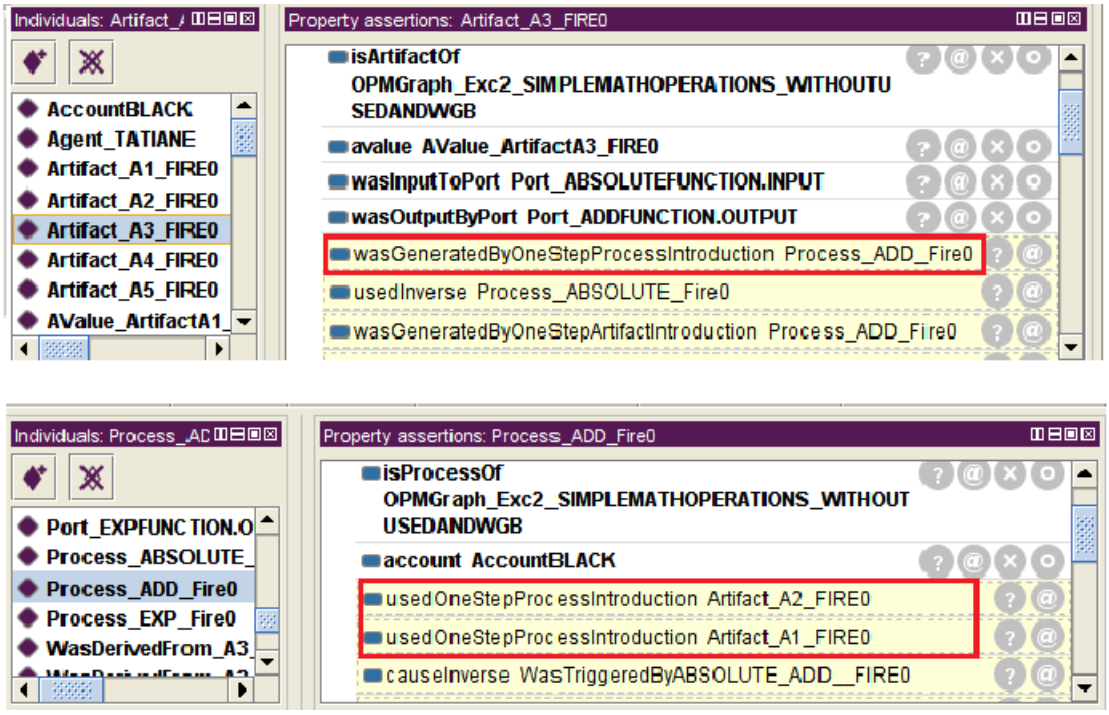


Figura 4.11. Validação da Inferência para regra de completude *Introdução de Processo*

A inferência das informações relacionadas à regra de completude Introdução de Processo definidas na documentação do modelo OPM, pelas regras ontológicas construídas na ontologia OPMO-e representam um ganho de informação considerável para o usuário do SciProvMiner. Visto que na regra de completude de Introdução de Processo na documentação do modelo OPM (MOREAU et al. 2011) é dito que uma aresta do tipo *wasDerivedFrom*esconde a presença de um processo P intermediário, do qual o artefato destino da dependência causal *wasDerivedFrom* foi gerado por este processo P, e o artefato origem da dependência causal *wasDerivedFrom* foi usado por este processo, e que as regras ontológicas construídas no SciProvMiner permitem inferir qual é esse "processo escondido", que no exemplo da Figura é o *Process*_ADD_Fire0, mostra o poder do SciProvMiner em inferir conhecimento não explícito para o usuário. Estas inferências informam ao usuário sobre quais foram às dependências causais indiretas responsáveis pela derivação de um artefato em outro.

A inversa da regra Introdução de Processo, denominada Eliminação de Processo, só é válida quando é declarado na instrumentalização do workflow que todos os dados de saída de seus processos dependem de suas entradas através do parâmetro *AllOutputsDependentAllInputs* do método *InitialConfiguration* do serviço Web de instrumentalização, conforme explicado no APÊNDICE A. Para o presente workflow esta regra é válida, pois a declaração necessária foi feita na intrumentalização do workflow. Esta regra ontológica é implementada através da propriedade *wasDerivedFromOneStep-ProcessElimination,* e pelo exemplo exibido na Figura 4.12 fica demonstrado que esta regra foi corretamente inferida, pois para o artefato *Artifact_A3_Fire0* existe a inferência "*Artifact_A3_FIRE0 wasDerivedFromOneStepProcessEliminationArtifact_A1_FIRE0*" e a inferência "*Artifact_A3_FIRE0 wasDerivedFromOneStepProcessEliminationArtifact-_A2_FIRE0*", que condiz com a regra de completude definida no modelo OPM. Essa regra informa ao usuário que o *Artifact_A3_FIRE0* foi derivado do (*wasDerivedFrom*) *Artifact_A1_FIRE0* através da regra de completude Eliminação de Artefato, por existir entre esses artefatos um processo que foi eliminado, do qual o artefato *Artifact_A3_FIRE0*foi gerado (*wasGeneratedBy*) e pelo qual o artefato *Artifact_A2_FIRE1* foiusado. A inferência do relacionamento do *Artifact*_A3_FIRE0 como o *Artifact_A2_FIRE0*pela propriedade *wasDerivedFrom*StepProcessElimination se desenvolveu da mesma maneira. É bom salientar que mesmo em workflows que não forem instrumentalizados com as dependências causais *wasDerivedFrom*, essa inferência vai acontecer.

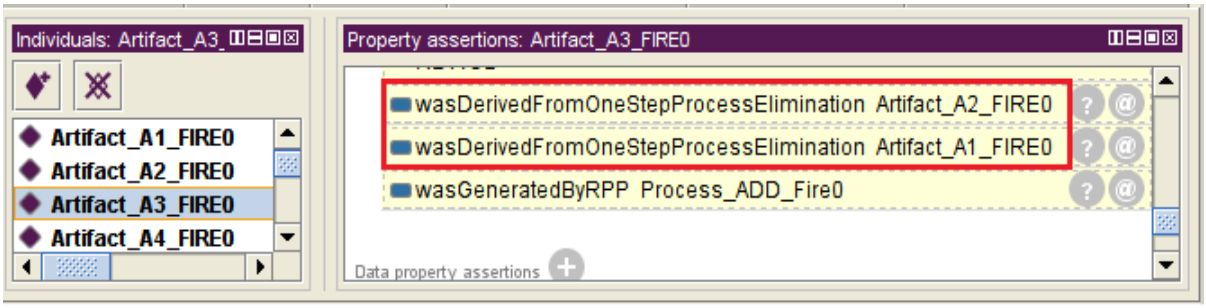


Figura 4.12. Validação da Inferência para regra de completude *Eliminação de Processo*

A inferência da regra de completude Introdução de Artefato definida na documentação do modelo OPM e implementada no SciProvMiner através de regra ontológica, também foi validada para o workflow em análise. A Figura 4.13 apresenta um exemplo de inferência desta regra, onde para o processo *Process_EXP_Fire0* existe a inferência "*Process_EXP_Fire0usedOneStepArtifactIntroductionArtifact_A4_FIRE0"* e para o artefato

*Artifact*_A4_Fire0 existe a inferência "*Artifact_A4_FIRE0 wasGeneratedByOneStep-ArtifactIntroductionProcess_ABSOLUTE_Fire0*". As inferências estão destacadas na Figura 4.12 com um retângulo. Essas inferências informam ao usuário que o processo *Process_EXP_Fire0* foi desencadeado (*wasTriggeredBy*) pelo processo *Process_ABSOLUTE_ Fire0*, pelo fato do processo *Process_EXP_ Fire0* usar *(used)* o artefato*Artifact_A4_FIRE0* e pelo fato deste artefato ser gerado *(wasGeneratedBy)* pelo processo *Process_ABSOLUTE_Fire0*. Esta informação também representa um ganho considerável de conhecimento para o usuário, pois implementa a regra de Introdução de Artefato, definida no modelo OPM, onde é dito ser possível estabelecer que uma aresta do tipo "P1 *wasTriggeredBy* P2" está escondendo a existência de um artefato usado por P1 e gerado por P2. Pelas regras implementadas na ontologia OPMO-e utilizada no SciProvMiner se torna possível inferir qual é este "artefato escondido" que foi usado por P1 e gerado por P2. No exemplo da Figura 4.13, o artefato é o *Artifact_A4_FIRE0* que foi usado pelo processo*Process_EXP_ Fire0*egerado pelo processo *Process_ABSOLUTE_ Fire0*.

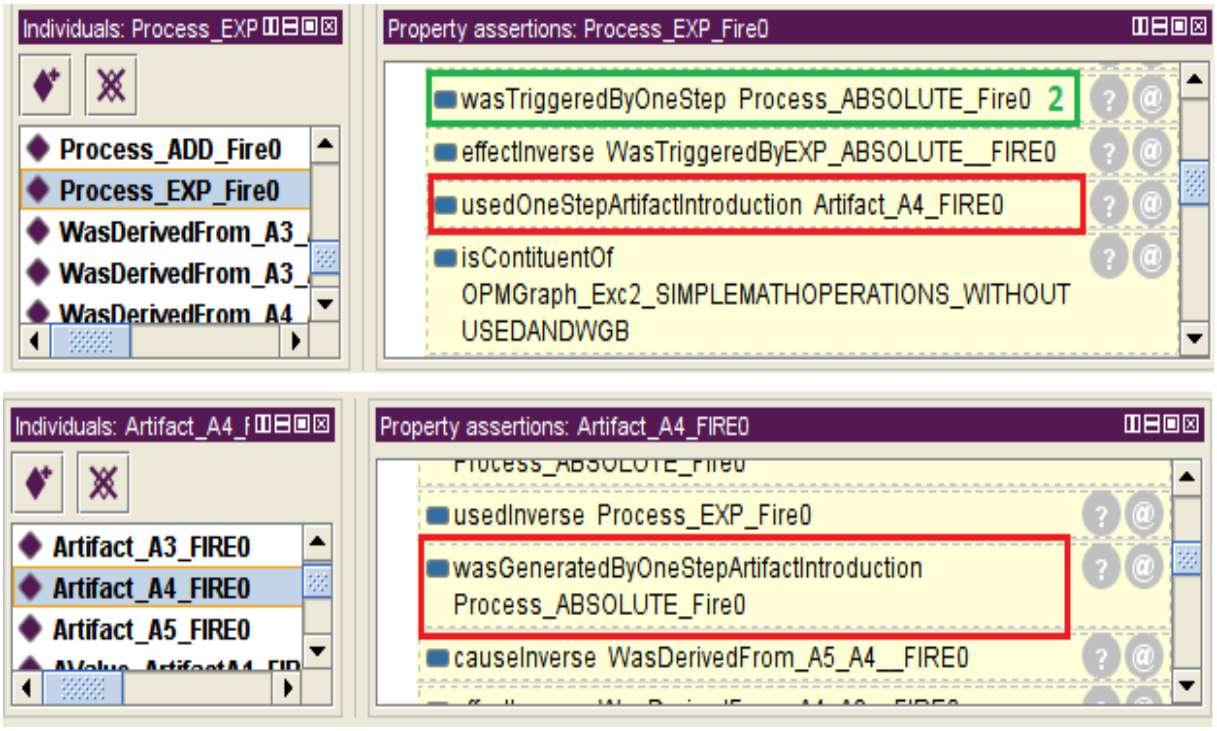


Figura 4.13. Validação da Inferência para regra de completude Introdução e Eliminação de Artefato

A validação da inferência da regra de completude *Eliminação de Artefato* definida na documentação do modelo OPM (MOREAU et al. 2011) também foi realizada na ontologia OPMO-e com os indivíduos do workflow *SimpleMathOperations_FullInstrumentalization*.

Pelo exemplo apresentado na Figura 4.13, na inferência destacada com um retângulo e o número "2", pode ser observado que para o processo *Process_EXP_ Fire0* existe a inferência "*Process_EXP_Fire0 wasTriggeredByOneStepProcess_ABSOLUTE_Fire0"* que é a inferência inversa da regra de completude anteriormente apresentada Introdução de Artefato. Essa regra infere para o usuário, baseado na regra de completude de Eliminação de Artefato definido no modelo OPM (MOREAU et al. 2011), que o processo*Process_EXP_Fire0* foi desencadeado pelo processo *Process_ABSOLUTE_ Fire0* por existir entre eles um artefato, que foi usado pelo processo*Process_EXP_Fire0* e foi gerado (*wasGeneratedBy*) pelo Processo *Process_ABSOLUTE-_Fire0*. É bom salientar que mesmo em workflows que não forem instrumentalizados com as dependências causais *wasTriggeredBy*, essa inferência vai acontecer.

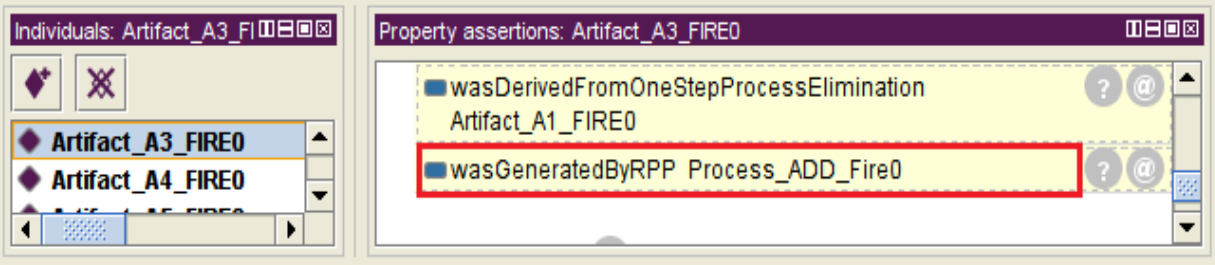


Figura 4.14. Validação da Inferência para a regra *wasGeneratedByRPP*

A regra *wasGeneratedByRPP* definida na ontologia OPMO-e também foi corretamente inferida para os indivíduos do workflow que está sendo analisado. Um exemplo é mostrado na Figura 4.14, onde para o artefato *Artifact_A3_FIRE0* existe a inferência "*Artifact_A3_FIRE0 wasGeneratedRPP Process_ADD_ Fire0*". Esta regra foi implementada na ontologia OPMO com o objetivo de tornar possível que seja inferida a dependência causal *wasGeneratedBy* entre um artefato e um processo mesmo que esta dependência causal não tenha sido explicitamente informada pelo usuário. Para que esta regra seja inferida basta que a proveniência prospectiva seja capturada corretamente, bem como os artefatos e processos da proveniência retrospectiva, uma vezque esta propriedade é baseada na regra "*wasOutputByPort o isPortOf o isTaskOf*", que relaciona indivíduos *Port* e *Task* da proveniência prospectiva com indivíduos do tipo *Artifact* e *Process* da proveniência retrospectiva. Esta inferência prove o conhecimento ao usuário sobre qual o processo que gerou um determinado artefato, mesmo que esta informação não tenha sido explicitamente informada pelo usuário.

A Figura 4.15 mostra as inferências para a Tarefa *ComponentTask_ADDFUNCTION*. Nesta Figura pode ser visto que para o componente *ComponentTask_ADDFUNCTION* foram inferidas as portas que ele possui pela propriedade *hasPort*, a qual processo esta tarefa está relacionada pela propriedade *isTaskOf*, quais são suas portas de entrada, que no caso é uma única, pela propriedade componentHasInputPort, a porta de saída pela propriedade *componentHasOutputPort*, infere que esta tarefa é a tarefa inicial do workflow, pela propriedade *isInitialTask*. As inferências até aqui mostradas para o componente *ComponentTask_ADDFUNCTION* foram realizadas por serem propriedades que são inversas de propriedades declaradas explicitamente. Muitas dessas propriedades são utilizadas na formação de outras, como explicado no capítulo 3.

Já as propriedades *ConectedComponents*, que inferem a quais componentes um componente está conectado, *likeDestinationComponent*, que infere de quais componentes o *ComponentTask_ADDFUNCTION* é sucessor, a propriedade *likeDestinationComponent**, que realiza a inferência transitiva da propriedade *likeDestinationComponent*, a propriedade *likeSourceComponent*, que infere de quais componentes o componente selecionado é antecessor e a propriedade *likeSourceComponent** que realiza a inferência transitiva da propriedade *likeSourceComponent*, são propriedades contruídas a partir de cadeia de propriedades com o objetivo de enriquecer o conhecimento do usuário acerca da proveniência prospectiva capturada.

Na Figura 4.15 pela propriedade *connectedComponents* foi inferido que o componente selecionado está diretamente conectado aos componentes *Component_CONSTANT*, *componentTask_ABSOLUTEFUNCTION*, component_ CONSTANT2. Esta propriedade traz o conhecimento para o usuário sobre quais componentes estão relacionados diretamente com um dado componente.

Pela propriedade *likeSourceComponent* o usuário fica informado que o componente selecionado é antecessor direto do componente *componentTask_ABSOLUTEFUNCTION* (Figura 4.15). Esta propriedade dá a conhecer o usuário de quais são os componentes sucessores do componente selecionado.

Através da propriedade *likeSourceComponent** são inferidos quais são os componentes que são sucessores de forma direta ou indireta do componente selecionado. No exemplo da Figura 4.15, o componente *componentTask_ADDFUNCTION se relaciona com os componentescomponentTask_ ABSOLUTEFUNCTION* e*componentTask_ EXPFUNCTION*

*através da propriedade likeSourceComponent**, trazendo a informação de quais são os componentes sucessores diretos ou indiretos do componente selecionado.

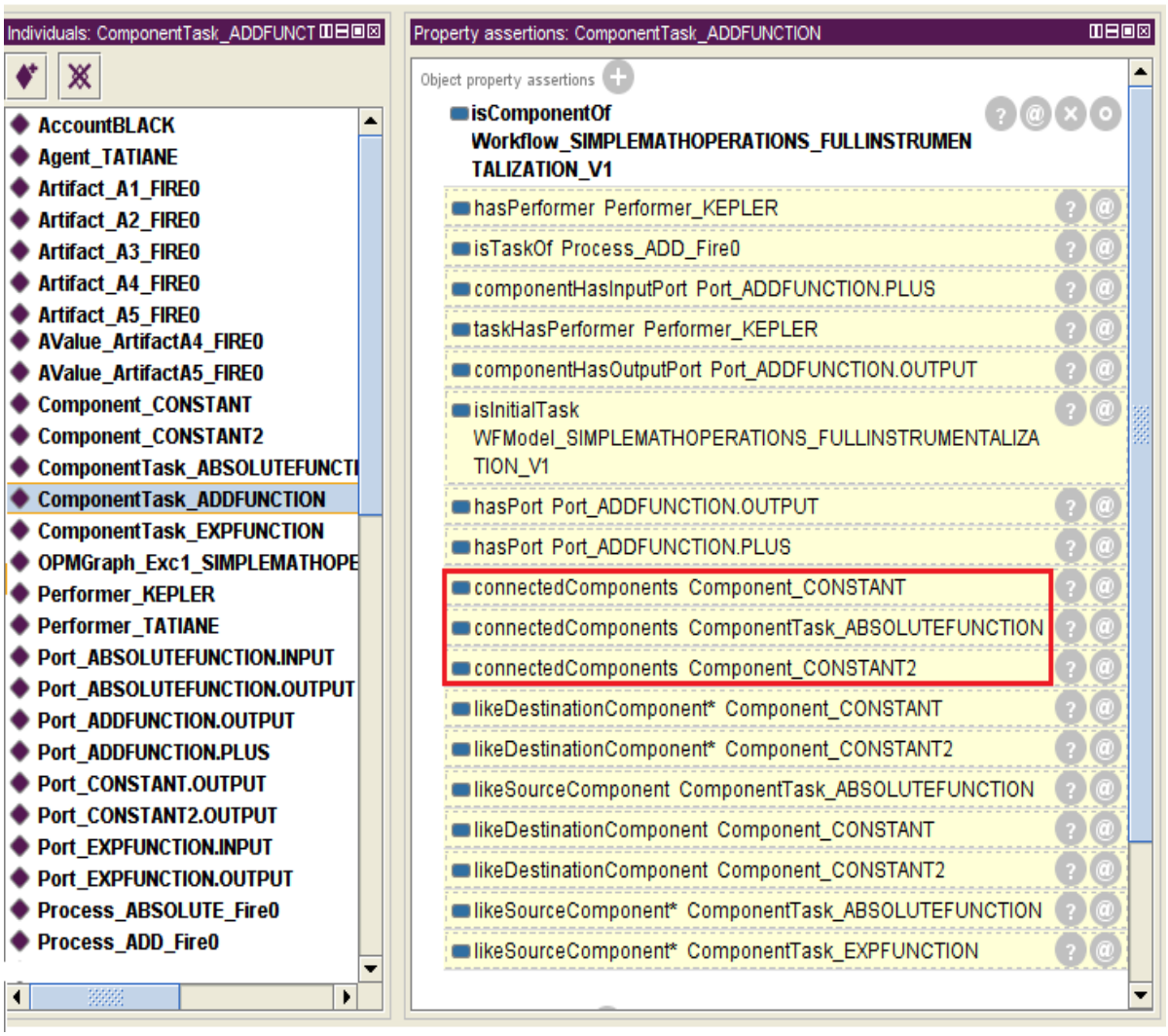


Figura 4.15. Validação da Inferência para as propriedades connectedComponents, *likeSourceComponent* e *likeDestinationComponent*.

Pela propriedade *likeDestinationComponent* na Figura 4.15 o usuário tem a informação de que o componente selecionado é sucessor direto dos componentes *Component_CONSTANT*, e *Component_CONSTANT2*, trazendo a informação ao usuário dos componentes que são antecessores no fluxo do componente selecionado. A propriedade *likeDestinationComponent** informa ao usuário quais são os componentes antecessores diretos e indiretos do componente que está sendo analisado.

As inferências em múltiplos passos *wasDerivedFrom**, *wasGeneratedBy**, *used** e *wasTriggeredBy** definidas na documentação do modelo OPM (MOREAU et al. 2011), também foram corretamente inferidas. A Figura 4.16 mostra um exemplo das inferências em

múltiplos *wasDerivedFrom** e *wasGeneratedBy** para o artefato *Artifact_A4_FIRE_0*, e a Figura 4.17 apresenta um exemplo das inferências em múltiplos passos *used** e *wasTriggeredBy**.

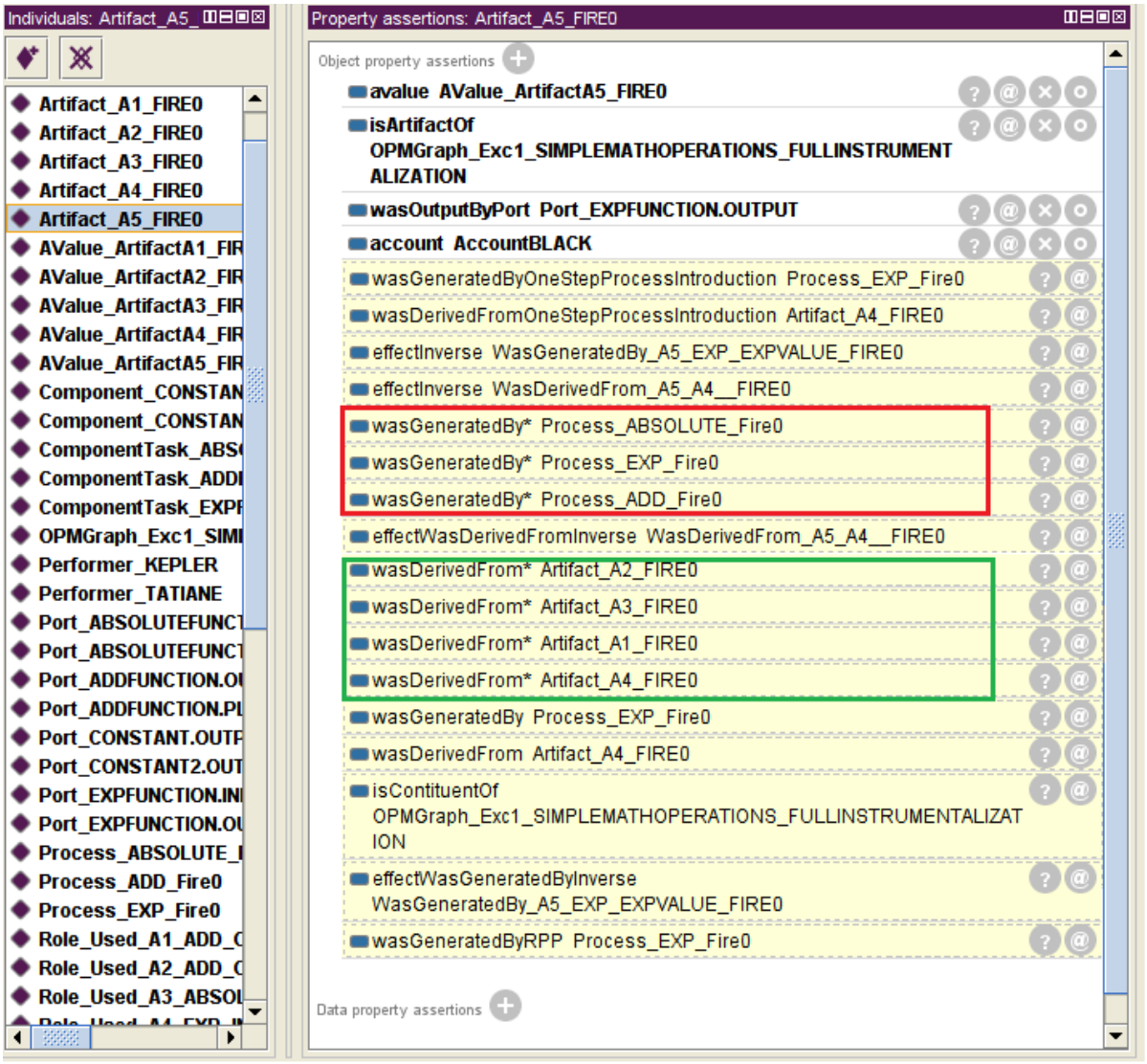


Figura 4.16. Validação da Inferência para as propriedades *wasGeneratedBy e *wasDerivedFrom****

A Figura 4.16 mostra que o artefato *Artifact_A5_FIRE_0* possui as inferências relacionadas à propriedade "*wasGeneratedBy** com os processos *Process_ADD_Fire0*", *Process_ABSOLUTE_Fire0 e Process_EXP_Fire0*. Essas inferências respondem ao usuário a questão sobre quais foram as causas indiretas pelas quais o artefato *Artifact_A5_FIRE_0 foi gerado*. A inferência "*wasDerivedFrom**" da Figura 4.16 está relacionada a todos os artefatos que fazem parte direta ou indiretamente da formação do artefato *Artifact_A5_FIRE_0*, de tal maneira que esta regra pode responder ao usuário sobre quais as causas indiretas pelas quais o

artefato *Artifact_A5_FIRE_0*foi gerado, conforme regra especificada na documentação do modelo OPM (MOREAU et al. 2011) sobre a derivação em múltiplos passos da dependência causal *wasDerivedFrom*.

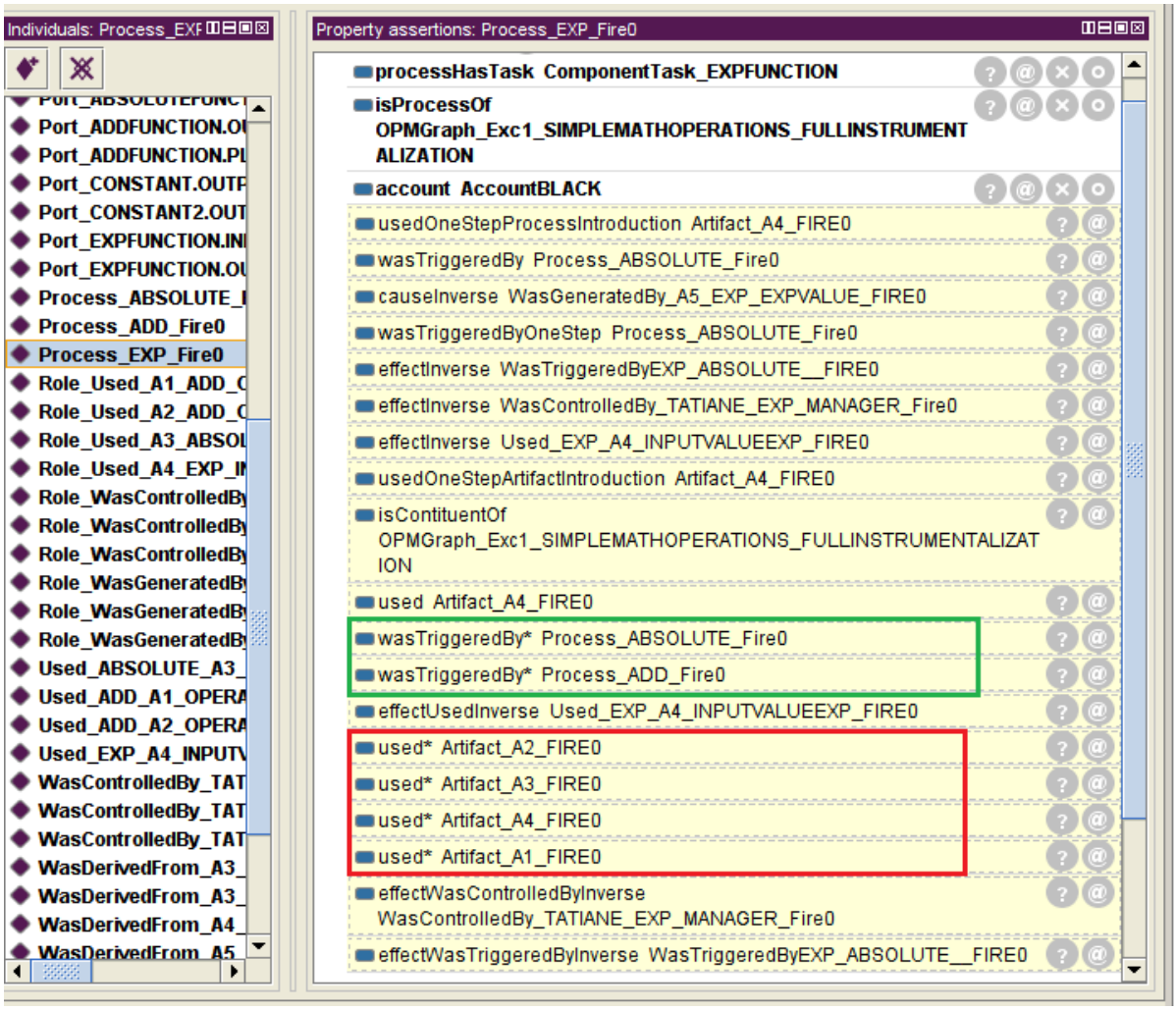


Figura 4.17. Validação da Inferência para as propriedades *used* e wasTriggeredBy**

A Figura 4.17 ilustra que foi inferido para o processo PROCESS_EXP_Fire0 o relacionamento com os artefatos *Artifact_A1_FIRE_0, Artifact_A2_FIRE_0, Artifact_A3_FIRE_0, Artifact_A4_FIRE_0* através da propriedade*used**. Essas inferências fornecem ao usuário o conhecimento sobre os artefatos que direta ou indiretamente foram usados pelo processo PROCESS_EXP_Fire0, visto que o usuário pode não estar interessado apenas nas inferências diretas, mas também nas indiretas. Esta Figura 4.17 também apresenta o relacionamento do processo PROCESS_EXP_Fire0 com os processos *Process_ABSOLUTE_Fire0* e *Process_ADD_Fire0* através da propriedade *wasTriggeredBy**, dando o conhecimento ao usuário de que o processo PROCESS_EXP_Fire0 foi disparado

direta ou indiretamente pelos processos com os quais ele se relaciona nesta propriedade. Essas inferências estão de acordo com a proveniência capturada do workflow em análise.

Através dos exemplos acima exibidos pode-se verificar que a utilização de recursos da Web semântica tais como ontologia e máquina de inferência pelo SciProvMiner confere aos usuários conhecimento que não foi explicitamente informado por ele, atingindo assim o objetivo da arquitetura proposta neste trabalho, que é fornecer ao usuário informações novas e úteis acerca da proveniência de dados do experimento científico além daquelas explicitamente informadas. Além disso, pela capacidade de inferência atribuída ao SciProvMiner, através da utilização de ontologia e máquina de inferência, é possível fazer otimizações na instrumentalização do workflow, diminuindo a possibilidade de introdução de erro na coleta da proveniência, por ser uma tarefa manual e o trabalho do cientista em instrumentalizar o workflow e continuar sendo inferidas informações de proveniência implícitas.

Existem níveis de otimização que devem ser considerados. O primeiro nível leva em consideração apenas a otimização proposta em relação à sublimação da instrumentalização das dependências causais *wasGeneratedBy* do workflow, nos casos em que essa regra é válida, conforme explicitado na seção 3.2.4.3. Tomando por base o workflow *SimpleMathOperations* completamente instrumentalizado mostrado na Figura 4.2, as três instâncias de instrumentalização de serviço Web relacionadas ao método *wasGeneratedBy* deixam de ser necessárias, otimizando em aproximadamente 18% a instrumentalização do workflow original. Em um segundo nível de otimização, é considerada a otimização garantida pela implementação da regra de completude denominada Eliminação de Artefato, apresentada na seção 3.1.3, onde deixa de ser necessária a instrumentalização das instâncias com método *wasTriggeredBy*, por serem inferidas pela regra, acumulando um total de otimização de cinco instâncias de instrumentalização que deixaram de ser necessárias, o que representa aproximadamente 29,5% da instrumentalização total do workflow. A Figura 4.18 apresenta a instrumentalização do workflow *SimpleMathOperations* com as otimizações acima citadas.

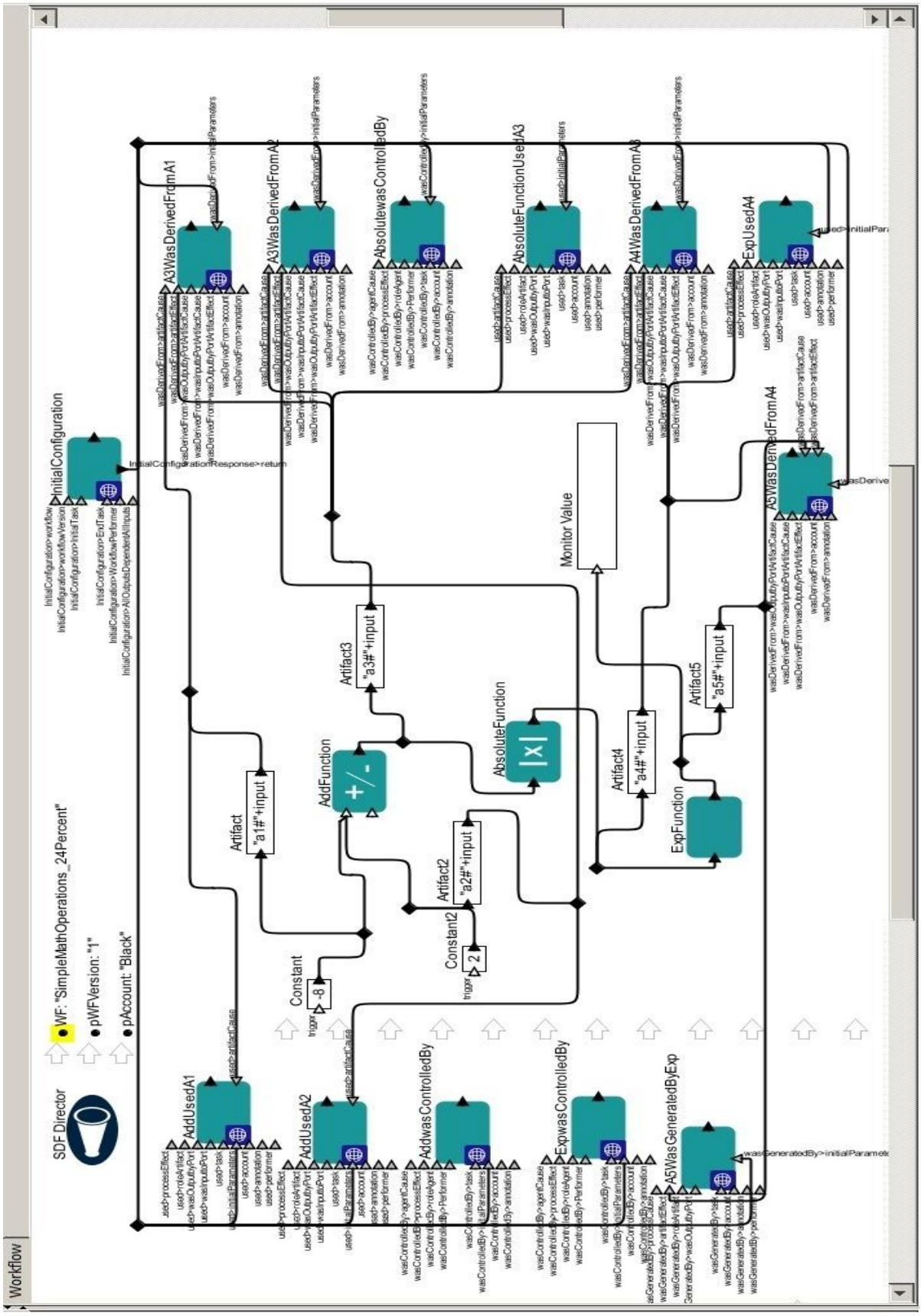


Figura 4.18. Instrumentalização considerando otimização da aresta *wasGeneratedBy* e da regra de completude denominada Eliminação de Artefato proposta em Moreau et al. (2011).

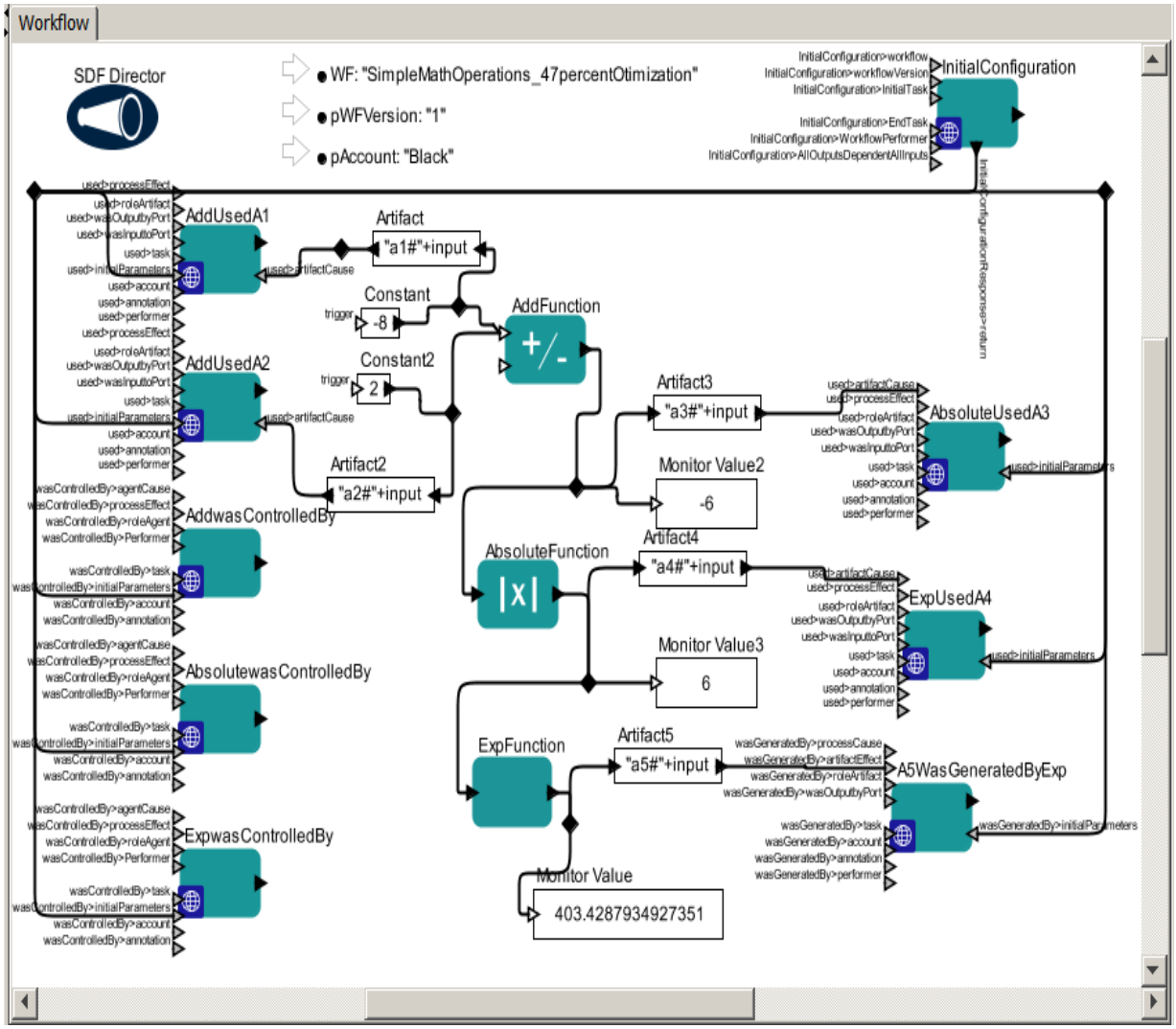


Figura 4.19. Instrumentalização do workflow *SimpleMathOperations* no terceiro nível de otimização.

Um terceiro nível de otimização se dá quando a regra de completude Eliminação de Processo apresentada anteriormente neste trabalho é válida para o workflow que está sendo instrumentalizado, como acontece no presente workflow. Nesses casos é possível sublimar as instâncias de instrumentalização com o método *wasDerivedFrom*, que no workflow em análise são quatro. No entanto, para capturar as informações de proveniência prospectiva relacionadas à porta de saída da tarefa *ExpFunction* é necessário que uma instância de *wasGeneratedBy* ou de *wasDerivedFrom* seja colocada com esta finalidade. Portanto, o total de instâncias de instrumentalização a menos deste workflow é de oito, correspondendo a aproximadamente 47% de otimização em relação à instrumentalização completa do workflow. A Figura 4.19 ilustra o workflow *SimpleMathOperations* instrumentalizado com as

otimizações propostas, onde foram necessárias apenas oito instanciações do serviço *Web* do SciProvMiner para instrumentalizar o workflow.

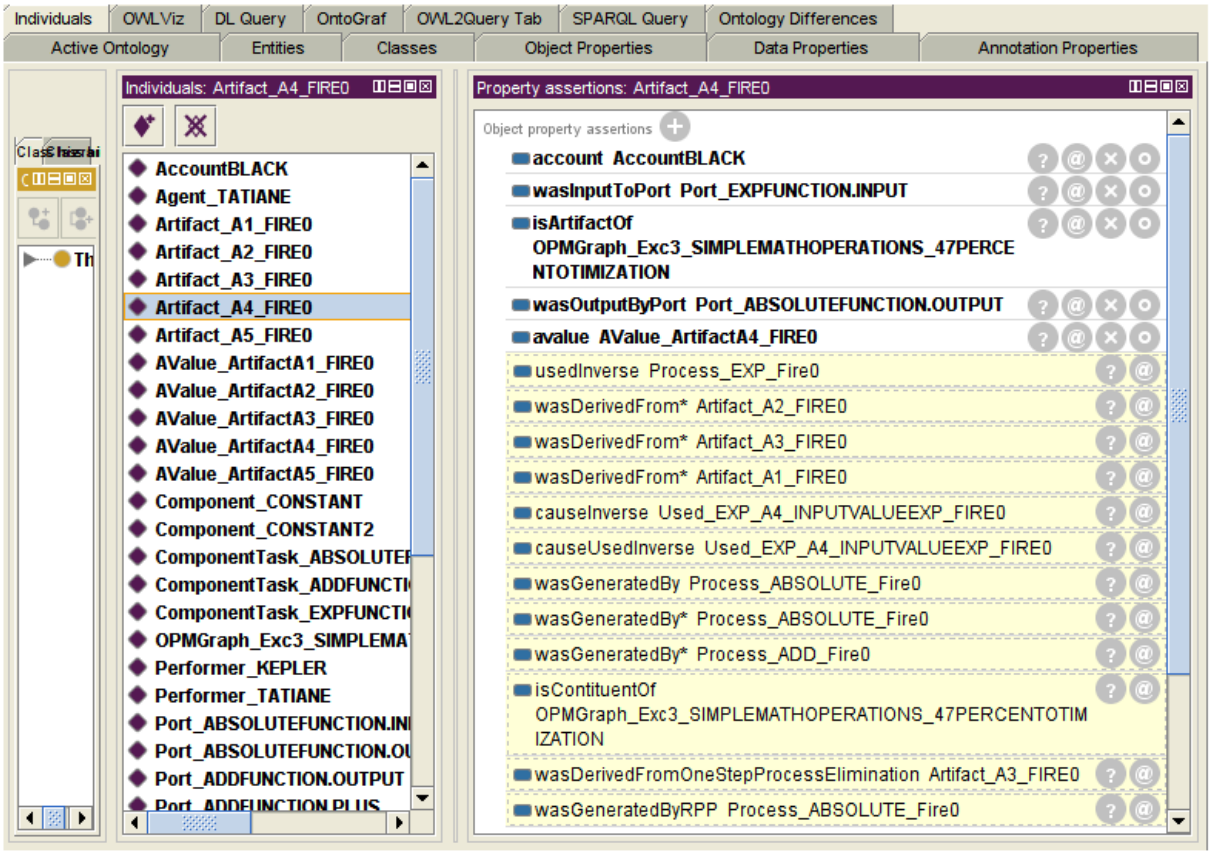


Figura 4.20. Inferências para o artefato *Artifact*_A4_FIRE0 considerando o workflow instrumentalizado no terceiro nível.

Apesar de esta opção garantir a maior otimização, existe uma determinada perda de informação, pois, devido às limitações relacionadas à transitividade da OWL-DL, não foi possível adicionar as regras *wasDerivedFromOneStepProcessElimination* como subpropriedade de *wasDerivedFrom*, nem a regra *wasTriggeredByOneStep* como subpropriedade de *wasTriggeredBy* e, portanto, as regras que usam em sua formação as propriedades *wasDerivedFrom* e *wasTriggeredBy* não serão inferidas. Dentro deste cenário não são inferidas as regras *usedOneStepArtifactIntroduction* e *usedOneStep-ProcessIntroduction*, que são sub-propriedades de *used* nem tampouco as regras *wasGeneratedByOneStepProcessIntroduction* e *wasGeneratedByOneStepProcessIntroduction* que são sub-propriedades de *wasGeneratedBy*, deixando de fornecer o conhecimento ao usuário sobre as dependências causais indiretas responsáveis pela derivação de um artefato em outro e quais foram as dependências causais indiretas responsáveis por um processo ser disparado (*wasTriggeredBy*) por outro.

A Figura 4.20 ilustra as inferências obtidas para o artefato *Artifact_A4_FIRE0* considerando a captura da proveniência do workflow *SimpleMathOperations* com as otimizações de instrumentalização no terceiro nível. Pode ser constatado que apesar de não ter sido instrumentalizada a dependência causal do tipo *wasGeneratedBy* entre este artefato e o processo que o gerou, foi inferido através da propriedade *wasGeneratedByRPP* que este artefato foi gerado pelo processo *Process_ABSOLUTE_FIRE0*. Também foi inferido para este artefato pela propriedade *wasDerivedFromOneStepProcessElimination* que este artefato foi derivado do artefato *Artifact_A3_FIRE0*, sem que nenhuma dependência causal do tipo *wasDerivedFrom* tenha sido instrumentalizada. Neste exemplo também pode ser visto que as inferências em múltiplos passos *used** e *wasGeneratedBy** foram devidamente realizadas. No entanto, a inferência das propriedades *wasGeneratedByOneStepProcessIntroduction* e *wasGeneratedByOneStepArtifactIntroduction* para este artefato, ambas com o processo *Process_ABSOLUTE_FIRE0* como *range*, não foram realizadas, pelos fatos acima explicados. Outra informação inferida que pode ser observada na Figura 4.20 é a de que o artefato *Artifact_A4_FIRE0* pertence à classe *NoUserInput*. Uma vez que a restrição para pertencer a esta classe depende da propriedade *wasGeneratedBy*, e que a dependência causal *wasGeneratedBy* não foi explicitamente informada nesta base de conhecimento, pode ser concluído que foi realizada inferência com base em inferências.

A Figura 4.21 mostra a inferência realizada para o processo *Process_EXP_Fire0*, também considerando a captura da proveniência do workflow *SimpleMathOperations* com as otimizações de instrumentalização no terceiro nível. Pode ser observado que foi inferida a dependência causal *wasTriggeredBy* segundo o modelo OPM através da propriedade *wasTriggeredByOneStep* entre o processo em questão e o processo *Process_ABSOLUTE_FIRE0*, apesar desta dependência causal não ter sido explicitamente informada nesta base de conhecimento. Pode ser constatado também que as inferências em múltiplos passos *used** e *wasTriggeredBy** foram devidamente realizadas. No entanto, a propriedade *usedOneStepProcessIntrodution* e *usedOneStepArtifactIntrodution* ambas relacionadas ao artefato *Artifact_A4_FIRE0* não foram inferidas, pelos motivos anteriormente explicados.

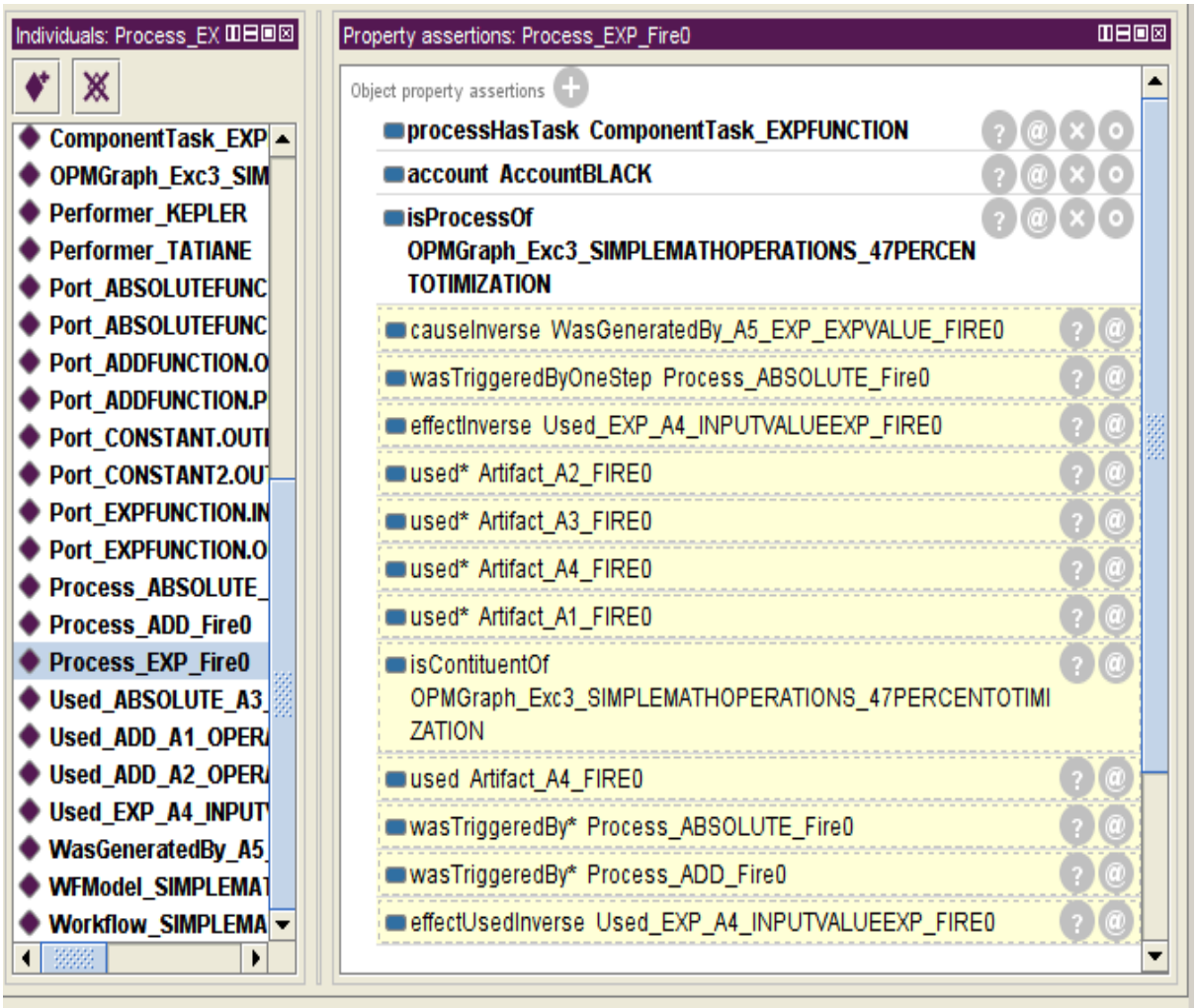


Figura 4.21. Inferências para o processo *Process*_EXP_FIRE0 considerando o workflow instrumentalizado no terceiro nível.

Outra opção de otimização pode ser considerada utilizando apenas instâncias do serviço Web do SciProvMiner com os métodos *wasDerivedFrom* e *wasTriggeredBy*. Nesta opção as instâncias com o método *wasTriggeredBy* capturam as tarefas e os *performers* das tarefas e as instâncias com o método *wasDerivedFrom* capturam as portas das atividades e os fluxos de dados, garantindo assim que a captura da proveniência prospectiva do workflow seja realizada plenamente. Uma instrumentalização utilizando esta opção só é possível em workflows onde para todos os seus processos, as suas saídas dependem de todas as suas entradas, como é o caso do *SimpleMathOperations*. A Figura 4.22 ilustra esta instrumentalização, que elimina a necessidade de sete instâncias de instrumentalização em relação ao workflow original, o que implica em aproximadamente 41% de otimização.

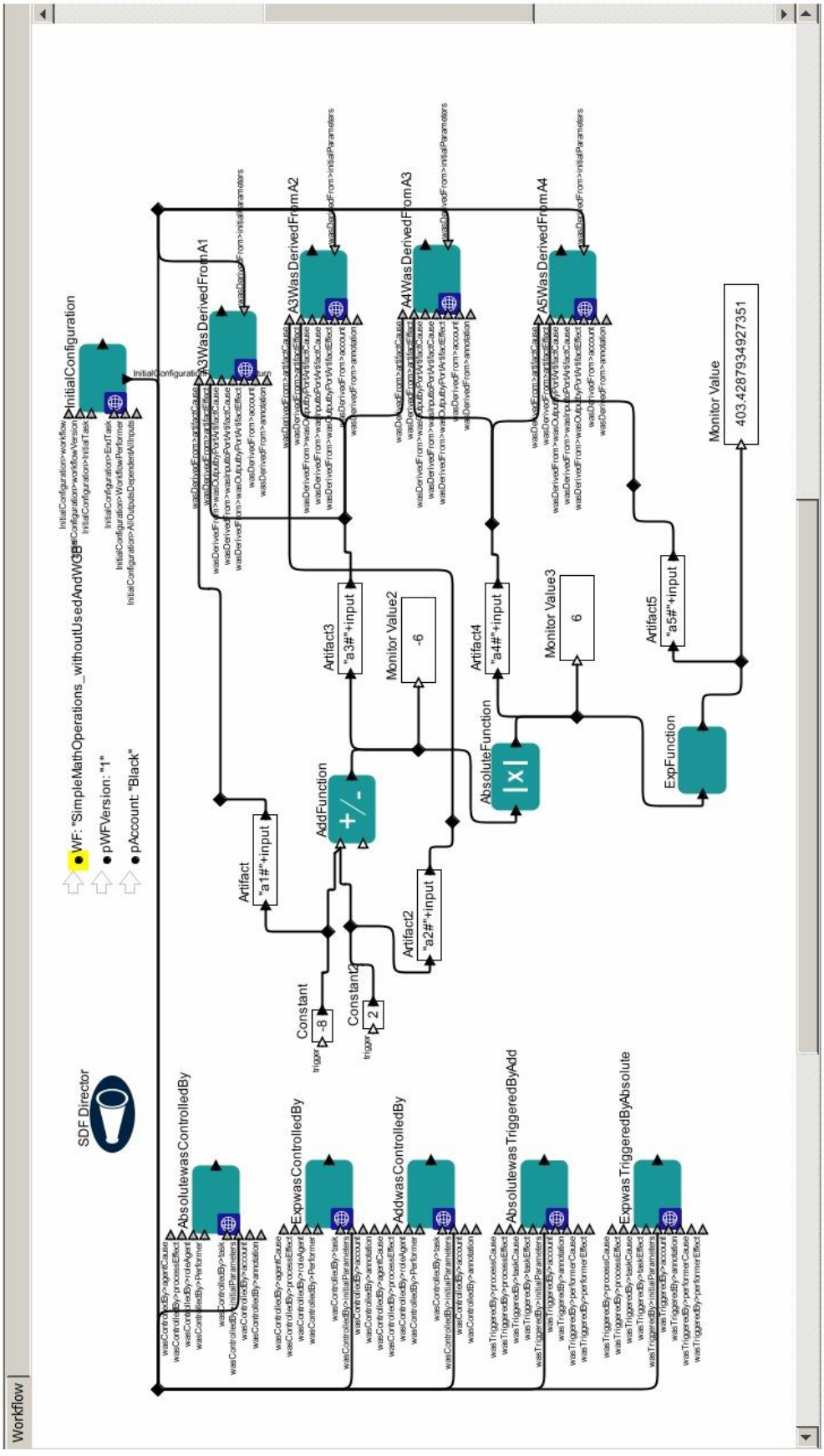


Figura 4.22. Instrumentalização do workflow *SimpleMathOperations* considerando apenas instâncias de instrumentalização com os métodos *wasDerivedFrom* e *wasTriggeredBy*.

É importante destacar que, apesar desta opção não prover o melhor nível de otimização, como o SciProvMiner utiliza a ontologia OPMO-e onde as regras de completude Introdução de Artefato e Introdução de Processo são implementadas, mesmo com a otimização realizada, todas as informações obtidas através de inferências no workflow sem otimização de instrumentalização, são obtidas nesta versão otimizada.

A Figura 4.23 mostra as inferências obtidas para o artefato *Artifact_A4_FIRE0* considerando os dados de proveniência capturados do workflow *SimpleMathOperations* com a última opção de otimização na instrumentalização apresentada.

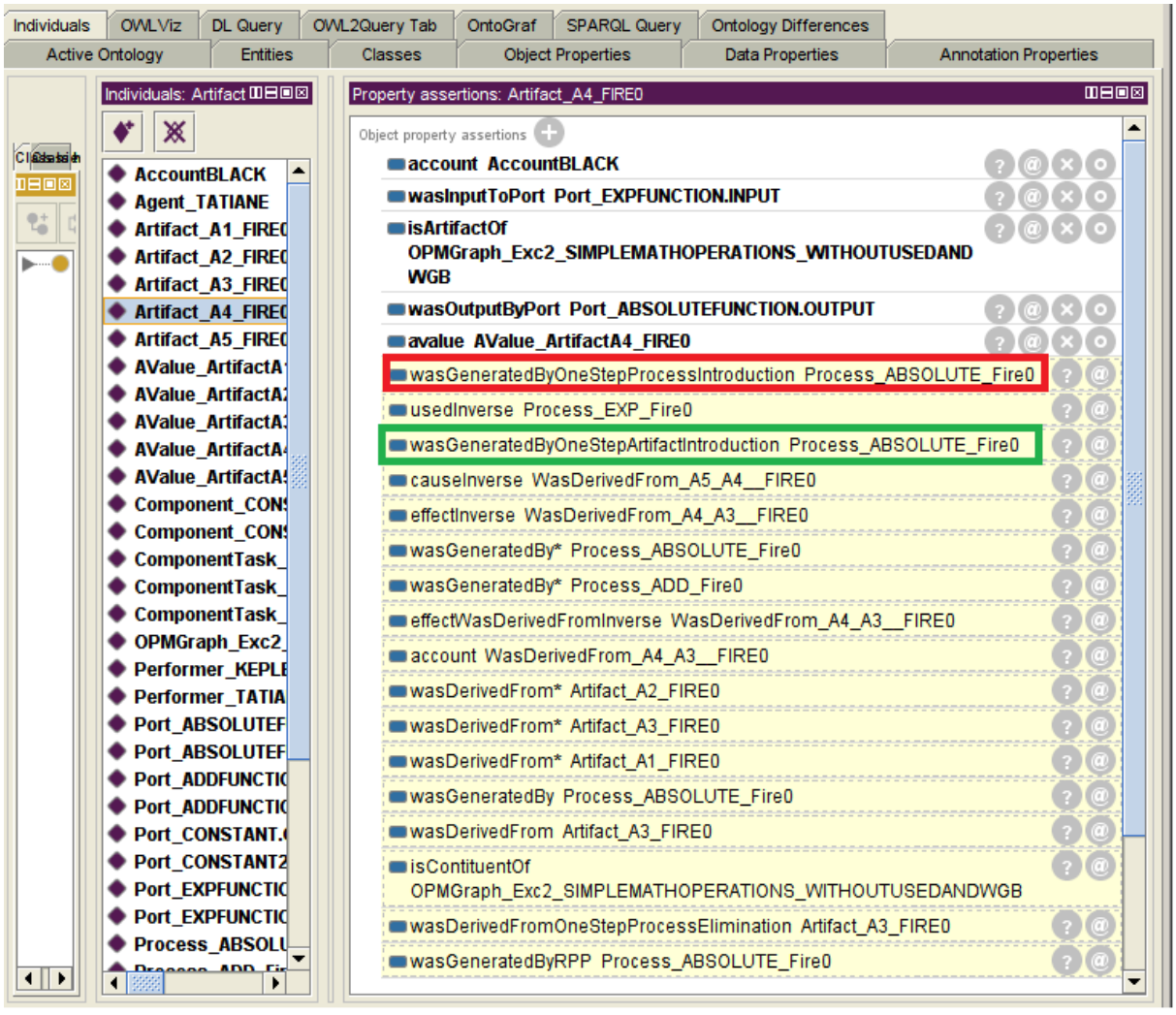


Figura 4.23. Inferências para o artefato *Artifact*_A4_FIRE0 considerando o workflow instrumentalizado apenas com as dependências causais *wasTriggeredBy* e *wasDerivedFrom*.

Considerando a figura 4.23 pode ser notado que todas as inferências obtidas para o artefato *Artifact*_A4_FIRE0 do workflow instrumentalizado no terceiro nível (Figura 4.21) também são obtidas para este artefato no workflow instrumentalizado apenas com as dependências

causais *wasTriggeredBy* e *wasDerivedFrom*, sendo que neste workflow não foram instrumentalizadas a dependência causais *wasGeneratedBy* e *used* segundo o modelo OPM. Nesta representação da proveniência do workflow também foram inferidas as propriedades *wasGeneratedByOneStepProcessIntrodution* e *wasGeneratedByOneStepArtifactIntrodution*, que estão destacadas na Figura 4.23, tendo como *range* o processo *Process*_ABSOLUTE_FIRE0.

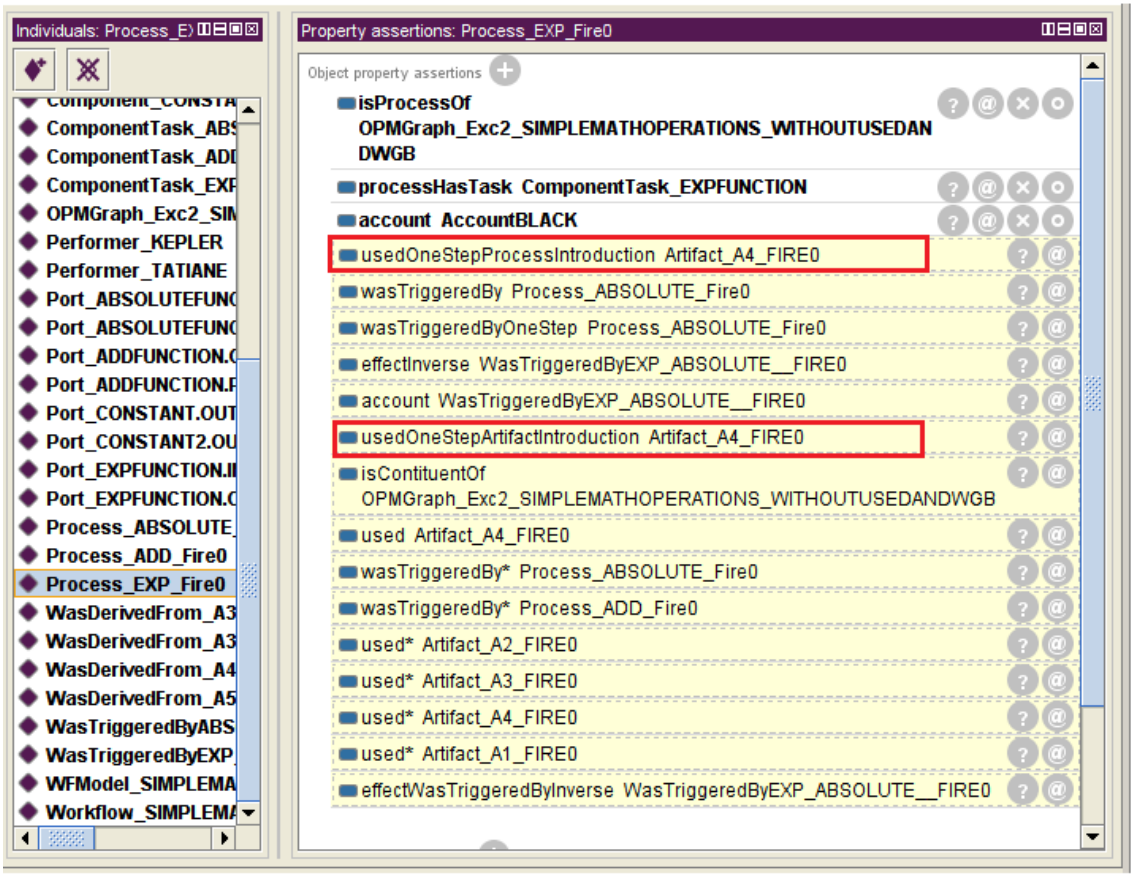


Figura 4.24. Inferências para o processo *Process*_EXP_Fire0 considerando o workflow instrumentalizado apenas com as dependências causais *wasTriggeredBy* e *wasDerivedFrom*.

Pela Figura 4.24 também pode ser visto que todas as inferências obtidas para o processo *Process_EXP_Fire0* do workflow instrumentalizado no terceiro nível (Figura 4.22) também foram obtidas nesta base de conhecimento na qual foram informadas apenas as dependências causais *wasTriggeredBy* e *wasDerivedFrom*. Além disso, também foram inferidas para o processo *Process_EXP_Fire0* as propriedades *usedOneStepProcessIntrodution* e *usedOneStepArtifactIntrodution*que estão em destaque na Figura 4.24, ambas relacionadas ao artefato *Artifact_A4_FIRE0*, que não haviam sido inferidas na base de conhecimento gerada pelo workflow *SimpleMathOperations* instrumentalizado no terceiro nível. Com isso pode-se

concluir que todas as inferências do workflow *SimpleMathOperations* plenamente instrumentalizados, no que diz respeito às dependências causais segundo o modelo OPM, foram inferidas nesta base de conhecimento com menos informações explicitamente informadas.

Esta validação permitiu a utilização das funcionalidades disponíveis no SciProvMiner tais como a captura da proveniência prospectiva e retrospectiva do workflow realizada através da instrumentalização do workflow por serviços Web, a geração do grafo de proveniência retrospectiva, segundo o modelo OPM, a possibilidade de se fazer a consulta aos dados armazenados no banco de dados relacional do SciProvMiner através da linguagem SQL, a geração do arquivo OWL baseado na ontologia OPMO-e com os indivíduos da proveniência prospectiva e retrospectiva capturados a partir do workflow em questão e as consultas ontológicas que podem ser feitas utilizando a linguagem SPARQL na interface do SciProvMiner ou abrindo o arquivo OWL gerado na ferramenta Protégé.

Além disso, ficou evidente o ganho de conhecimento do cientista que utilizar os recursos web semânticos disponíveis no SciProvMiner, uma vez que novas informações de proveniência que não foram explicitamente informadas pelo usuário, são inferidas a partir da utilização de máquinas de inferência na ontologia OPMO-e populada com os indivíduos da proveniência prospectiva e retrospectiva, que foram instrumentalizadas pelo cientista no workflow original para serem capturadas.

Também ficou evidencializado a possibilidade de otimizações na instrumentalização do workflow, que pode ser realizada em diversos níveis, sendo que a base de conhecimento do SciProvMiner irá maximizar as informações de proveniência se as consultas forem feitas na Camada de Consulta aos Dados de Proveniência da arquitetura do SciProvMiner, onde a máquina de inferência Pellet é aplicada sobre a ontologia OPMO-e, e as inferências são realizadas. No entanto, se as consultas forem feitas na Camada de Consulta ao Banco de Dados Relacional da arquitetura do SciProvMiner, apenas as informações fornecidas explicitamente pelos usuários estarão disponíveis para serem consultadas. Esta validação apontou ainda que, apesar de não ser a opção com maior grau de otimização, se o workflow for instrumentalizado apenas com as dependências causais *wasDerivedFrom* e *wasTriggeredBy* ele será capaz de fornecer ao usuário todas as informações relacionadas às dependências causais não informadas explicitamente, se processadas as consultas na camada do SciProvMiner que utiliza recursos web semânticos e máquinas de inferência. Porém se o usuário optar por instrumentalizar o workflow com o maior grau de otimização algumas

informações deixarão de ser inferidas. Portanto caberá ao cientista decidir qual será o grau de otimização que será aplicado na instrumentalização do workflow, avaliando a relação custo benefício que se pretende obter e as características do workflow que está sendo instrumentalizado. No entanto, é importante salientar que as informações das *roles* das dependências causais que não foram explicitamente declaradas não constarão na base de dados relacional, e nem serão inferidas pela base de conhecimento Web semântico.

4.2. WORKFLOW LOAD do PC3

Na segunda prova de conceito, o SciProvMiner foi utilizado para captura de proveniência de um workflow especificado no *Third Provenance Challenge* (PC3) que foi organizado para avaliar a eficiência do modelo OPM na representação e compartilhamento de proveniência com o objetivo de melhorar a especificação (SIMMHAN et al. 2011). O *Load* foi o workflow desenvolvido pela equipe SDSC participante do PC3 para responder ao desafio, sendo que um dos membros desta equipe, Daniel Crawl, foi o responsável por disponibilizar o workflow para ser utilizado no presente trabalho. O *Load* é baseado no workflow científico de carga do Pan-STARRS (*Panoramic Sky Survey and Rapid Response System*), que armazena arquivos de dados dentro de um banco de dados relacional para o projeto Pan-STARRS. Este workflow foi escolhido pelo evento por ser utilizado em aplicações do mundo real e por ser adequado para flexionar as diferentes características do modelo OPM. As especificações do workflow estão detalhadas em Simmhan et al. (2011). Assim, esta prova de conceito tem como objetivo mostrar a captura de proveniência em um workflow real, com grande carga de dados, apresentando assim as possibilidades de otimização das instrumentalizações e a captura de informações estratégicas para os cientistas, a partir da utilização do SciProvMiner. Além disso, tem-se o fato dele ter sido o workflow escolhido para o PC3, e ter por característica capturar informações de proveniência dos mais variados tipos, possuir subtarefas e laço de repetição e, portanto, ser adequado para testar a capacidade do SciProvMiner em lidar com estes requisitos.

Portanto, o workflow *Load* construído por Daniel Crawl e seus colaboradores, foi desenvolvido no Kepler e está apresentado na Figura 4.25.

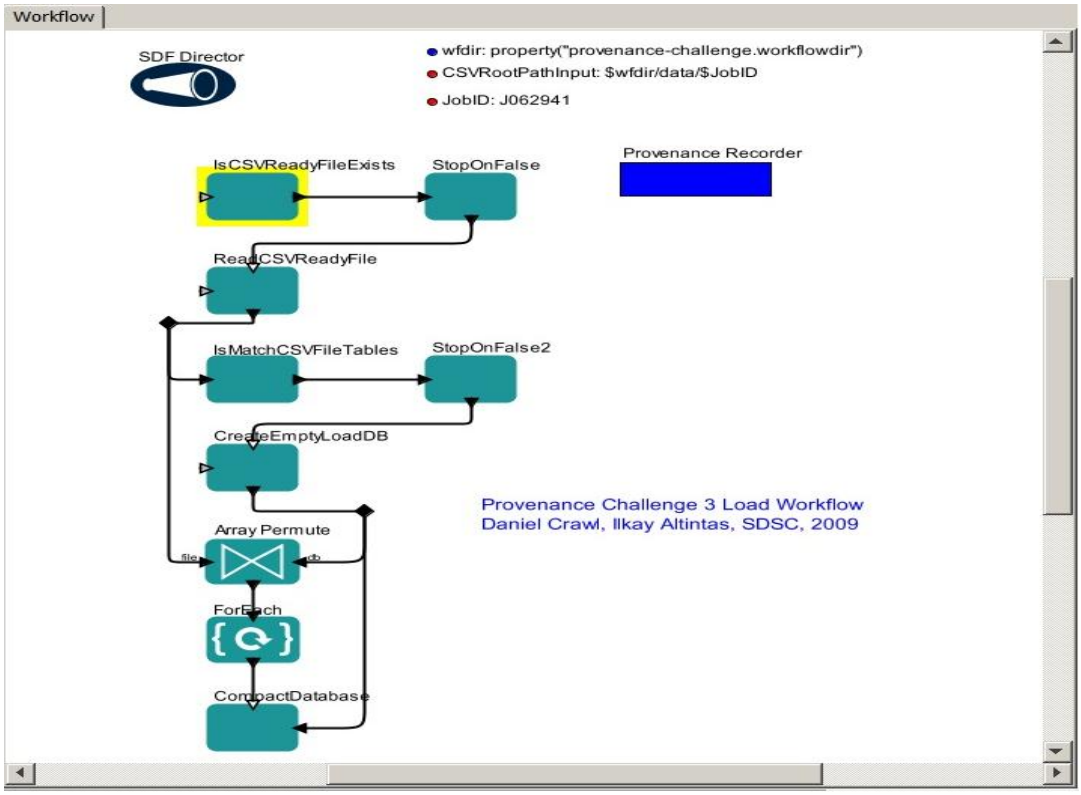


Figura 4.25. Workflow *Load*desenvolvido por Daniel Crawl e Ikay Altintas.

As tarefas do workflow *Load* são as seguintes:

- *IsCSVReadyFileExists*: recebe o parâmetro *CSVRootPathInput* e checa a existência do diretório root e do arquivo de manifesto csv_read. csv.
- *ReadCSVReadyFile*: lê o conteúdo do arquivo de manifesto csv_read.csv e cria um *CSVFileEntry* para esperar o metadata de cada arquivo CSV no manifesto. Esta atividade retorna uma lista de *CSVFileEntry*. Cada *CSVFileEntry* contém o *FilePath* do arquivo CSV a ser carregado, o *HeaderPath* para o arquivo de cabeçalho com a lista da coluna de dados, o *RowCount* do número de linhas no arquivo, o nome da tabela alvo (*TargetTable*) no banco de dados e o *checksum* MD5 para o arquivo. A coluna de nomes *ColumnsNames* não é populada por esta atividade.
- IsMatchCSVFileTables: recebe a lista de *CSVFileEntry* e verifica se todas as tabelas a serem carregadas tem o arquivo de dados CSV correspondente listado no manifesto.
- *CreateEmptyLoadDB*: recebe o parâmetro *JobID*, e cria um banco de dados '*Load*' vazio com uma lista estática de tabelas para carregar o arquivo de Batch CSV dentro. Retorna um *DatabaseEntry* com o *DBName*, identificador *DBGrid* e a string de conexão (*ConnectionString*) da recém-criada instância de banco de dados.

- *ArrayPermute*: concatena os parâmetros recebidos em um array e passa esse array para a tarefa seguinte.
- *ForEach*: esta tarefa executa todas as subtarefas que se encontram dentro dela para cada item do array recebido como parâmetro e retorna um *DatabaseEntry* com todas as tabelas carregadas e validadas.
- *CompactDatabase*: compacta o banco de dados depois de todas as operações de escrita serem completadas.

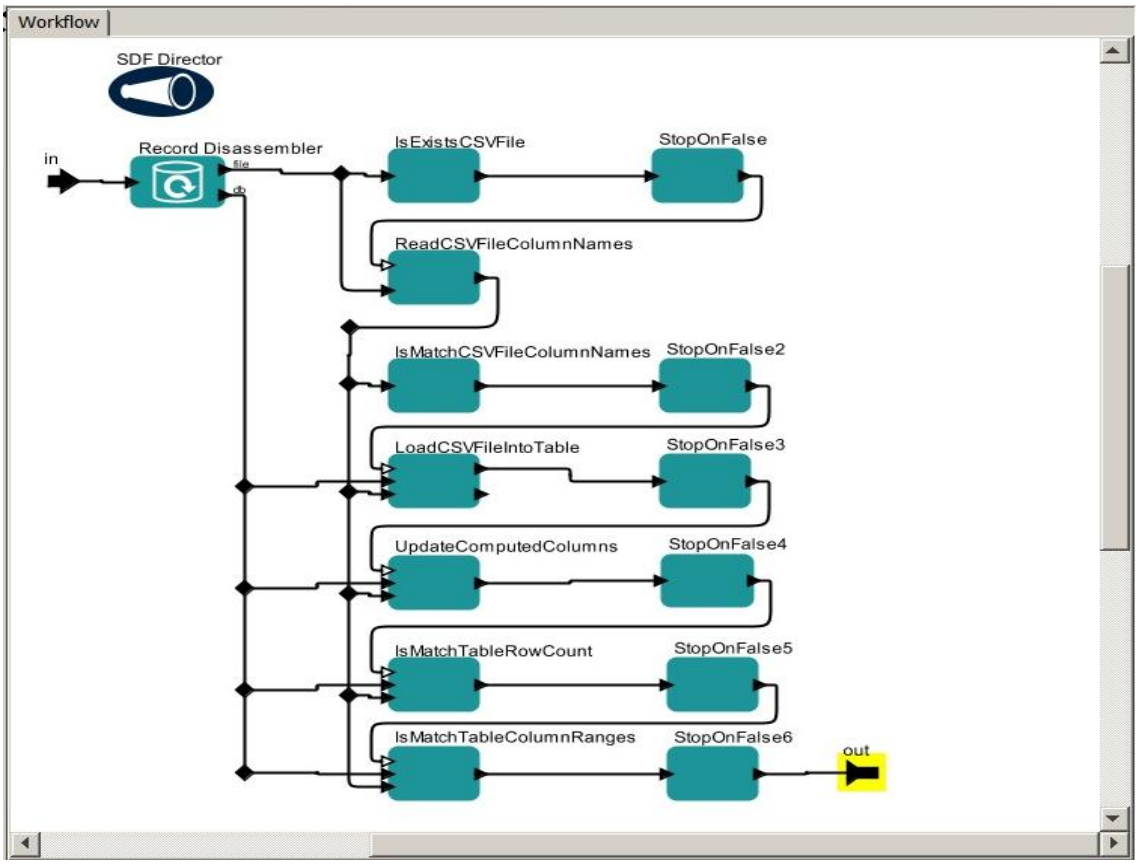


Figura 4.26. Subtarefas da tarefa *ForEach* do Workflow *Load*desenvolvido por Daniel Crawl e Ikay Altintas.

A tarefa *ForEach* contém dentro dela as seguintes subtarefas(Figura 4.26), que ela executa para cada arquivo de entrada:

- *Record Disassembler*: é um componente do Kepler que desmembra os registros de entrada em portas de saída, de acordo com as portas de saída especificadas pelo usuário.
- *IsExistsCSVFile* : verifica a existência do arquivo de dados CSV e arquivo *Header* listados no manifesto.

- *ReadCSVFileColumnNames*: lê a lista dos nomes das colunas presentes no arquivo de dados CSV a partir do arquivo *Header* e retorna a entrada *CSVFileEntry* atualizado com o campo ColumnsNames populado a partir do arquivo *Header*.
- *IsMatchCSVFileColumnNames*: recebe um *CSVFileEntry* lido a partir do arquivo manifesto com a coluna de nomes populada e verifica se todas as colunas esperadas para a tabela alvo estão presentes no arquivo de dados CSV correspondente.
- *LoadCSVFileIntoTable*: recebe um *DatabaseEntry* com a tabela alvo para se carregar o arquivo de dados CSV e um *CSVFileEntry* para ser armazenado dentro da tabela alvo correspondente no banco de dados e carrega o arquivo de dados CSV dentro da tabela correspondente.
- *UpdateComputedColumns*: recebe os mesmos parâmetros citados na tarefa anterior e atualiza as colunas computadas na tabela alvo que foram carregadas. Retorna verdadeiro se as colunas derivadas foram atualizadas com sucesso a partir das colunas existentes.
- *IsMatchTableRowCount*: verifica se o número de linhas carregadas dentro da tabela casa com o número esperado de linhas no arquivo de dados CSV.
- *IsMatchTableColumnRanges*: checa se os dados carregados dentro das colunas das tabelas batem com os tipos de valores esperados para a coluna.

O workflow *Load* foi então instrumentalizado para ter os dados de proveniência coletados pelo SciProvMiner. Dado que o workflow possui bastante tarefas, e que várias destas tarefas recebem mais de um parâmetro de entrada, foi escolhida a instrumentalização com o maior nível de otimização possível, apesar de se saber que haveria alguma perda de conhecimento inferido. A Figura 4.27 apresenta o workflow *Load* instrumentalizado e a Figura 4.28 apresenta as subtarefas da tarefa *ForEach* instrumentalizadas.

Foram necessárias para a instrumentalização deste workflow 39 instâncias de instrumentalização, sendo 36 instâncias de serviço Web do SciProvMiner com o método *used*, uma instância com o método *initialConfiguration* e duas instâncias com o método *wasGeneratedBy*. Neste cenário já fica caracterizada a contribuição em relação à otimização da instrumentalização do workflow, pois se não fosse essa otimização, para este workflow, seriam necessárias por volta de mais 21 instâncias de *wasGeneratedBy*. Como neste workflow todas as saídas dos processos dependem de suas entradas, seriam necessárias, no mínimo, mais 36 instâncias de *wasDerivedFrom* e outras 22 instâncias de *wasTriggeredBy*.

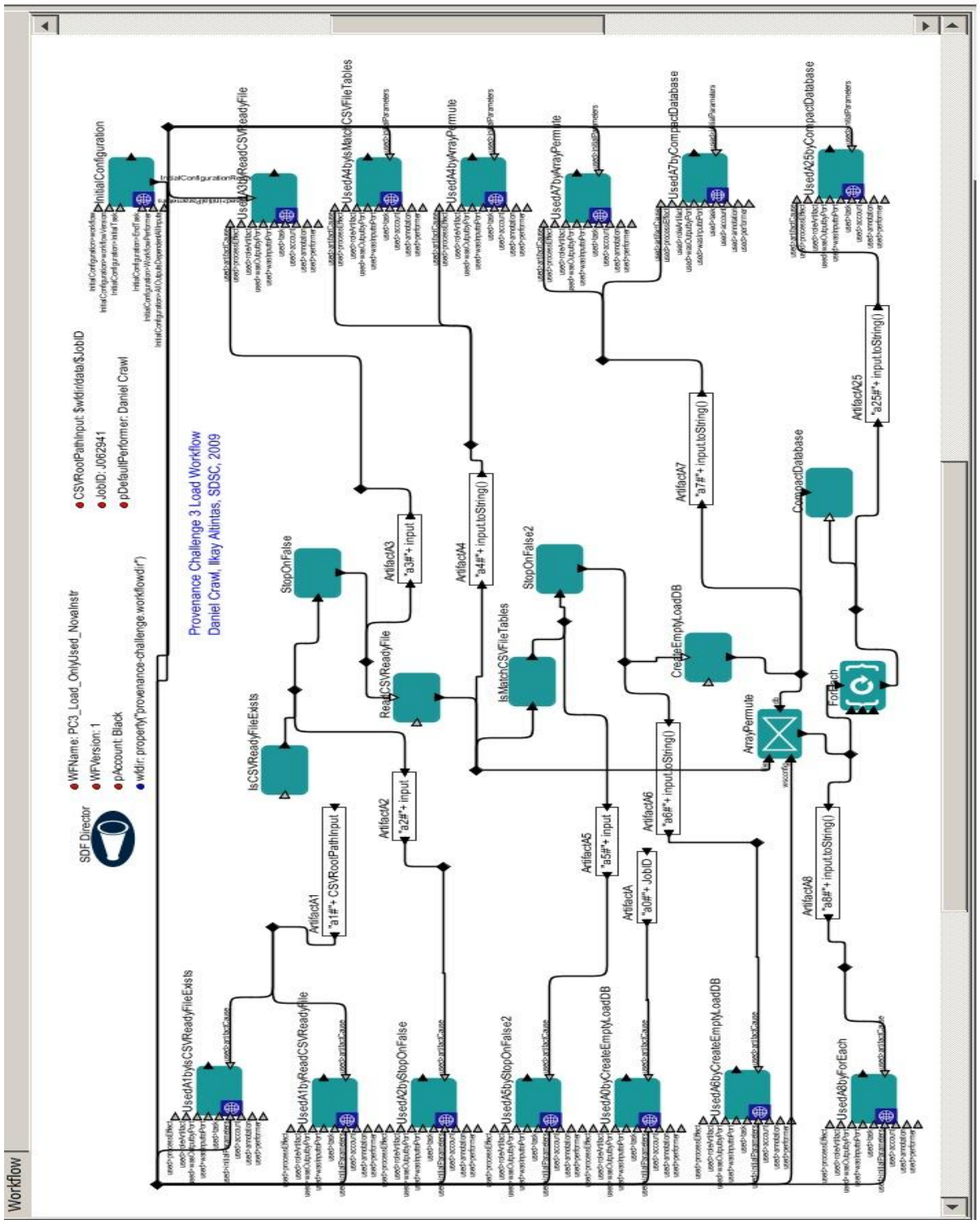


Figura 4.27. Workflow *Load* Instrumentalizado.

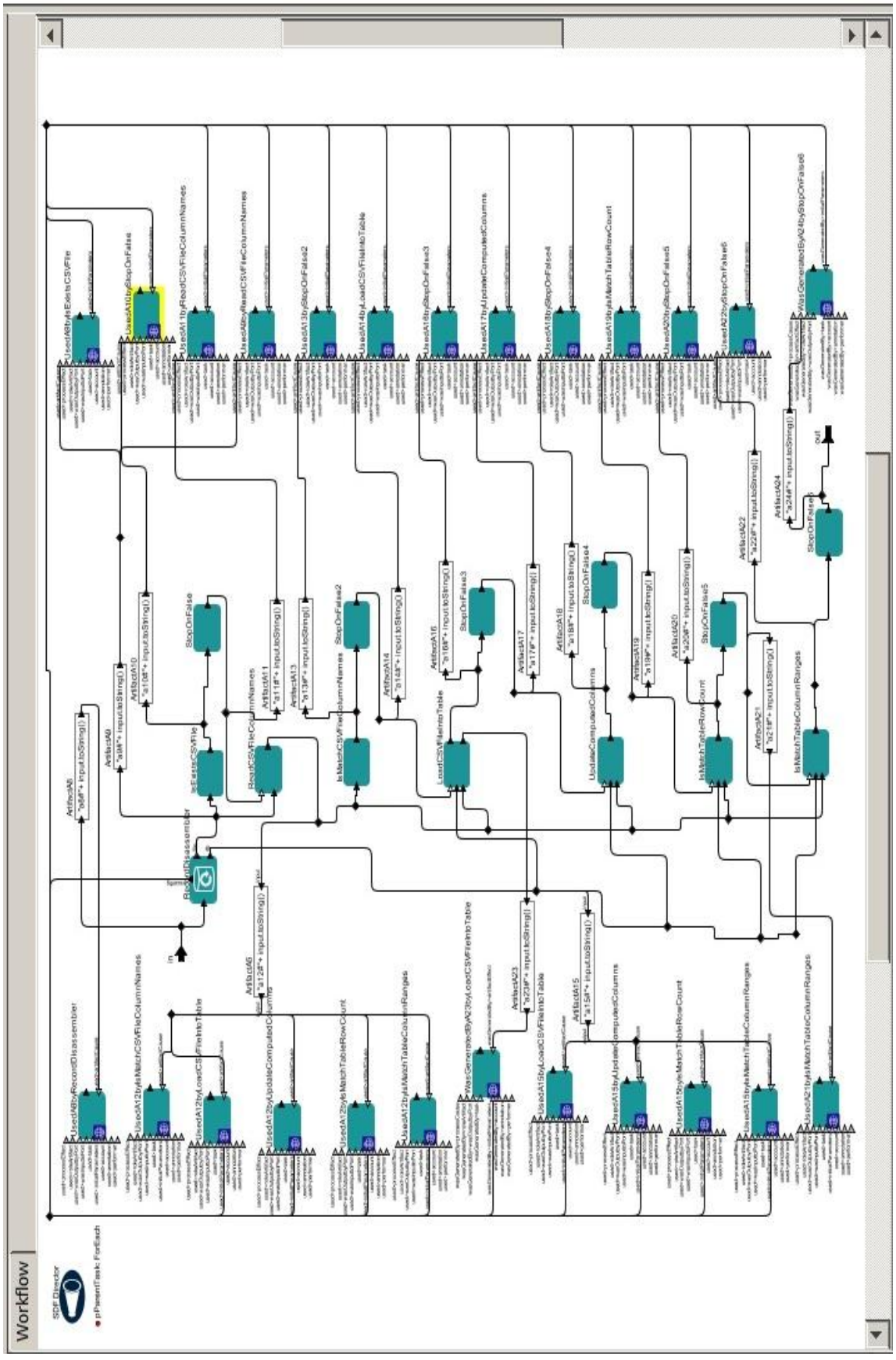


Figura 4.28. Instrumentalização das Subtarefas da tarefa *ForEach* do workflow *Load* Instrumentalizado.

O workflow *Load* está parametrizado para armazenar no banco de dados, as informações que estão no diretório especificado pelo parâmetro *CSVRootPathInput* dentro da pasta J062941 especificado pelo parâmetro *JobID*. A Figura 4.29 exibe os arquivos que se encontram dentro da pasta J062941 que serão utilizados pelo workflow.

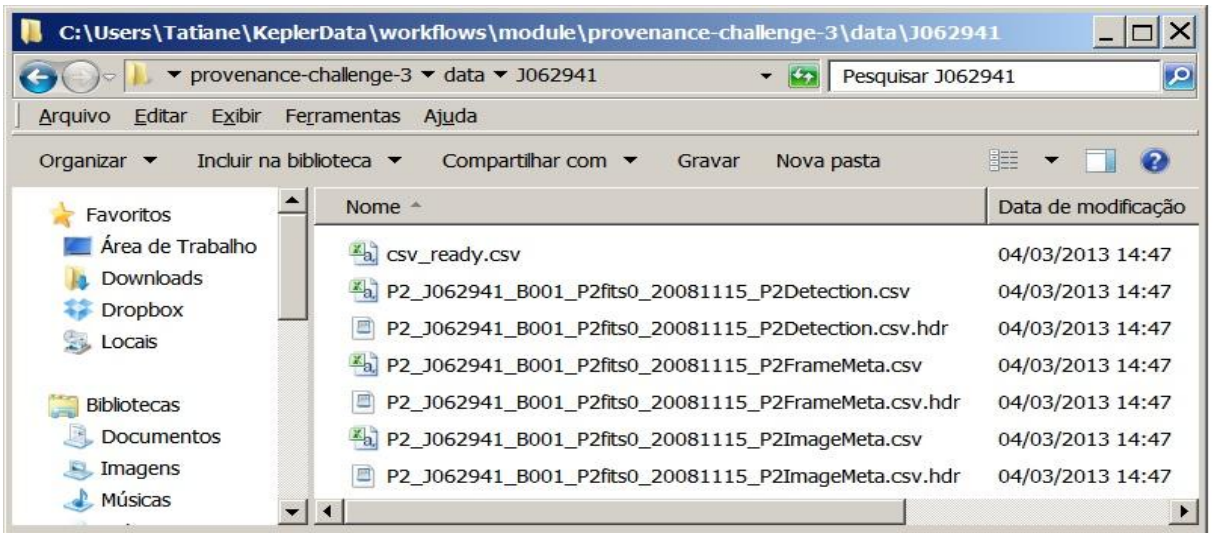


Figura 4.29. Diretório com os arquivos que serão utilizados pelo workflow *Load*.

Ao executar o workflow *Load*, o SciProvMiner suportou bem a carga de chamadas ao serviço Web para armazenamento da proveniência, armazenou corretamente as informações dos artefatos no banco de dados e as relações de tarefa-subtarefa entre *ForEach* e suas subtarefas. Os dados de proveniência de cada iteração do laço de repetição foram corretamente armazenados no banco de dados, mostrando que o SciProvMiner é robusto o suficiente para processar a captura de proveniência de workflows no nível do *Load*. Por questões de espaço, e por já ter sido mostrado no workflow *SimpleMathOperations*, não serão apresentadas as interfaces do SciProvMiner para consulta à camada de banco de dados SQL para este workflow, por não haver diferenças consideráveis. Depois de ter sido solicitada a geração do arquivo OWL no sistema SciProvMiner e ter sido gerada a ontologia OPMO-e com os indivíduos deste workflow, a ontologia foi aberta no Protégé e as inferências foram processadas na ontologia pela máquina de inferência Pellet. A base de conhecimento semântica foi populada com 400 indivíduos, classificados dentre as opções de classes da ontologia OPMO estendida. Alguns dos resultados obtidos são analisados a seguir.

A Figura 4.30 mostra os diferentes valores recebidos pelo artefato ArtifactA12, em cada uma das execuções do laço de repetição chamadas de "*Fire*". A Figura 4.31 exibe todas as subtarefas da tarefa *ComponentTask_FOREACH* inferidas na ontologia.

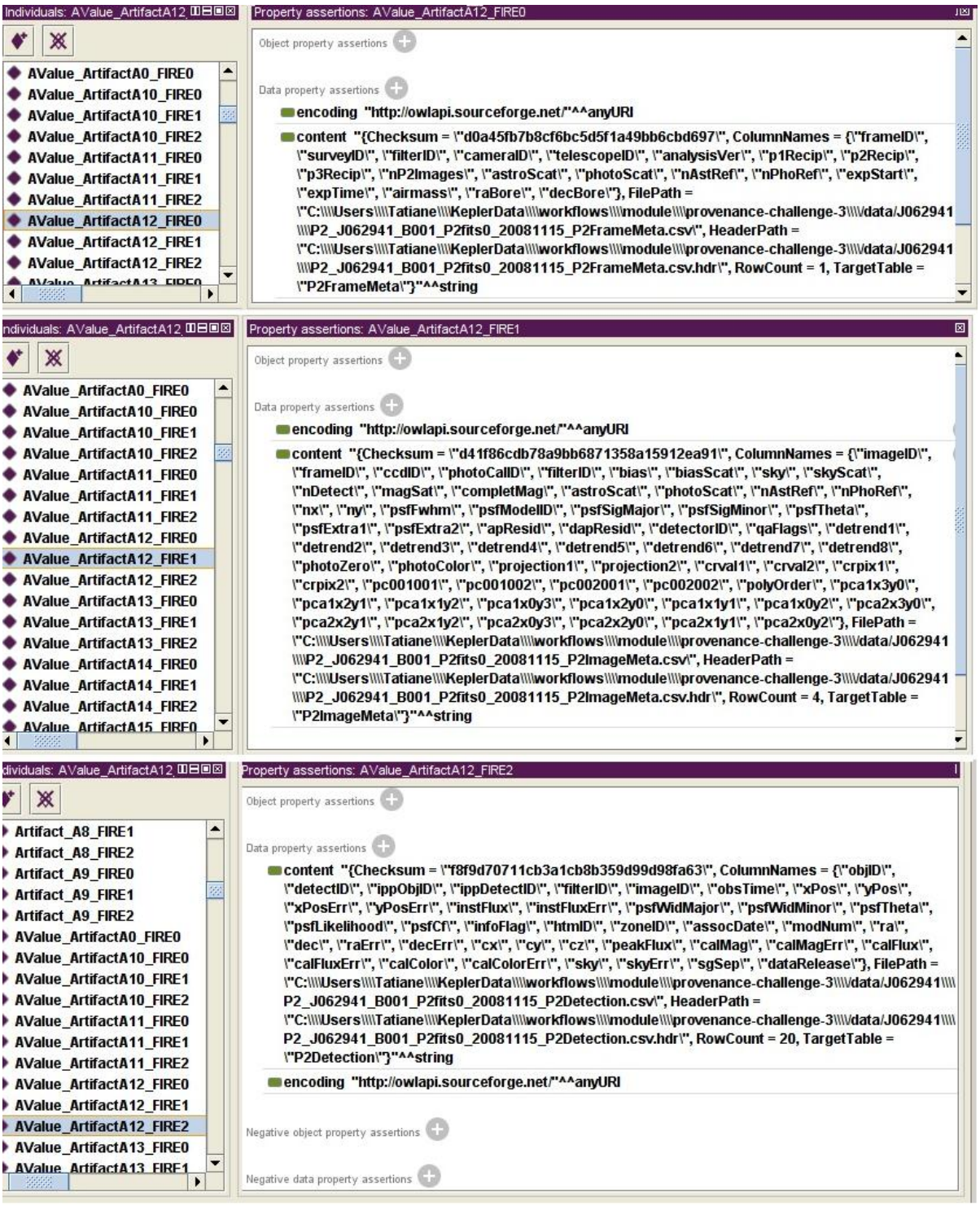


Figura 4.30. Valores para o artefato Artifact12 em cada uma das execuções do laço de repetição do workflow *Load*.

A Figura 4.32 mostra as inferências obtidas para o artefato *Artifact_A4_Fire0*, onde existem duas inferências relacionadas à propriedade *wasDerivedFromSepProcessElimination* que relacionam este artefato aos artefatos *Artifact*_A3_Fire0 e *Artifact*_A1_Fire0 respectivamente, através das dependências causais do tipo *wasDerivedFrom*, sendo que não

foram explicitamente informadas pelo usuário na captura da proveniência. Também se encontra inferido para este artefato a propriedade *wasGeneratedByRPP* tendo como *range* o processo *Process*_READCSVREADFILE_ Fire0, inferindo uma dependência causal do tipo *wasGeneratedBy* segundo o modelo OPM entre o artefato e processo relacionado. A Figura 4.32 também está mostrando as inferências em múltiplos passos *wasGeneratedBy** e *wasDerivedFrom** devidamente realizadas.

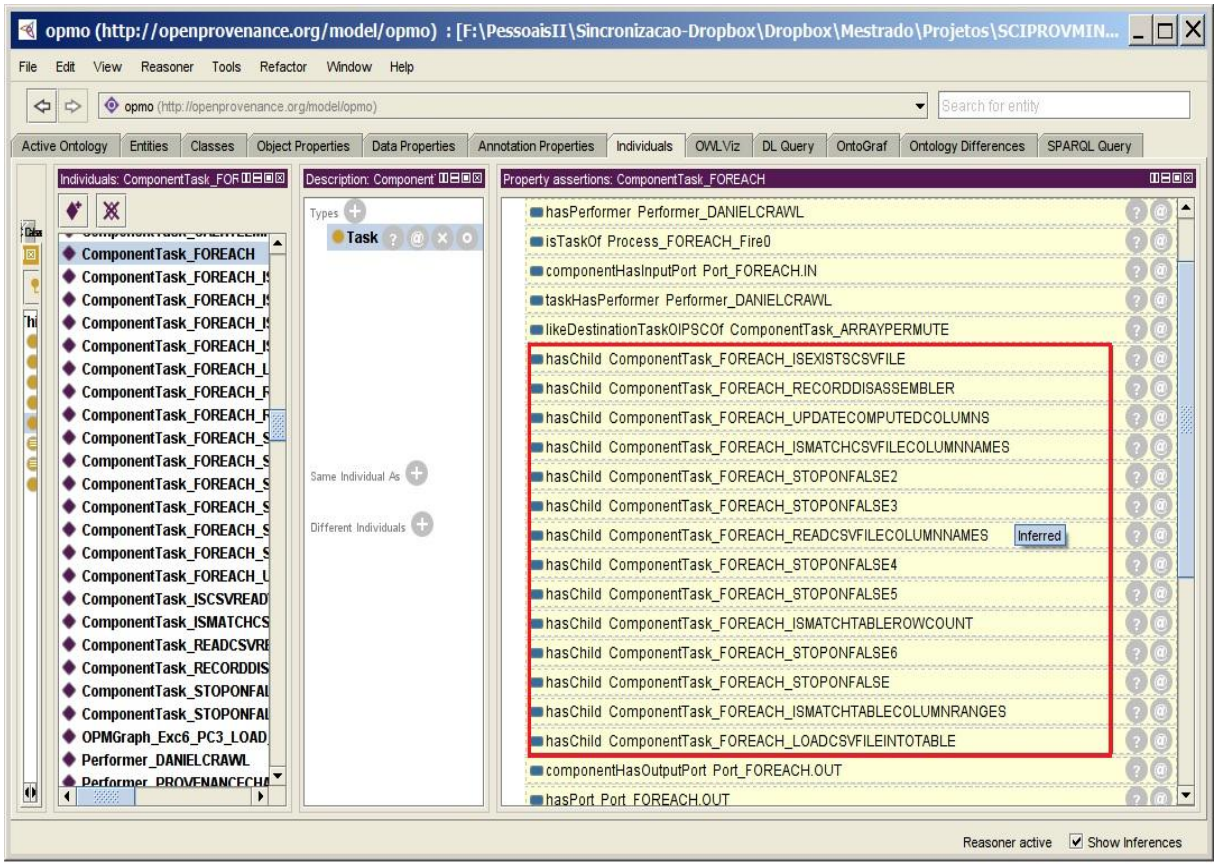


Figura 4.31. Subtarefas de ComponentTask_*ForEach* do workflow *Load* inferidas na ontologia.

Como a escolha de otimização da instrumentalização do workflow foi de terceiro nível, onde são instrumentalizadas predominantemente as dependências causais do tipo *Used*, e não são instrumentalizadas dependências causais do tipo *wasDerivedFrom* e *wasTriggeredBy*, a base de conhecimento não é capaz de inferir para este artefato as propriedades *wasGeneratedByOneStepProcessIntroduction* e *wasGeneratedByOneStepProcessIntroduction* que deveria ter como *range* o processo *Process_READCSVREADYFILE_ Fire0*.

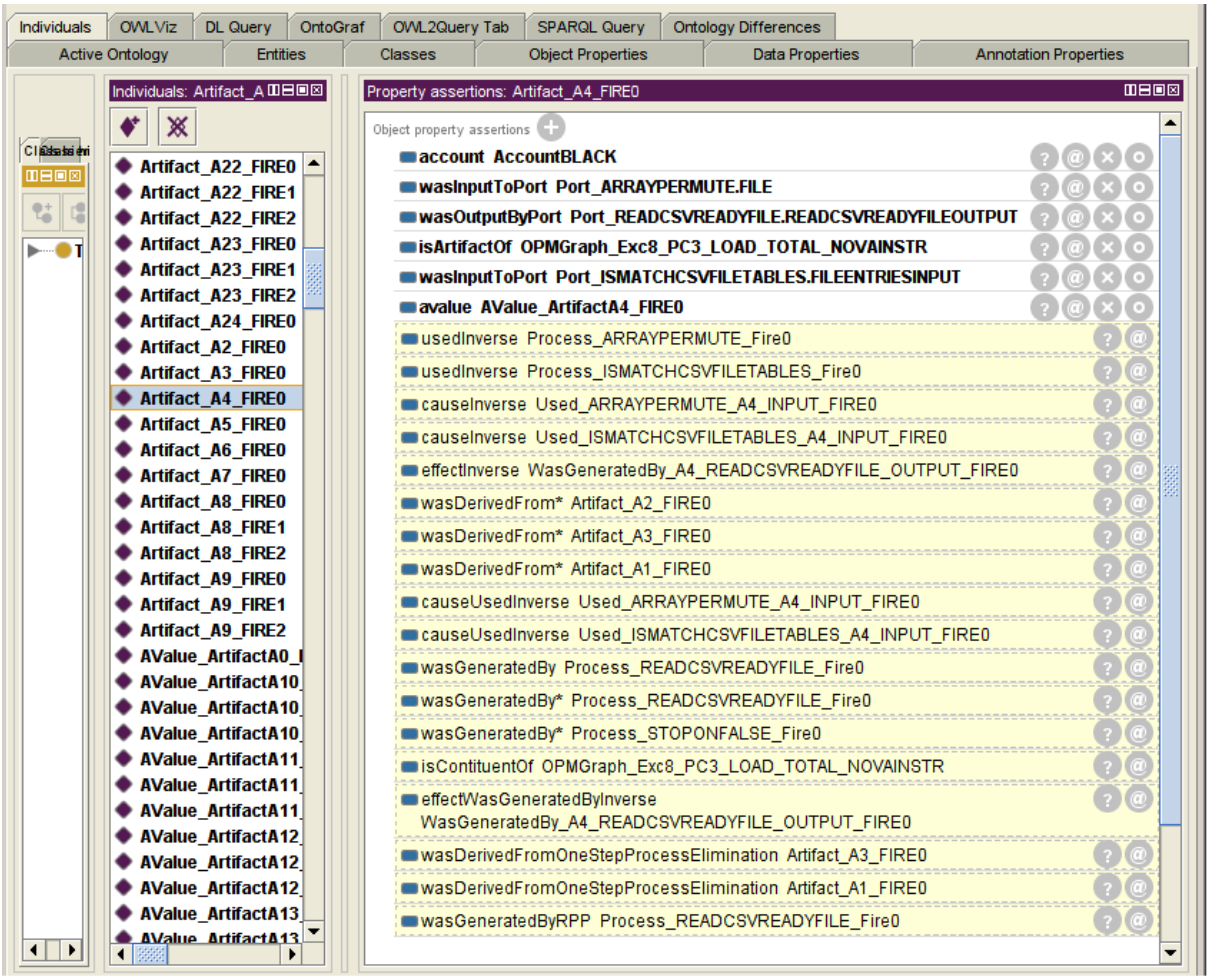


Figura 4.32. Inferências para o artefato *Artifact*_A4_Fire0 do workflow *Load*.

A Figura 4.33 apresenta as inferências para o processo *Process_READCSVREADYFILE_Fire0*, onde pode ser visto que foi inferida a propriedade *wasTriggeredByOneStep* entre este processo e o *Process_STOPONFALSE_Fire0*, mesmo não tendo sido instrumentalizada explicitamente nenhuma dependência causal *wasTriggeredBy*. Pode ser notado que as inferências em múltiplos passos *used** e *wasTriggeredBy** também foram inferidas corretamente. As propriedades *usedOneStepProcessIntrodution* e *usedOneStepArtifactIntroduction* não foram inferidas por motivos já explicados anteriormente.

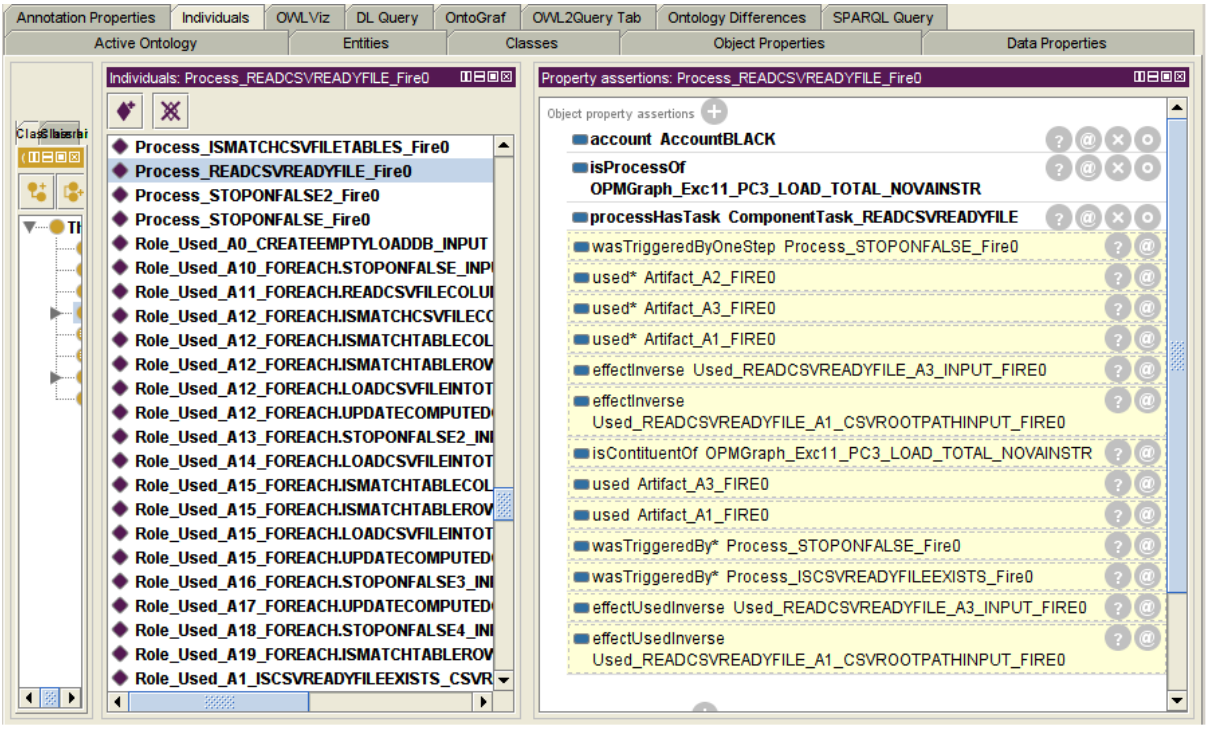


Figura 4.33. Inferências para o processo *process_READCSVREADYFILE_A4_Fire0* do workflow *Load*.

Apesar da camada de banco de dados e do algoritmo de captura de proveniência do SciProvMiner estarem preparados para suportarem workflows com laços de repetição, as propriedades construídas na ontologia OPMO-e neste trabalho para a implementação das regras de completude e inferência definidas no modelo OPM e das otimizações na instrumentalização do workflow não foram construídas levando-se em consideração os *Fires* de execução que ocorrem em laços de repetição. Por isso ocorreram erros nas inferências das dependências causais contidas no laço de repetição. A Figura 4.34 exemplifica o erro ocorrido, onde ao invés de ser inferida para o artefato *Artifact_A10_Fire0* a propriedade wasGeneratedByRPP apenas para o processo de mesmo *fire Process_FOREACH.-ISEXISTSCSVREADFILE_Fire0*, o artefato foi relacionado com este processo em todos os *fires* de execução ocorridos no workflow. Esse erro limita a utilização do SciProvMiner em workflows com laços de repetição. Esta correção deve ser abordada em trabalhos futuros, para que o SciProvMiner possa ab*ranger* workflows com laços de repetição em sua camada de consulta aos grafo de proveniência, onde a ontologia OPMO-e é utilizada como base.

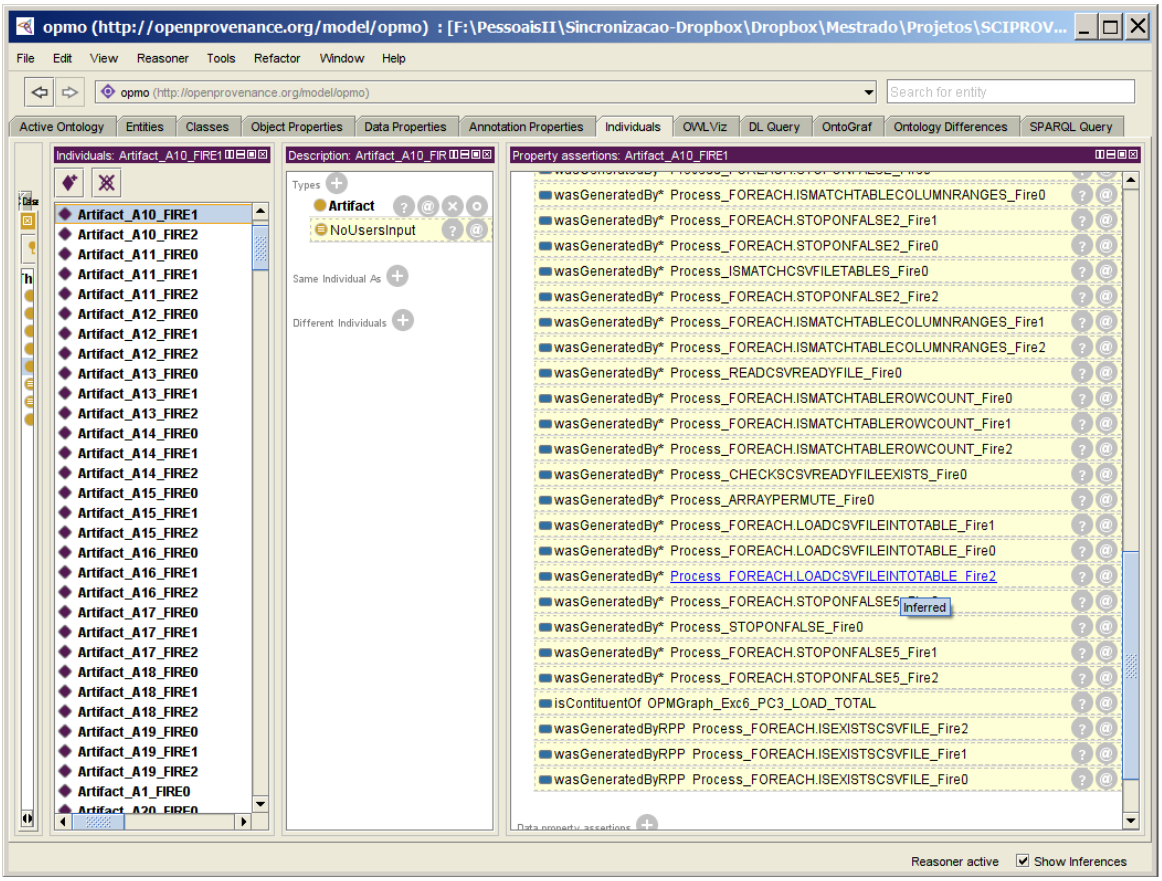


Figura 4.34. Inferências para um artefato contido em um laço de repetição.

A validação relacionada ao workflow *Load* do *Third Provenance Challenge* mostrou que o SciProvMiner é robusto o suficiente para capturar corretamente as informações de proveniência instrumentalizadas em workflows do porte do Load do PC3, que possui laços de repetição, com várias tarefas e subtarefas, persisti-las no banco de dados relacional e construir o documento OWL correlacionado a captura da proveniência de dados deste workflow. Além disso, a possibilidade de otimização do processo de instrumentalização se mostrou bastante importante neste workflow, pois diminuiu substancialmente o trabalho do cientista na instrumentalização do workflow maximizando as informações de proveniência contidas na base de conhecimento do SciProvMiner quando as consultas são feitas na Camada de Consulta aos Dados de Proveniência da arquitetura do SciProvMiner, onde a máquina de inferência Pellet é aplicada sobre a ontologia OPMO-e, e as inferências são realizadas. A validação aplicada a este workflow também mostrou que serão necessários alguns ajustes na ontologia OPMO-e para inferir corretamente as informações em workflows que possuem laços de repetição, pois a implementação das regras de completude e inferência definidas no modelo OPM e das otimizações na instrumentalização do workflow não foram construídas

levando-se em consideração os *Fires* de execução que ocorrem em laços de repetição, e por isso ocorreram erros nas inferências.

4.3. ANÁLISE DA PROVA DE CONCEITO

O SciProvMiner, por capturar a proveniência prospectiva trouxe de ganho a possibilidade de responder a consultas relacionadas à especificação do workflow, tais como quais são as tarefas do workflow, quais tarefas possuem subtarefas, quais são as tarefas sucessoras e antecessoras de uma tarefa no fluxo do workflow, qual é o componente sucessor imediato de um componente, qual é o antecessor imediato de um componente, quais componentes estão conectados, quais tarefas foram executadas pelo workflow, visto que se ocorrer algum erro durante a execução do workflow, alguma tarefa pode deixar de ser realizada, quais tarefas deixaram de ser executadas pelo workflow em caso de algum erro na execução, entre outras. Essas duas últimas perguntas fazem parte das questões do *Third Provenance Challenge*, que não puderam ser respondidas por nenhuma equipe baseadas apenas o modelo OPM (LIM et al. 2011). Além disso, a proveniência prospectiva foi utilizada como base para a formação de regras na ontologia OPMO em conjunto com a proveniência retrospectiva, tais como *wasGeneratedByRPP*, que possibilitou a inferência da dependência causal *wasGeneratedBy* entre um artefato e um processo, mesmo sem ter sido explicitamente informada pelo usuário. Esta interação de informações de proveniência prospectiva e retrospectiva também tornou possível que fosse inferido o "artefato escondido" na regra de completude Introdução de Artefato e também o "processo escondido" na regra de completude Introdução de Processo, ambas definidas na documentação do modelo OPM (MOREAU et al. 2011), sendo que apenas com informação de proveniência retrospectiva não seria possível a descoberta dessas informações implícitas.

A extensão do modelo OPMO trouxe como ganho para o SciProvMiner além das questões relacionadas à proveniência prospectiva, a possibilidade de cobertura de todas as regras de completude e inferência definidas na documentação do modelo OPM, que, como mostrado nas provas de conceito, aumentaram consideravelmente o conhecimento do usuário sobre a proveniência capturada. Pela implementação das propriedades relacionadas à regra Introdução de Processo, foram feitas inferências que informaram ao usuário sobre as dependências causais indiretas responsáveis pela derivação de um artefato em outro. Pela implementação da propriedade relacionada à Eliminação de Processo, é possível fornecer ao usuário a informação de que um artefato foi derivado a partir de outro, e quais são estes artefatos, sem

que tenha sido explicitamente declarada a dependência causal *was derived by*. Pela implementação das propriedades relacionada à regra Introdução de Artefato, foi possível informar ao usuário quais foram as dependências causais indiretas responsáveis por fazer um processo ser desencadeado (*wasTriggeredBy*) por outro. Pela implementação da propriedade relacionada à Eliminação de Artefato, é possível fornecer ao usuário a informação de que um processo foi desencadeado por outro, e quais são estes processos, sem que tenha sido explicitamente declarada a dependência causal *wasTriggeredBy*. A utilização das inferências em múltiplos passos definidas na documentação do modelo OPM (MOREAU et al. 2011) trouxeram o conhecimento para o usuário das causas indiretas que pelas quais um artefato ou um processo se tornaram o que eles são, pois muitas vezes o usuário pode não estar interessado apenas nas causas diretas, mas também nas indiretas (MOREAU et al. 2011).

As inferências proporcionadas pelas regras implementadas na ontologia OPMO-e tornaram possível otimizações na instrumentalização do workflow, podendo produzir um resultado sem perda nas informações fornecidas pelas inferências da ontologia comparadas a versão completamente instrumentalizada do mesmo workflow. Essas otimizações são importantes devido ao fato da tarefa de instrumentalização do SciProvMiner ser manual, e pelo fato de que, quanto mais otimizada e automatizada for a tarefa de instrumentalização, menor será a chance de introdução de erro na captura da proveniência e maior a chance do usuário se interessar pela solução.

5. CONSIDERAÇÕES FINAIS

Prover informação histórica de experimentos científicos, em vistas a tratar o problema de perda de conhecimento do cientista sobre o experimento é um dos desafios de *e-Science* (MATTOSO et al. 2008), que encontra respaldo no documento dos Grandes Desafios da Computação no Brasil definido pela Sociedade Brasileira da Computação para o período de 2006 a 2016 (CARVALHO et al. 2006). A esse desafio dá-se o nome de proveniência de dados em *e-Science*.

Devido ao fato de o tratamento da proveniência de dados constituir um tema de pesquisa em aberto e, no contexto de *e-Science*, existirem questões que se trabalhadas podem fornecer ao cientista informações importantes a respeito do experimento realizado com o potencial de auxiliá-lo a formar uma visão da qualidade, da validade e da atualidade acerca da informação produzida no contexto do experimento científico modelado computacionalmente, o presente trabalho teve como objetivo contribuir para essa área, apresentando a arquitetura SciProvMiner, para coleta e consulta de proveniência utilizando recursos da Web semântica para ampliação do conhecimento gerado e otimização do processo de coleta.

5.1. CONTRIBUIÇÕES

O SciProvMiner é uma extensão da arquitetura SciProv (VALENTE 2011), que propõe um modelo de proveniência de dados e processos cujo propósito consiste em interagir com os sistemas de gerenciamento de workflows científicos utilizados em um ambiente colaborativo com a finalidade de capturar e gerir as informações de linhagem geradas.

O SciProvMiner possui as características apresentadas pelo SciProv e estende esta arquitetura adicionando a ela a capacidade de capturar a proveniência prospectiva e a possibilidade de se utilizar as regras de completude e inferência definidas na documentação do modelo OPM.

As contribuições específicas do SciProvMiner são:

- Desenvolvimento de um modelo para contemplar a proveniência prospectiva e retrospectiva como uma extensão do *Open Provenance Model* (OPM), que em sua forma original modela somente proveniência retrospectiva.

- Especificação e implementação de um coletor de proveniência que utiliza a tecnologia de serviços Web para capturar ambos os tipos de proveniência segundo o modelo acima;
- Desenvolvimento de uma ontologia denominada OPMO-e, que estende a ontologia *Open Provenance Model Ontology* (OPMO) de forma a modelar o conhecimento acerca da proveniência prospectiva além da retrospectiva já contemplada na OPMO. Nesta ontologia foram implementadas propriedades baseadas no conceito de cadeia de propriedades disponível na OWL2, que viabilizaram a implementação de regras de completude e inferência definidas na documentação do modelo OPM. Estas regras aumentam o conhecimento do cientista sobre o experimento realizado por inferir informações que não foram explicitamente fornecidas pelo usuário e tornando possível a otimização do processo de captura de proveniência, e a consequente diminuição do trabalho do cientista para instrumentalizar o workflow. Esta é uma das grandes contribuições do SciProvMiner, pois até então a ontologia OPMO não contava com esta funcionalidade;
- Especificação de um banco de dados relacional onde são armazenadas as informações de proveniência capturadas pelo coletor, que pode ser utilizado para ser consultado a respeito da proveniência explicitamente capturada, além de servir como base para a geração do grafo de proveniência retrospectiva, e de ser a base para a criação do documento OWL baseado na ontologia OPMO-e, proposta neste trabalho, com os indivíduos do workflow para o qual se deseja representar o conhecimento acerca da proveniência.

A captura da proveniência prospectiva, disponibilizada a princípio para ser utilizada apenas para incorporar ao SciProvMiner a capacidade deste em responder a consultas relacionadas à especificação do workflow, trouxe um ganho ainda maior para a arquitetura, pois possibilitou ao SciProvMiner incorporar na ontologia OPMO-e as regras de completude definidas no modelo OPM em Moreau et al. (2011), que, considerando somente a proveniência retrospectiva, não seriam possíveis de serem implementadas. Estas regras se propõem a encontrar componentes do modelo que não foram informados de maneira explícita pelo usuário (artefato ou processo, dependendo da regra) e que, se descobertos, aumentam o conhecimento do cientista acerca do experimento realizado. Outra contribuição foi que, pela utilização de propriedades na ontologia OPMO-e relacionadas à proveniência prospectiva em composição com propriedades relacionadas à proveniência retrospectiva, foi possível a inferência da dependência causal *wasGeneratedBy* segundo o modelo OPM, mesmo que esta

não tenha sido explicitamente informada à base de conhecimento, contribuindo para a otimização do processo de coleta proposta no SciProvMiner.

Além disso, a prova de conceito apresentada no presente trabalho mostra indícios dos possíveis ganhos que podem ser obtidos com a utilização do SciProvMiner, considerando o aumento do conhecimento do cientista com relação à proveniência capturada, através de informações inferidas na base de conhecimento que não foram explicitamente informadas e a otimização do processo de instrumentalização que é importante devido ao fato desta tarefa ser manual, e pelo fato de quanto mais otimizada e automatizada for a tarefa de instrumentalização, menor será a chance de introdução de erros na captura da proveniência e maior a chance do usuário se interessar pela solução.

5.2. LIMITAÇÕES

A prova de conceito relacionada ao workflow *Load* do *Third Provenance Challenge* mostrou que o SciProvMiner possui limitações em se tratando de workflows com laços de repetição, pois apesar da camada de banco de dados e do algoritmo de captura de proveniência do SciProvMiner estarem preparados para suportarem workflows com laços de repetição, as propriedades construídas na ontologia OPMO-e neste trabalho, para a implementação das regras de completude e inferência definidas no modelo OPM, e das otimizações na instrumentalização do workflow, não foram construídas levando-se em consideração os *Fires* de execução que ocorrem em laços de repetição.

Outra limitação na utilização do SciProvMiner se refere a workflow com distintos fluxos alternativos (fluxos bifurcados por atividades de decisão), pois o SciProvMiner não está atualmente preparado para este tipo de situação.

Além disso, o SciProvMiner precisa evoluir no que se refere ao tratamento de grandes volumes de dados. Considerando as informações de proveniência armazenadas no banco de dados, o tratamento atual se mostra adequado, apesar de poder ser ainda aprimorado. No entanto, para que seja possível a utilização das funcionalidades da Web semântica, a carga dos indivíduos na ontologia e a utilização da máquina de inferência em grandes volumes de dados necessitam de melhor tratamento no contexto do SciProvMiner.

5.3. TRABALHOS FUTUROS

Alguns trabalhos futuros permitirão a evolução da arquitetura SciProvMiner. Listamos a seguir os principais:

- Adaptação das regras incorporadas na ontologia OPMO para ab*ranger* workflows com laços de repetição em sua camada de consulta aos grafo de proveniência, onde a ontologia OPMO-e é utilizada como base;
- Preparação da arquitetura do SciProvMiner como um todo para ab*ranger* workflows que trabalham com fluxos alternativos;
- Tornar o processo de instrumentalização do workflow mais automatizado, para facilitar essa atividade que, atualmente, é manual e tornar essa tarefa mais agradável para o usuário;
- Incorporar ao SciProvMiner formas de o usuário consultar as informações da base de conhecimento ontológica de forma mais amigável, em mais alto nível, de tal forma que fique transparente para ele que as consultas são realizadas de fato na linguagem de consulta da Web semântica SPARQL. Com isso não será necessário o domínio desta linguagem pelo usuário. Uma das possibilidades neste contexto é o uso de algum tipo de ferramenta gráfica para a visualização das possíveis consultas.
- Melhor tratamento do volume de informações ontológicas de forma a otimizar seu processamento.

6. REFERÊNCIAS BIBLIOGRÁFICAS

AMANN, B.; CONSTANTIN, C.; CARON, C.; GIROUX, P. WebLab PROV: computing fine-grained provenance links for XML artifacts. **In: Proceedings of the Joint EDBT/ICDT 2013 Workshops**, p. 298-306, 2013.

ANTONIOU, G.; HARMELEN F.V. **A Semantic Web Primer**, The MIT Press, 2 edition March, 2008.

BELHAJJAME, K.; DEUS, H.; GARIJO, D.; KLYNE, G.; MISSIER, P.; SOILAND-REYES, S.; ZEDNIK, S. Prov model primer. **URL: http://www. w3. org/TR/prov-primer**, 2012.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. et al. The semantic Web. **Scientific american**, New York, NY, USA:, v. 284, n. 5, p. 28-37, 2001.

BIVAR, B.; SANTOS, L.; KOHWALTER, T. C.; MARINHO, A.; MATTOSO, M.; BRAGANHOLO, V. Uma Comparação entre os Modelos de Proveniência OPM e PROV, **In Proocedings of BRESCi 2013**, Maceió, Brazil, 2013.

BOWERS, S.; MCPHILLIPS, T.; LUDASCHER, B. Declarative rules for inferring fine-grained data provenance from scientific workflow execution traces. I**n: Provenance and Annotation of Data and Processes**, p. 82-96, 2012.

CALDIERA, V. R. B. G.; ROMBACH, H. D. The goal question metric approach. **Encyclopedia of software engineering**, v. 2, n. 1994, p. 528-532, 1994.

CARVALHO, A. d. L.; LEON, F. d. Ponce de et al. Grandes Desafios da Computação no Brasil: 2006-2016. **SEMINÁRIO GRANDES DESAFIOS DE PESQUISA EM COMPUTAÇÃO NO BRASIL**, v. 2016, 2006.

CHEBOTKO, A.; LU, S.; FEI, X.; FOTOUHI, F. RdfProv: A relational rdf store for querying and managing scientific workflow provenance**. Data & Knowledge Engineering**, Elsevier, v. 69, n. 8, p. 836-865, 2010.

CUEVAS-VICENTTIN, V.; DEY, S.; WANG, M. L. Y.; SONG, T.; LUDASCHER, B. Modeling and querying scientific workflow provenance in the d-opm. In**: Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis**, p. 119-128, 2012.

DAVIDSON, S. B.; FREIRE, J. Provenance and scientific workflows: challenges and opportunities, In: ACM. **Proceedings of the 2008 ACM SIGMOD international conference on Management of data**, p. 1345-1350, 2008.

DENTLER, K.; CORNET, R.; TEIJE, A. ten; KEIZER, N. Comparison of reasoners for large ontologies in the OWL 2 EL profile, **Semantic Web**. IOS Press, v. 2, n. 2, p.71-87, 2011.

CUEVAS-VICENTTIN, V.; DEY, S.; WANG, M. L. Y.; SONG, T.; LUDASCHER, B. Modeling and querying scientific worklow provenance in the d-opm. In: Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, p. 119-128, 2012.

FENSEL, D. **Ontologies: A Silver Bullet for Knowledge Management and Elec-tronic Commerce**. 2. ed., 2003. ISBN 3540003029.

FREIRE, J.; KOOP, D.; SANTOS, E.; SILVA, C. T. Provenance for Computational Tasks: A Survey, **Computing in Science & Enginnering**, v. 10, n. 3, p. 11-21, 2008.

GRUBER, T. R. et al. A Translation Approach to Portable Ontology Specifications, **Knowledge Acquisition**, Academic Press, v. 5, n. 2, p. 199-220, 1993.

HEBELER, J.; FISHER, M.; BLACE, R.; PEREZ-LOPEZ, A. **Semantic Web Programming**. Wiley Publishing, 2009.

LIM, C.; LU, S.; CHEBOTKO, A.; FOTOUHI, F. Storing, reasoning, and querying opm-compliant scientific workflow provenance using relational databases. **Future Gener. Comput. Syst.**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 27, n. 6, p. 781-789, jun. 2011.

LIM, C.; LU, S.; CHEBOTKO, A.; FOTOUHI, F. Prospective and retrospective provenance collection in scientific workflow environments; In: IEEE. **Services Computing(SCC), 2010 IEEE International Conference on**, p. 449-456, 2010.

MARINHO, A. S. **PROVMANAGER**: **Uma Abordagem para Gerenciamento de Proveniência de Experimentos Científicos**, dissertação de mestrado, Engenharia de Sistemas e Computação, UFRJ/COPPE, Rio de Janeiro, RJ, Brasil, 2011.

MATTOSO, M.; WERNER, C.; TRAVASSOS, G.; BRAGANHOLO, V.; MURTA, L. Gerenciando experimentos científicos em larga escala, **SBC**, p. 121–135, 2008.

MOREAU, L.; CLIFFORD, B.; FREIRE, J.; FUTRELLE, J.; GIL, Y.; GROTH, P.; KWASNIKOWSKA, N.; MILES, S.; MISSIER, P.; MYERS, J. et al. The *Open Provenance Model* core specification (v1.1). **Future Generation Computer Systems**, v. 27 n. 6, p.743 –756, 2011.

MOREAU, L.; FREIRE, J.; FUTRELLE, J.; MCGRATH, R. E.; MYERS, J.; PAULSON, P. The *Open Provenance Model*: An overview. In: **Provenance and Annotation of Data and Processes**, p. 323-326, 2008.

ROURE, D.; GOBLE, C. Supporting *e-Science* Using Semantic Web Technologies– The Semantic Grid. In: **Semantic *e-Science***. Springer, p. 1-28, 2010.

SIMMHAN, Y.; GROTH, P.; MOREAU, L. Special section: The third provenance chal-lenge on using the *Open Provenance Model* for interoperability. **Future Generation Computer Systems**, Elsevier, v. 27, n. 6, p. 737-742, 2011.

SIMMHAN, Y. L.; PLALE, B.; GANNON, D. A survey of data provenance in *e-Science*. **ACM Sigmod Record**, ACM, v. 34, n. 3, p. 31-36. 2005.

SIRIN, E.; PARSIA, B.; GRAU, B. C.; KALYANPUR, A.; KATZ, Y. Pellet: A practical owl-dl reasoner. **Web Semant.**, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 5, n. 2, p. 51-53, jun. 2007.

TRAVASSOS, G.; BARROS, M. Contributions of in virtuo and in silico experiments for the future of empirical studies in software engineering. In: **Proceedings of the ESEIW 2003: Workshop on Empirical Studies in Software Engineering**, p. 117-130, 2003.

VALENTE, W., A., G. **SciProv: uma Arquitetura para a Busca Semântica emMetadados de Proveniência no Contexto de *e-Science***, dissertação de mestrado, UFJF, Juiz de Fora, MG, Brasil, 2011.

YU L. **A Developer's Guide to the Semantic**. Atlanta: Springer, 2011. 608p.

WONG, S. C.; MILES, S.; FANG, W.; GROTH, P.; MOREAU, L. Provenance-based validation of *e-Science* experiments. In: **Proceedings of the 4th international conference on The Semantic Web**, p. 801-815, 2005.

APÊNDICE A - MODELOS DE PERSISTÊNCIA

Considerando a importância em se obter uma ampla cobertura de informações sobre proveniência, e considerando que o modelo OPM proposto em Moreau et al. (2011) prevê apenas a captura da proveniência retrospectiva, foi utilizada uma extensão do modelo OPM baseado no modelo proposto em Lim et al. (2011) que captura além da proveniência retrospectiva a proveniência prospectiva, para projetar o modelo de proveniência do SciProvMiner. O modelo de proveniência do SciProvMiner é expresso através do diagrama Entidade-Relacionamento mostrado na Figura A.1, onde a proveniência retrospectiva se encontra representada no retângulo à direita da Figura e a proveniência prospectiva representada no retângulo à esquerda.

A proveniência prospectiva modela uma especificação abstrata do workflow como uma receita para derivação de dados futuros (LIM et al. 2011). O modelo de proveniência prospectiva do SciProvMiner inclui as seguintes entidades principais: *WFModel*, *Workflow*, *Component*, *Performer* e *Port*. A entidade *WFModel* modela uma noção em alto nível do workflow. Esta entidade possui, além do atributo de chave primária Id, os atributos *InitialTask e EndTask* que representam respectivamente a tarefa inicial e final do workflow e os atributos *WorkflowRef e WorkflowVersion* para representar respectivamente, a identificação e a versão do workflow que está sendo modelado e o atributo *AllOutputsDependentAllInputs* no qual o usuário atribui o valor true para informar ao coletor de proveniência que todos os dados de saída de um processo do workflow são dependentes de todos os seus dados de entrada, e false, caso contrário. Essa informação é muito relevante no contexto do workflow, pois, caso este atributo seja configurado como true uma importante otimização de instrumentalização do workflow pode ser considerada, segundo a regra de completude de Introdução de Processo definida na documentação do modelo OPM (MOREAU et al. 2011). A entidade *Workflow* corresponde a uma execução do workflow de modelo *WFModel* a ela associado pelo relacionamento *workflowHasModel*. Por este mesmo relacionamento é modelado que uma instância de *WFModel* pode ter várias instâncias de *Workflow* associadas a ela. A entidade *Workflow* possui o atributo *Id* para identificação única das instâncias desta entidade, e o atributo *Entire* que é configurado com o valor verdadeiro se na execução do workflow corrente a proveniência prospectiva tiver sido totalmente capturada e falso caso contrário. A entidade, *Component*, que faz parte do workflow (relacionamento

partOf), representa todos os componentes do workflow, sejam parâmetros, tarefas, ou qualquer outro tipo de componente que possa ser representado em um workflow, armazenando o tipo do componente em seu atributo *Type*. Os outros atributos desta entidade são *Id* para identificação das instâncias desta entidade, e *Name* para a descrição do componente. *Task* é uma especialização da entidade *Component*, e representa uma tarefa computacional do workflow. Uma entidade do tipo *Task* pode conter e ser sub-tarefas de outras tarefas, modelado no relacionamento *Contains*. A entidade *Performer*, possuindo os atributos *Id* e *Name*, representa um sujeito, tal como um cientista ou uma ferramenta de workflow, que realiza uma tarefa, representado pelo relacionamento *performs* entre *task e performer*. O workflow também possui um *performer*, representado pelo relacionamento *hasPerformer*. A entidade *Port*, com os atributos *Id* para a sua identificação, *Name* para a sua descrição e *Type* para distinguir se aquela instância de porta é de entrada (*input*) ou de saída (*output*), representa uma porta de um componente; duas portas podem ser conectadas para formar um fluxo de dados que é capturado pelo relacionamento *IsConnectedTo*.

A partir da implementação da modelagem da proveniência prospectiva é possível coletar informações da especificação do workflow, tais como sua descrição, as tarefas que fazem parte do workflow, subtarefas de uma tarefa, o sujeito que desempenhou uma tarefa bem como as portas de entrada e saída de uma tarefa, onde a porta de saída de uma tarefa pode ser conectada a porta de entrada de outra tarefa, caracterizando assim o fluxo de dados.

A proveniência retrospectiva modela uma execução do workflow passada e a informação da derivação de dados, isto é, quais tarefas foram realizadas e como artefatos de dados foram derivados (LIM et al. 2011). O modelo de proveniência retrospectiva apresentada no diagrama de E-R do SciProvMiner é baseada no modelo OPM (MOREAU et al. 2011) e possui as entidades *OPMGraph, Account, Annotation, Agent e Process, Artifact e Agent*, que são especializações da entidade *OPM-Entity*. As dependências causais segundo o modelo OPM são modeladas a partir dos relacionamentos *used, wasGeneratedBy, wasGeneratedBy, wasDerivedFrom e wasTriggeredBy*. De acordo com o modelo OPM, grafos, artefatos, processos e agentes são identificados por identificadores únicos e as arestas de dependência causal são identificadas por suas entidades fonte, destino e pelo papel (para aquelas que tiverem papel). Na modelo E-R do SciProvMiner, as entidades *Account* e *Annotation* também são identificadas por identificadores únicos. No modelo do SciProvMiner todas as instâncias das entidades e dependências causais do modelo OPM devem ter um account associado e uma mesma instância da entidade ou dependência causal segundo o modelo OPM pode ter mais de

um *account* relacionado a ele, o que gera entre a entidade *Account* e as entidades ou as dependências causais segundo o modelo OPM relacionamentos do tipo N x N sendo, portanto, esses relacionamentos identificados de forma composta pelos identificadores da entidade ou dependência causal que está sendo relacionada, adicionado ao identificador da instância do account envolvida no relacionamento. Da mesma forma, a entidade *Annotation* ao se relacionar com as entidades associativas originadas do relacionamento da entidade *Account* com as entidades segundo o modelo OPM e suas dependências causais, geram relacionamentos do tipo N x N, que são identificados por todos os atributos identificadores pertencentes às entidades relacionadas. A entidade *OPMGraph* possui além do id, o atributo *annotation* para armazenar sua descrição. A entidade *Account* também possui um atributo para a sua descrição denominado *description*. A entidade OPM_*Entity* possui o atributo *value* que é herdado por suas especializações *Process*, *Artifact* e *Agent*. A entidade *Artifact* possui o atributo *label*, que é uma identificação secundária para as instâncias da entidade *Artifact*. A entidade *Process* e as dependências causais possuem o atributo *Fire*, que representa qual é a iteração da execução do workflow ao qual aquele processo ou dependência causal faz parte. Os relacionamentos que modelam as dependências causais *used*, *wasGeneratedBy*, *wasDerivedFrom* e *wasTriggeredBy* possuem os atributos *OTimeLower* e *OTimeUpper*, que modelam o conceito de restrição temporal definida no modelo OPM. O relacionamento que modela a dependência causal *wasControlledBy* possui os atributos *OTimeStartUpper*, *OTimeStartLower*, *OTimeEndUpper*, *OTimeEndLower*, pois para esta dependência causal o modelo OPM permite duas marcações temporais (*OTimeStart*, e *OTimeEnd*).

Os relacionamentos entre os modelos de proveniência prospectiva e retrospectiva são capturados pelos relacionamentos *InstanceOf, WasOutputByPort* e *WasInputByPort*, uma vez que *OPMGraph, Process* e *Agent* são instanciações em tempo de execução de *Workflow, Task* e *Performer* respectivamente, e *Artifact* pode ser consumido ou produzido por uma instância de *Port*.

O esquema para capturar a proveniência prospectiva conta com dez tabelas que são: *WFModel, Workflow, Component, Contains, Performs, Performer, Workflow_has_Performer, Port, Component_has_port*, e *IsConnectedTo*. O esquema de banco de dados para a proveniência retrospectiva possui 28 tabelas, e se encontra ilustrado do lado direito da Figura A.2. As entidades do modelo E-R mostradas na Figura A.1 se tornaram as tabelas OPMGraph, Account, Annotation, e a entidade Entity_OPM foi especializada nas tabelas *Process*, *Artifact* e *Agent* e não foi transformada em tabela física do modelo relacional. Todas as dependências

causais modeladas no diagrama E-R da figura A.2 eram relacionamentos N x N entre as entidades do modelo OPM, e, portanto, foram transformadas nas tabelas *Used*, *WasGeneratedBy*, *WasGeneratedBy*, *WasTriggeredBy* e *wasDerivedFrom*. Nas relações entre a proveniência prospectiva e retrospectiva, o único relacionamento N x N que ocorreu foi o *wasInputToPort* entre as entidades *Port* e *Artifact* do diagrama E-R da Figura A.2. Este relacionamento se transformou na tabela *WasInputToPort*.

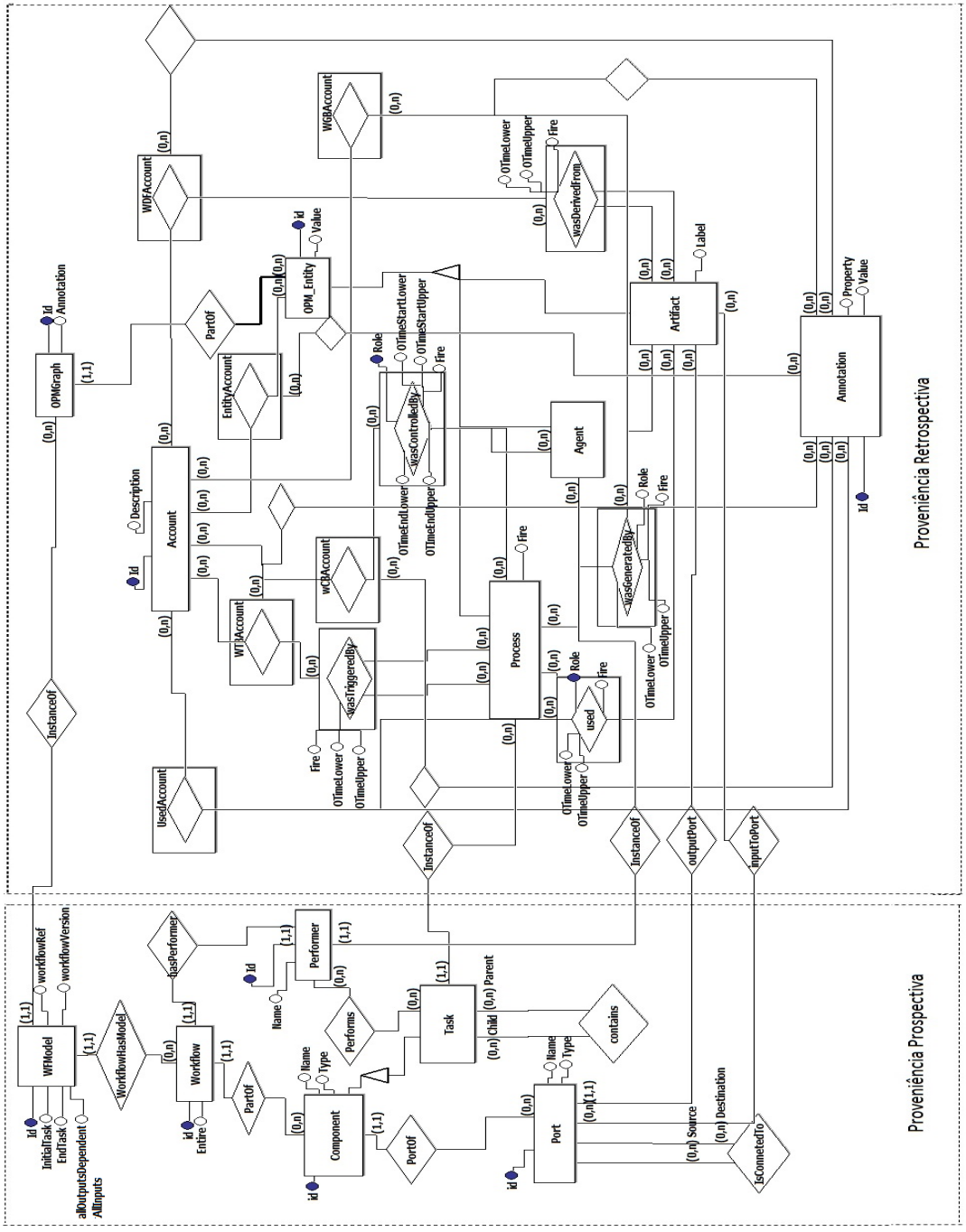


Figura A.1. Diagrama E-R do SciProvMiner

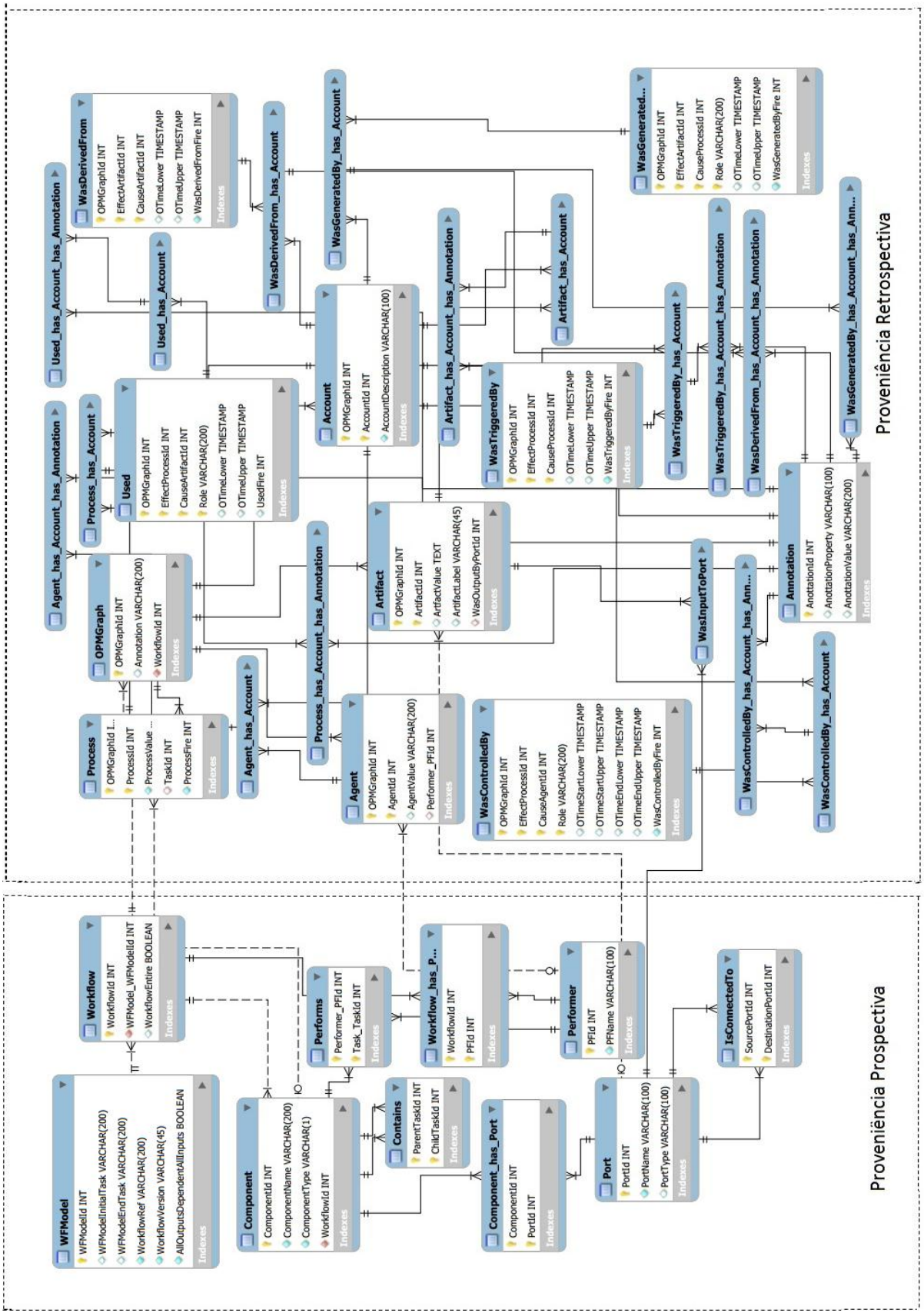


Figura A.2 Modelo Relacional do SciProvMiner

APÊNDICE B - INSTRUMENTALIZAÇÃO

Segundo as regras de negócio adotadas pelo SciProvMiner, cada vez que um serviço Web for instrumentalizado com um método associado a uma dependência causal que se relacione com a entidade Artefato, correspondendo aos métodos *used* (associado à dependência causal *Used*, segundo o modelo OPM), *wasDerivedFrom* (associado à dependência causal *wasDerivedFrom*, segundo o modelo OPM) e *wasGeneratedBy* (associado à dependência causal *wasGeneratedBy),* além de um parâmetro para a captura do Artefato propriamente dito, haverá um parâmetro denominado *wasOutputByPort* que receberá a informação da porta de saída do componente do worfklow pela qual o artefato fica disponível para que outros componentes do workflow possam consumi-lo. Esse parâmetro deve ser colocado para cada artefato presente na dependência causal que está sendo modelada.

Nos métodos do serviço Web denominados *used* e *wasDerivedFrom*, além do parâmetro *wasOutputByPort*, o parâmetro *wasInputToPort* também deve ser informado, para artefatos que fizerem parte da dependência causal que está sendo capturada. Este parâmetro captura a informação da porta de entrada da tarefa do workflow pela qual este artefato está entrando para ser utilizado. Como esses métodos capturam as informações de qual componente do workflow e porta um artefato saiu, e por qual porta de qual componente um artefato entrou, é possível capturar a conexão de portas, que forma o fluxo de dados do workflow. Este fluxo de dados é persistido no relacionamento *IsConnectedTo* do modelo de banco de dados SciProvMiner. Outras entidades do modelo de dados que capturam informações de proveniência prospectiva, também são persistidas pelos parâmetros *wasInputToPort* e *wasOutputByPort*, são as entidades *Component* e *Port*, sendo que, se o componente for do tipo tarefa, ele pode ser uma subtarefas, e então, a entidade *Contains* também será persistida. Os métodos que possuem o parâmetro *wasOutpuByPort* persistem os dados do relacionamento entre as entidades *Artifact* e *Port*, através do atributo *wasOutputByPort* da entidade *Artifact*, que associam informações de proveniência prospectiva e retrospectiva. Os métodos que possuem o parâmetro *wasInputToPort* persistem o relacionamento entre *Port* e *Artifact* através da entidade *wasInputToPort*, sendo também um relacionamento entre proveniência prospectiva e retrospectiva. Os métodos que possuem o parâmetro *Artifact* persistem as informações da entidade *Artifact*, que corresponde a informações de proveniência retrospectiva.

As regras de negócios adotadas no SciProvMiner determinam que todos os métodos do serviço Web que precisarem capturar o processo, correspondendo aos métodos *used*, *wasGeneratedBy*, *wasGeneratedBy*, *wasTriggeredBy*, terão além do parâmetro *Process* para capturar as informações do processo que está sendo modelado, o parâmetro *Task*, que representa uma tarefa computacional que faz parte do workflow, ao qual aquele processo está associado. Uma tarefa é um tipo de componente, e por isso os dados da tarefa são persistidos na entidade *Component* do modelo de dados do SciProvMiner. Se a tarefa for uma subtarefa então a entidade *Contains* também será persistida. Com exceção do método *wasGeneratedBy*, todos os outros métodos acima descritos possuem o parâmetro *performer*, para receber o sujeito que realizou a tarefa, tal como um cientista ou uma ferramenta de workflow. As informações dadas neste parâmetro são persistidas na entidade *Performer* do banco de dados. O relacionamento entre a tarefa e o sujeito que realizou a tarefa é persistido na entidade *Performs*. Até aqui, todas as informações persistidas estão relacionadas à captura de proveniência prospectiva. Já as informações do processo, que dizem respeito à captura de proveniência retrospectiva, são persistidas na entidade *Process*, sendo que o relacionamento entre o processo e a tarefa é persistida no atributo *TaskId* da entidade *Process*. Este relacionamento envolve informações de proveniência prospectiva e retrospectiva.

Todos os métodos que capturam dependências causais segundo o modelo OPM, possuem os parâmetros *account* e *annotation*, que capturam os conceitos das entidades do modelo OPM de mesmo nome.

O formato de entrada dos dados varia de parâmetro para parâmetro, de acordo com as informações que devem ser capturadas por aquele parâmetro. A maioria dos parâmetros são strings simples. Abaixo são especificados os formatos de entrada de dados para cada um dos possíveis parâmetros dos serviços Web que possuem um formato especial de entrada.

- **artifact**: "label#valor": exemplo: "a1#45", sendo que o label representa esse parâmetro unicamente no workflow. A Figura 3.5 ilustra como os artefatos são instrumentalizados para serem capturados no workflow SimpleAddition desenvolvido no SGWfC Kepler.
- ***Task***: se for uma **subtarefa** será preenchido da seguinte forma: “nomeTarefa” .“nomeSubtarefa”. Ex.: AddOrSubtract.Add. Se for uma tarefa vai conter apenas o nome da tarefa. Exemplo: AddOrSubtract.
- ***wasOutputByPort* e *wasInputToPort***: tipo do componente [C,T, NP] # componente# porta, onde tipo da porta [C, T, NP] determina o tipo do componente, sendo T para representar o tipo de componente que é uma tarefa computacional que faz parte do

workflow, NP é um componente que não possui porta, como por exemplo um parâmetro, e C um componente que não é uma tarefa, mas possui porta, com o é o caso de uma constante ou uma expressão. Quando um componente é do tipo NP, o parâmetro “nome da porta” não precisa ser fornecido pelo usuário. O segundo parâmetro componente é preenchido pelo nome do componente. Se este componente for uma tarefa ele deve ser preenchido como um parâmetro do tipo “*Task*”. A porta deve ser preenchida pelo nome da porta associada a aquele componente que desempenha o papel esperado pelo parâmetro *wasOutputByPort* ou *wasInputToPort*, dependendo de qual desses parâmetros estão sendo preenchidos. Por exemplo, o parâmetro *wasOutputByPort* preenchido com a string "T#AddFunction#output" diz que o artefato em questão veio (ou saiu) da porta de nome output da tarefa AddFunction.

- **Annotation**: Todas as entidades do modelo OPM que estiverem sendo representadas na dependência causal relacionada ao método selecionado podem ser anotadas neste parâmetro. O formato é [nomeEntidade]#anotation property #anotation value,[nomeEntidade]#anotation property#anotation value, onde nomeEntidade é o nome da entidade que está sendo instrumentalizada (nome do processo para entidade OPM *Process*, label do artefato para entidade OPM *Artifact*, nome do agente para entidade OPM *Agent*), a tralha é o separador entre o tipo de entidade e o texto de anotação. A vírgula separa as anotações. A quantidade de vírgulas + 1 denota a quantidade de anotações que foram realizadas naquela instrumentalização. Quando for anotar a dependência causal que o método daquela instância do serviço Web se propõe a capturar, em *nomeEntidade* coloca-se o nome da dependência causal que a aresta será anotada. Por exemplo, a anotação "*wasDerivedFrom*#subtype# AddFunctionOperator1" captura a anotação relacionada a dependência causal *wasDerivedFrom*, com a property *anotation* valendo subtype e o annotation value como AddFunctionOperator1.

Para diminuir o trabalho do cientista na tarefa de instrumentalização do workflow, quando um artefato for instrumentalizado para mais de uma dependência causal, com os mesmos valores para *wasInputToPort* e *wasOutputByPort* não é necessário instrumentalizar esses parâmetros repetidamente. Se na instrumentalização de ao menos uma dessas dependências causais esses parâmetros tiverem sido capturados já é o suficiente. É importante salientar que quanto mais padronizado for o nome das atividades do workflow e de suas portas de entrada e saída, mais facilitada será sua instrumentalização. Também é importante a criação de padrões para a instrumentalização do workflow, uma vez que a atividade de instrumentalização é feita

de maneira manual pelo cientista. Por exemplo, no workflow representado na Figura 3.5, para cada atividade foram definidos três parâmetros, que são *pProcessName*, *pTaskName* e *pPerformerName*, conforme Figura B.1 Esses parâmetros facilitam e tornam menos susceptíveis a erro o processo de instrumentalização, pois como ocorre redundância de solicitação de informações nas instâncias do serviço Web do SciProvMiner, quando essas informações se encontram parametrizadas, o usuário faz apenas uma referência à informação nos parâmetros do serviço Web de instrumentalização, não redigitando-as toda vez, como pode ser visto na Figura B.2 nos parâmetros "*workflow", "workflowVersion", "InitialTask"* e "*EndTask*". Essa padronização deve ser realizada de acordo com os recursos que cada SGWfC oferece.

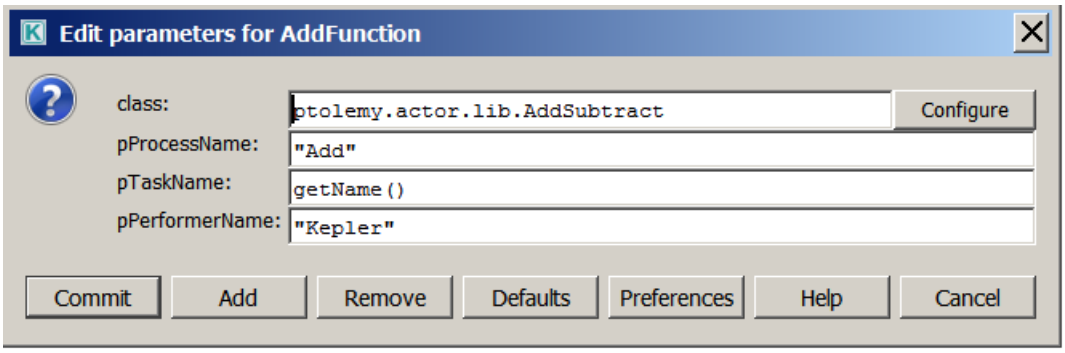


Figura B.1. Parâmetros do método *initialConfiguration*

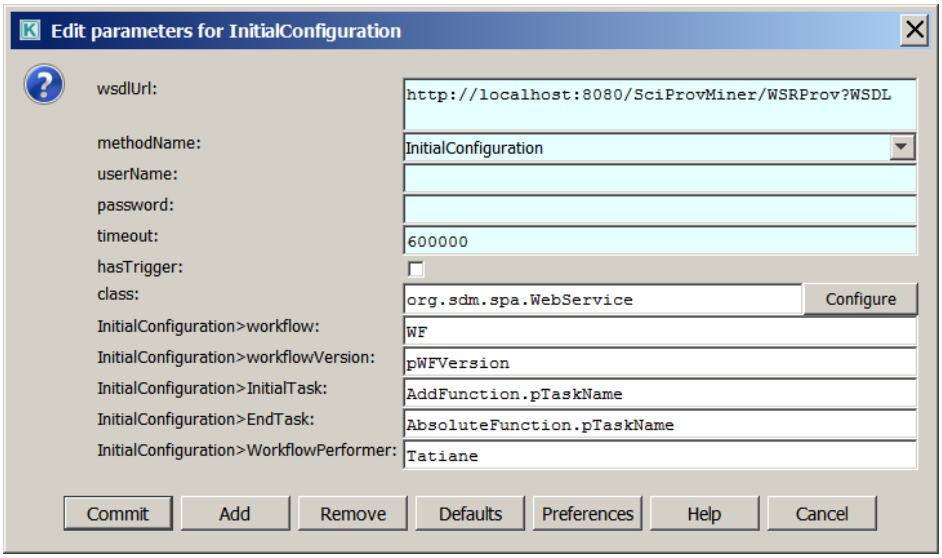


Figura B.2. Parâmetros do método *initialConfiguration*

Considerando a importância do método *initialConfiguration*, é importante detalhar seus parâmetros, ilustrados na Figura B.2, e detalhados a seguir:

- workflow: Dever ser preenchido com o nome do workflow que está sendo instrumentalizado para a captura da proveniência. No exemplo apresentado na Figura B.2, a sigla WF representa uma referência para “SimpleAddition”, pois WF é um parâmetro do workflow conforme ilustrado na Figura 3.5.
- workflowVersion: A cada vez que o cientista fizer alguma alteração no workflow, após ter sido feita uma coleta de proveniência, deve ser dada uma nova versão para o workflow. Neste parâmetro o cientista entra com a versão do workflow que terá a linhagem capturada. No exemplo apresentado na Figura B.2, pWFVersion representa uma referência para o valor “1”, pois pWFVersion é um parâmetro do workflow, como pode ser visto na Figura 3.5.
- InitialTask: deve ser preenchido com o nome da primeira tarefa a ser executada pelo workflow. No exemplo apresentado na Figura B.2, este parâmetro está com o valor AddFunction.pTaskName, que é uma referência para o nome da primeira tarefa descrita no workflow SimpleAddition, conforme padronização definida.
- EndTask: dever ser preenchido com o nome da última tarefa a ser executada pelo workflow. No exemplo apresentado na Figura B.2, este parâmetro está com o valor AbsoluteFunction.pTaskName, que é uma referência para o nome da primeira tarefa descrita no workflow SimpleAddition, conforme padronização definida na seção anterior.
- WorkflowPerformer: deve ser preenchido com o nome da pessoa que projetou o workflow. No exemplo apresentado na Figura B.2, este parâmetro foi preenchido com o valor “Tatiane”, significando que esta pessoa desenvolveu este workflow.
- *AllOutputsDependentAllInputs*: deve ser preenchido com o valor “true” caso todos os dados de saída de um processo do workflow são dependentes de todas as suas entradas.

Quando o método *initialConfiguration* é executado, ele persiste as informações no banco de dados, e retorna o workflowId, que é a identificação do workflow que está sendo executado, e o OPMGraphId, que representa qual a execução daquele workflow que está sendo armazenada, visto que um mesmo workflow, em uma mesma versão pode ser executado diversas vezes. O formato de retorno é workflowid#OPMGraphId, conforme especificado previamente. Na Figura 3.5, que ilustra o exemplo de instrumentalização do workflow *SimpleAddition*, pode ser visto que todas as demais instâncias do serviço Web instrumentalizadas no workflow recebem o retorno da instância configurada com o método *initialConfiguration* como valor para o parâmetro de entrada denominado *InitialParameters*.

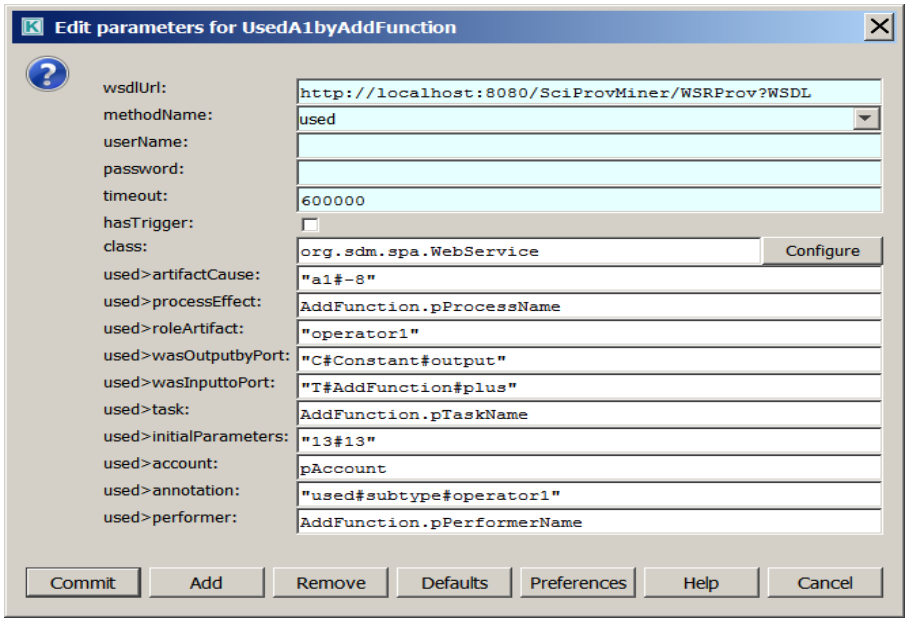


Figura B.3. Parâmetros Método *Used*

Quando o usuário deseja instrumentalizar uma dependência causal *used* segundo o modelo OPM, ele deve adicionar uma instância do serviço Web do SciProvMiner no workflow, selecionar o serviço *used* e preencher os parâmetros solicitados por ele. Os parâmetros solicitados pelo método *used*, ilustrados na Figura B.3, são:

- artifactCause: deve ser preenchido com os dados do artefato que é a causa da dependência causal *used*. Na Figura B.3, este parâmetro recebe o artefato instrumentalizado no formato "artifactLabel#artifactValue", através da ligação do valor de saída do componente *Artifact* ao parâmetro artifactCause da instância de serviço Web denominado UsedA1ByAddFunction.
- processEffect: deve ser preenchido com os dados do processo P que é o efeito da dependência causal *used*. Na Figura B.3, este parâmetro é preenchido com o valor AddFunction.pProcess, que é uma referência para o nome do processo associado à tarefa AddFunction do workflow, conforme padronização descrita anteriormente.
- roleArtifact: este parâmetro deve ser preenchido com o papel desempenhado pelo artefato A no processo P. Na Figura B.3, neste parâmetro é colocado o valor "operator1", significando que o artefato de label a1 desempenha o papel de "operator1" na atividade AddFunction.
- *wasOutputByPort*: a identificação da porta pela qual o artefato A foi originado. Este parâmetro é preenchido de acordo com a especificação do formato adequado para o seu

preenchimento, conforme detalhado na seção anterior. Na Figura B.3 este parâmetro é preenchido com o valor “C#Constant#output”, que significa que o artefato utilizado na dependência causal instrumentalizada neste serviço Web veio da porta output do componente Constant do workflow.

- *wasInputToPort*: a identificação da porta da tarefa T pela qual o artefato A foi consumido. Assim como o parâmetro anterior, este parâmetro é preenchido de acordo com a especificação do formato adequado para o seu preenchimento, conforme detalhado na seção anterior. No exemplo apresentado na Figura B.3 este parâmetro é preenchido com o valor “T#AddFunction#plus”, que significa que o artefato utilizado na dependência causal instrumentalizada neste serviço Web entrou na tarefa AddFunction pela porta denominada *plus*.
- task: deve ser definida a tarefa ou atividade T do workflow ao qual o Processo P está relacionado. Na Figura B.3, este parâmetro é preenchido com o valor AddFunction.pTaskName, que é uma referência para o nome da tarefa descrita no workflow, conforme padronização definida anteriormente neste apêndice.
- initialParameters: neste parâmetro deve ser informado o WorkflowId e o OPMGraphId, no formato “workflowId#OPMGraphId” retornado pela instância do serviço Web configurado com o método *InitialConfiguration* para aquele workflow. Neste parâmetro é necessário apenas uma referência ao valor retornado pela instância do serviço Web configurado com o método *InitialConfiguration.* A Figura B.4 apresenta como esta referência é feita no workflow SimpleAddition modelada no SGWfC Kepler, através da ligação do dado gerado pela instância do serviço Web denominada *InitialConfiguration* ao parâmetro de entrada denominado initialParameters da instância do serviço Web de nome UsedA1ByAddFunction.
- Account: neste parâmetro é definido o Account ao qual essa dependência causal, juntamente com entidades do modelo OPM relacionados nesta dependência, fazem parte. Na Figura B.3 este parâmetro é preenchido com o valor pAccount que é uma referência para o parâmetro do workflow denominado pAccount com o valor “Black”, que pode ser visualizado na Figura B.4.
- Annotation: anotações segundo o modelo OPM, das entidades envolvidas nesta dependência causal, inclusive a própria dependência causal. Na Figura B.3este parâmetro é expresso com o valor “*used*#subtype#operator1”, significando uma anotação para a dependência causal que está sendo instrumentalizada com *subtype* como valor de

annotation property e operator1 como valor de annotation value, segundo especificação do formato adequado para o preenchimento deste parâmetro descrito anteriormente.

- Performer: deve ser preenchido com o nome da pessoa ou ferramenta que realiza a tarefa, tal como um cientista ou uma ferramenta de workflow. Na Figura B.3, este parâmetro é preenchido com o valor AddFunction.pPerformerName, que é uma referência para o nome do performer da tarefa AddFunction descrita no workflow, conforme padronização definida na seção anterior.

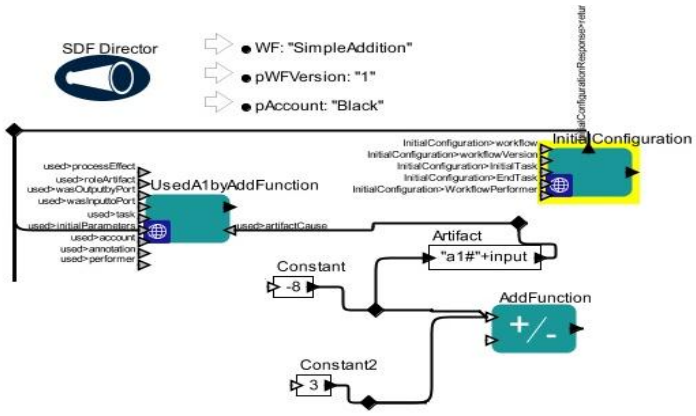


Figura B.4. Recorte da instrumentalização do workflowSimpleAddition focando na instância do serviço Web do SciProvMiner com o método *Used* denominado UsedA1ByAddFunction

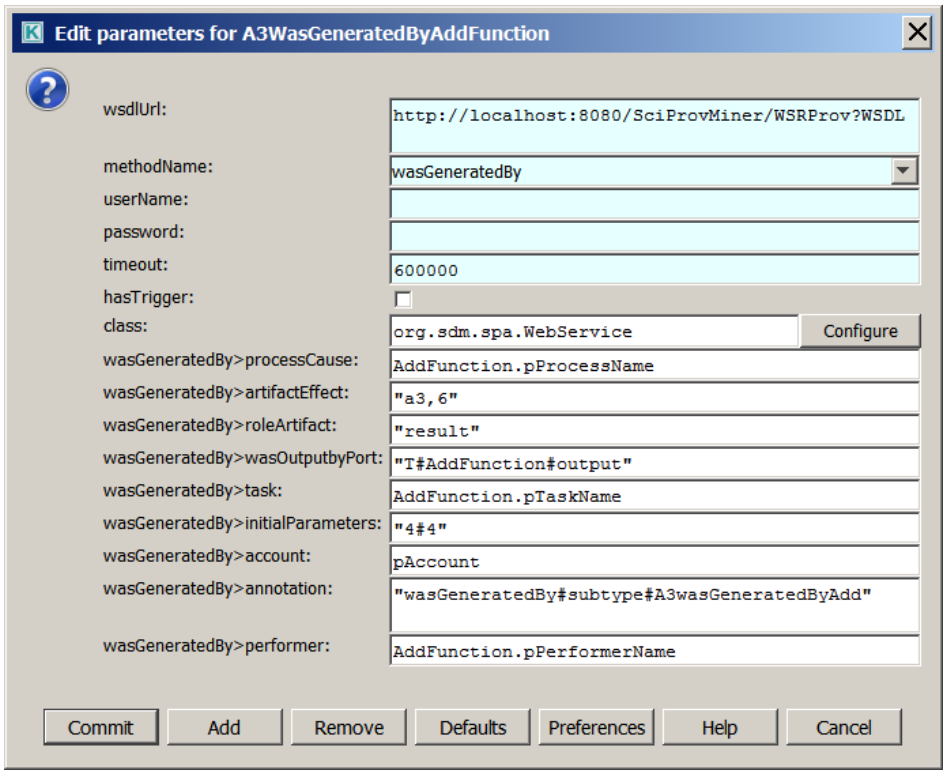


Figura B.5. Parâmetros do Método *wasGeneratedBy*

Para instrumentalizar uma dependência causal *wasGeneratedBy* segundo o modelo OPM, deve-se adicionar uma instância do serviço Web do SciProvMiner no workflow, selecionar o serviço *wasGeneratedBy* e preencher os parâmetros solicitados por ele. Os parâmetros solicitados pelo método *wasGeneratedBy*, ilustrados na Figura B.5, são:

- processCause: deve ser preenchido com os dados do processo P que é a causa da dependência causal *wasGeneratedBy*. Na Figura B.5, este parâmetro é preenchido com o valor AddFunction.pProcess, que é uma referência para o nome do processo associado a tarefa AddFunction do workflow, conforme padronização descrita na seção anterior.
- artifactEffect: deve ser preenchido com os dados do artefato que é o efeito da dependência causal *wasGeneratedBy*. Na Figura B.5, pode-se notar que este parâmetro recebe o artefato instrumentalizado no formato "artifactLabel#artifactValue", através da ligação do valor de saída do componente Artifact3 do workflow ao parâmetro artifactEffect da instância de serviço Web denominado UsedA1ByAddFunction.
- roleArtifact: este parâmetro deve ser preenchido com o papel desempenhado pelo artefato A no processo P. Na Figura B.5, neste parâmetro é colocado o valor "result", significando que o artefato de label a3 desempenha o papel de "result" na atividade AddFunction.
- *wasOutputByPort*: a identificação da porta de saída da tarefa T pela qual o artefato A poderá ser consumido por outras atividades. Este parâmetro é preenchido de acordo com a especificação do formato adequado para o seu preenchimento, conforme detalhado na seção anterior. Na Figura B.5 neste parâmetro é atribuído o valor "T#AddFunction#output", que significa que o artefato utilizado na dependência causal instrumentalizada neste serviço Web será disponibilizado para ser consumido por outras atividades através da porta output da tarefa AddFunction do workflow.
- task: Deve ser definida a tarefa ou atividade T do workflow ao qual o Processo P está relacionado. Na Figura B.5, este parâmetro é preenchido com o valor AddFunction.pTaskName, que é uma referência para o nome da tarefa descrita no workflow, conforme padronização definida na seção anterior.
- initialParameters: neste parâmetro deve ser informado o WorkflowId e o OPMGraphId, no formato "workflowId#OPMGraphId" retornado pela instância do serviço Web configurado com o método *InitialConfiguration* para aquele workflow. Neste parâmetro é necessário apenas uma referência ao valor retornado pela instância do serviço Web configurado com o método *InitialConfiguration*. A Figura B.6 apresenta como esta

referência é feita no workflow SimpleAddition modelada no SGWfC Kepler, através da ligação do dado gerado pela instância do serviço Web denominada *InitialConfiguration* ao parâmetro de entrada denominado initialParameters da instância do serviço Web de nome A3WasGeneratedByAddFunction.

- Account: neste parâmetro é definido o Account ao qual essa dependência causal, juntamente com entidades do modelo OPM relacionados nesta dependência, fazem parte. Na Figura B.5 este parâmetro é preenchido com o valor pAccount que é uma referência para o parâmetro do workflow denominado pAccount com o valor "Black", que pode ser visualizado na Figura B.6.
- Annotation: anotações segundo o modelo OPM das entidades envolvidas nesta dependência causal, inclusive a própria dependência causal. Na Figura B.5 este parâmetro é expresso com o valor "*wasGeneratedBy*#subtype# A3wasGeneratedByAddFunction", significando uma anotação para a dependência causal que está sendo instrumentalizada tendo como valor de annotation property a string "subtype" e como valor de annotation value a string "A3WasGeneratedByAddFunction", segundo especificação do formato adequado para o preenchimento deste parâmetro descrito na seção anterior.
- Performer: deve ser preenchido com o nome da pessoa ou ferramenta que realiza a tarefa, tal como um cientista ou uma ferramenta de workflow. Na Figura B.5, este parâmetro é preenchido com o valor AddFunction.pPerformerName, que é uma referência para o nome do performer da tarefa AddFunction descrita no workflow, conforme padronização definida na seção anterior.

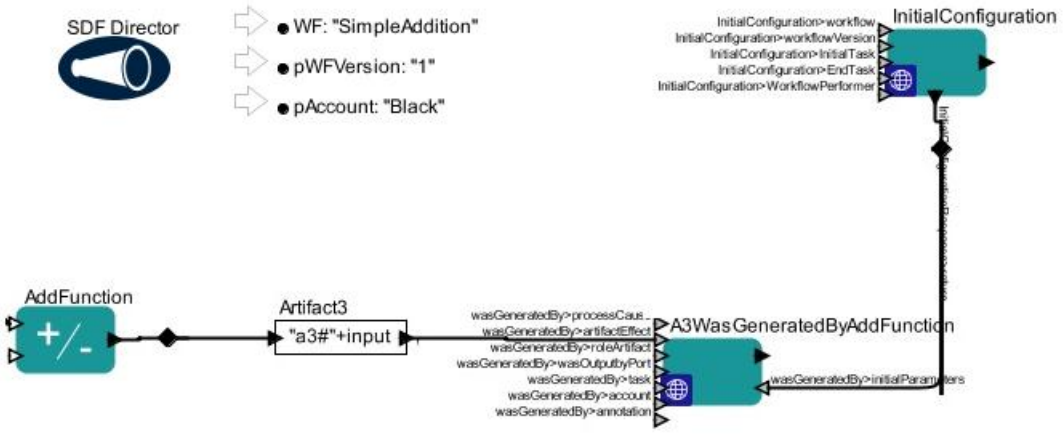


Figura B.6. Recorte da instrumentalização do workflowSimpleAddition focando na instância do serviço Web do SciProvMiner com o método *wasGeneratedBy* denominado A3WasGeneratedByAddFunction

Pode ser observado que, apesar da dependência causal *wasGeneratedBy* ter entre suas entidades relacionadas um artefato, o *wasInputToPort* não se encontra dentre os parâmetros do método *wasGeneratedBy*. Isso se deve ao fato da dependência causal *wasGeneratedBy* relacionar o artefato que está sendo gerado por um processo em que ambos se encontram associados a uma mesma tarefa, no caso a tarefa AddFunction, não havendo fluxo de dados entre atividades do workflow. O artefato gerado pelo processo nesta dependência causal poder ser consumido por diversas outras tarefas ou não ser consumido por tarefa nenhuma e, por essas razões, foi considerado que este parâmetro não deve ser informado neste método.

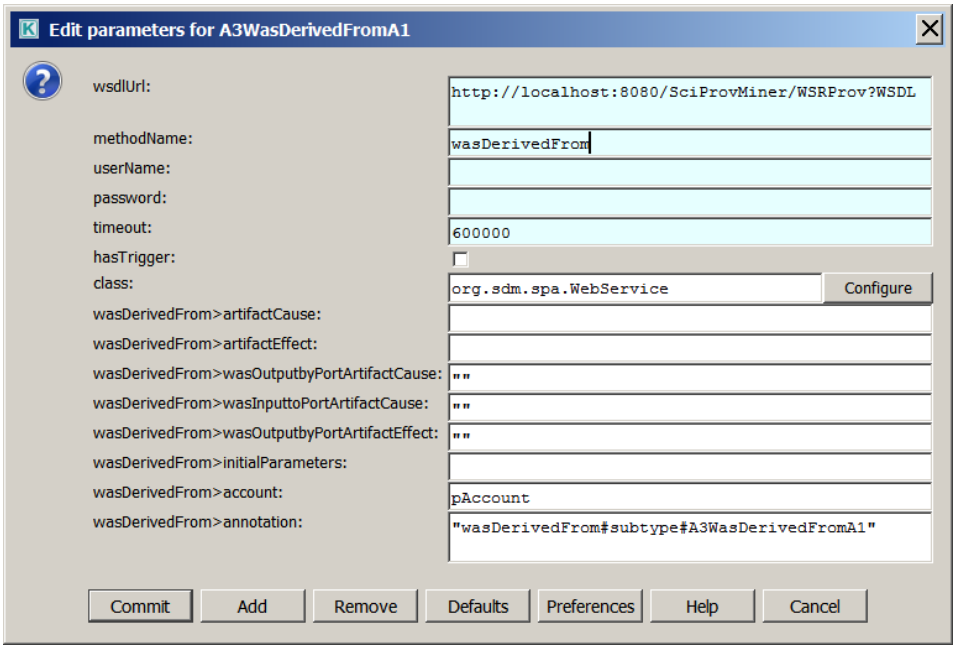


Figura B.7. Parâmetros do Método *wasDerivedFrom*

Para instrumentalizar uma dependência causal *wasDerivedFrom* segundo o modelo OPM, deve-se adicionar uma instância do serviço Web do SciProvMiner no workflow, selecionar o serviço *wasDerivedFrom* e preencher os parâmetros solicitados por ele. Os parâmetros solicitados pelo método *wasDerivedFrom*, ilustrados na Figura B.7, são:

- artifactCause: deve ser preenchido com os dados do artefato que é a causa da dependência causal *wasDerivedFrom*. Na Figura B.7, este parâmetro recebe o artefato instrumentalizado no formato "artifactLabel#artifactValue", através da ligação do valor de saída do componente *Artifact* ao parâmetro artifactCause da instância de serviço Web denominado A3*WasDerivedFrom*A1.
- artifactEffect: deve ser preenchido com os dados do artefato que é o efeito da dependência causal *wasDerivedFrom*. Na Figura B.7, pode-se notar que este parâmetro

recebe o artefato instrumentalizado no formato “artifactLabel#artifactValue”, através da ligação do valor de saída do componente Artifact3 do workflow ao parâmetro artifactEffect da instância de serviço Web denominado A3*WasDerivedFrom*A1.

- *wasOutputByPort*ArtifactCause: a identificação da porta pela qual o artefato causa pode ser consumido por outras tarefas do workflow. Este parâmetro é preenchido de acordo com a especificação do formato adequado para o seu preenchimento, conforme detalhado na seção anterior. Na Figura B.7este parâmetro é preenchido com uma string vazia representada por “”. Isso se deve ao fato desta informação já ter sido instrumentalizada na instância de serviço Web denominada AbsoluteFunctionUsedA3, e portanto não precisar ser repetida aqui, diminuindo desta forma o trabalho do cientista na instrumentalização do workflow.
- *wasInputToPort*ArtifactCause: a identificação da porta da tarefa T pela qual o artefato causa modelado nesta dependência causal consome este artefato. Assim como o parâmetro anterior, este parâmetro é preenchido de acordo com a especificação do formato adequado para o seu preenchimento, conforme detalhado na seção anterior. No exemplo apresentado na Figura B.7 este parâmetro também é preenchido com uma string vazia representada por “”. Da mesma forma que o parâmetro anterior, isso se deve ao fato desta informação já ter sido instrumentalizada na instância de serviço Web do SciProvMiner denominada AbsoluteFunctionUsedA3.
- *wasOutputByPort*ArtifactEffect: a identificação da porta pela qual o artefato efeito desta dependência causal pode ser consumido por outras tarefas do workflow. Este parâmetro é preenchido de acordo com a especificação do formato adequado para o seu preenchimento, conforme detalhado na seção anterior. Na Figura B.7 este parâmetro é preenchido com uma string vazia representada por “”. Isso se deve ao fato desta informação já ter sido instrumentalizada na instância de serviço Web denominada AddFunctionUsedA1, e portanto não precisar ser repetida aqui, diminuindo desta forma o trabalho do cientista na instrumentalização do workflow.
- initialParameters: Neste parâmetro deve ser informado o WorkflowId e o OPMGraphId, no formato “workflowId#OPMGraphId” retornado pela instância do serviço Web configurado com o método *InitialConfiguration* para aquele workflow. Neste parâmetro é necessário apenas uma referência ao valor retornado pela instância do serviço Web configurado com o método *InitialConfiguration.* A Figura B.8 apresenta como esta referência é feita no workflow SimpleAddition modelada no SGWfC Kepler, através da

ligação do dado gerado pela instância do serviço Web denominada *InitialConfiguration* ao parâmetro de entrada denominado initialParameters da instância do serviço Web de nome A3*WasDerivedFrom*A1.

- Account: neste parâmetro é definido o Account ao qual essa dependência causal, juntamente com entidades do modelo OPM relacionados nesta dependência, fazem parte. Na Figura B.7este parâmetro é preenchido com o valor pAccount que é uma referência para o parâmetro do workflow denominado pAccount com o valor "Black", que pode ser visualizado na Figura B.8.
- Annotation: anotações segundo o modelo OPM das entidades envolvidas nesta dependência causal, inclusive a própria dependência causal. Na Figura B.7 este parâmetro é expresso com o valor "*wasDerivedFrom*#subtype# A3*WasDerivedFrom*A1", significando uma anotação para a dependência causal que está sendo instrumentalizada com subtype como valor de annotation property e A3*WasDerivedFrom*A1como valor de annotation value, segundo especificação do formato adequado para o preenchimento deste parâmetro descrito na seção anterior.

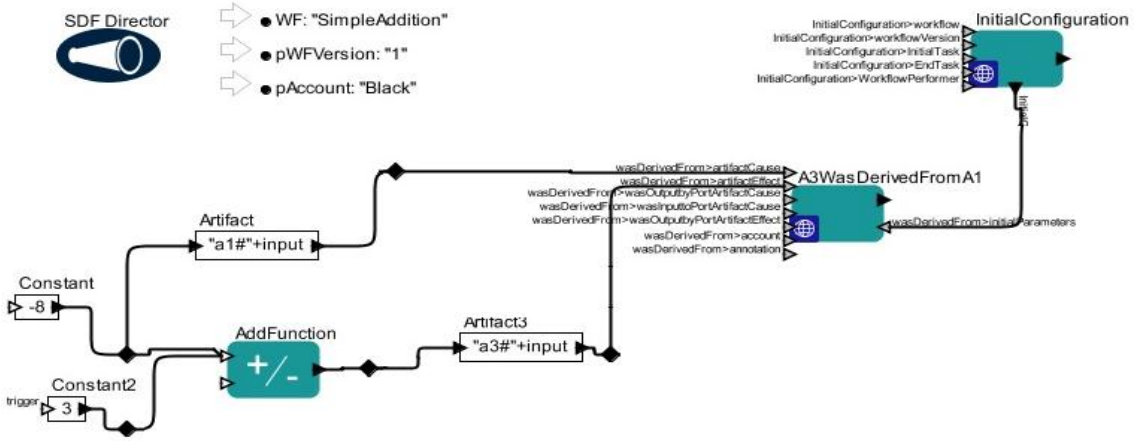


Figura B.8. Recorte da instrumentalização do workflowSimpleAddition focando na instância de serviço Web do SciProvMiner com o método *wasDerivedFrom* denominado A3*WasDerivedFrom*A1

Pode ser observado que, apesar da dependência causal *wasDerivedFrom* ter dois artefatos se relacionando, o *wasInputToPort* do artefato efeito não se encontra dentre os parâmetros do método *wasDerivedFrom*. O artefato efeito derivado pelo artefato causa nesta dependência causal poder ser consumido por diversas outras tarefas ou não ser consumido por tarefa nenhuma e, por essas razões, foi considerado que este parâmetro não deve ser informado neste método.

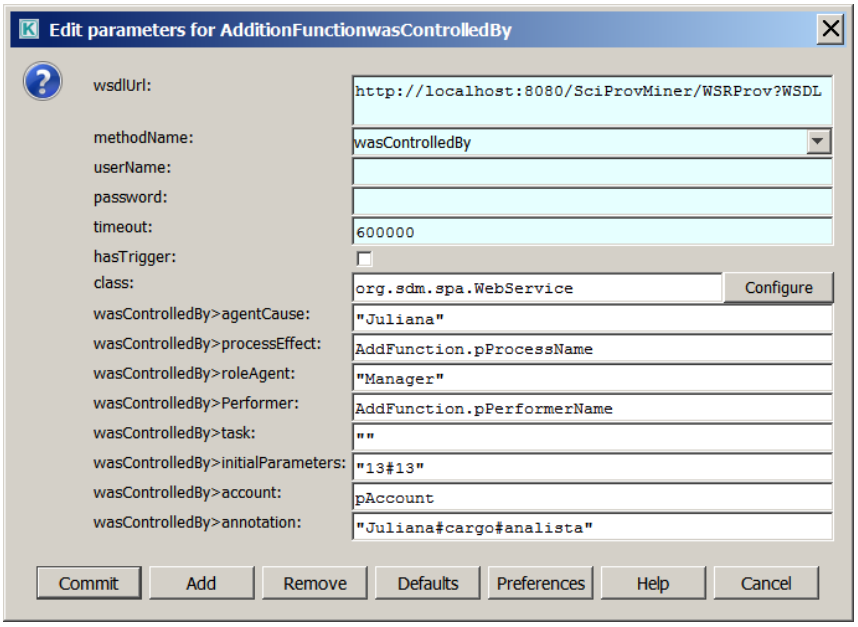


Figura B.9. Parâmetros do Método *wasGeneratedBy*

Para instrumentalizar uma dependência causal *wasControlledBy* segundo o modelo OPM, deve-se adicionar uma instância do serviço Web do SciProvMiner no workflow, selecionar o método *wasGeneratedBy* e preencher os parâmetros solicitados por ele. Os parâmetros solicitados pelo método *wasGeneratedBy*, ilustrados na Figura B.9, são:

- agentCause: deve ser preenchido com o nome do agente que é a entidade causa da dependência causal *wasControlledBy* segundo o modelo OPM. Na Figura B.9, este parâmetro recebe o a string “Juliana” como valor para este parâmetro.
- processEffect: deve ser preenchido com os dados do processo P que é o efeito da dependência causal *used*. Na Figura B.9, este parâmetro é preenchido com o valor AddFunction.pProcess, que é uma referência para o nome do processo associado à tarefa AddFunction do workflow, conforme padronização descrita na seção anterior.
- roleAgent: este parâmetro deve ser preenchido com o papel que agente A desempenha sobre o processo P na dependência causal *wasControlledBy*. Na Figura B.9,neste parâmetro é colocado o valor “Manager”, significando que o agente “Juliana” desempenha o papel de “manager” na atividade AddFunction.
- performer: deve ser preenchido com o nome da pessoa ou ferramenta que realiza a tarefa, tal como um cientista ou uma ferramenta de workflow. Na Figura B.9, este parâmetro é preenchido com o valor AddFunction.pPerformerName, que é uma

referência para o nome do performer da tarefa AddFunction descrita no workflow, conforme padronização definida na seção anterior.

- task: deve ser definida a tarefa ou atividade T do workflow ao qual o Processo P está relacionado. Na Figura B.9, este parâmetro é preenchido com o valor AddFunction.pTaskName, que é uma referência para o nome da tarefa descrita no workflow, conforme padronização definida na seção anterior.
- initialParameters: neste parâmetro deve ser informado o WorkflowId e o OPMGraphId, no formato "workflowId#OPMGraphId" retornado pela instância do serviço Web configurado com o método *InitialConfiguration* para aquele workflow. Neste parâmetro é necessário apenas uma referência ao valor retornado pela instância do serviço Web configurado com o método *InitialConfiguration*. A Figura B.10 apresenta como esta referência é feita no workflow *SimpleAddition* modelada no SGWfC Kepler, através da ligação do dado gerado pela instância do serviço Web denominada *InitialConfiguration* ao parâmetro de entrada denominado initialParameters da instância do serviço Web de nome *AdditionFunctionwasControlledBy*.
- Account: neste parâmetro é definido o Account ao qual essa dependência causal, juntamente com entidades do modelo OPM relacionados nesta dependência, fazem parte. Na Figura B.9este parâmetro é preenchido com o valor pAccount que é uma referência para o parâmetro do workflow denominado pAccount com o valor "Black", que pode ser visualizado na Figura B.10.
- Annotation: anotações segundo o modelo OPM das entidades envolvidas nesta dependência causal, inclusive a própria dependência causal. Na Figura B.9 este parâmetro é expresso com o valor "*used*#subtype#operator1", significando uma anotação para a dependência causal que está sendo instrumentalizada com subtype como valor de annotation property e operator1 como valor de annotation value, segundo especificação do formato adequado para o preenchimento deste parâmetro descrito na seção anterior.
- Performer: deve ser preenchido com o nome da pessoa ou ferramenta que realiza a tarefa, tal como um cientista ou uma ferramenta de workflow. Na Figura B.9, este parâmetro é preenchido com o valor AddFunction.pPerformerName, que é uma referência para o nome do performer da tarefa AddFunction descrita no workflow, conforme padronização definida na seção anterior.

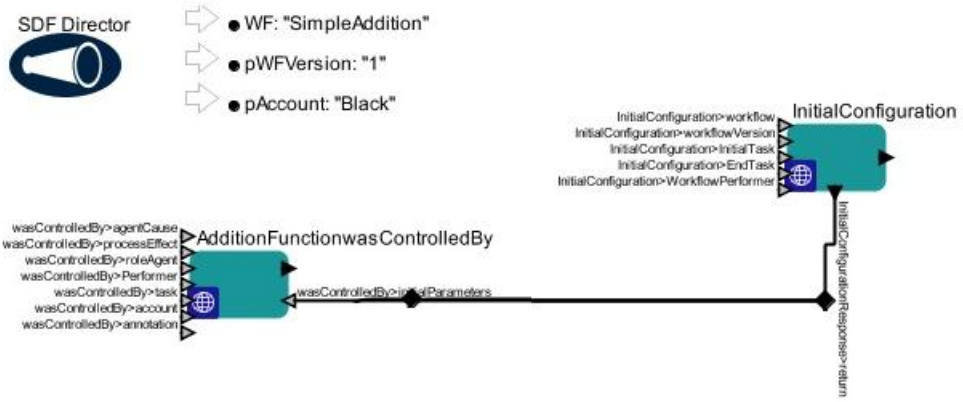


Figura B.10. Recorte da instrumentalização do workflowSimpleAddition focando na instância de serviço Web do SciProvMiner com o método *wasGeneratedBy* denominado AdditionFunction*wasControlledBy*

Por fim, para instrumentalizar uma dependência causal *wasTriggeredBy* segundo o modelo OPM, deve-se adicionar uma instância do serviço Web do SciProvMiner no workflow, selecionar o método *wasTriggeredBy* e preencher os parâmetros solicitados por ele. Os parâmetros solicitados para este método estão ilustrados na Figura B.11, sendo:

- processCause: deve ser preenchido com os dados do processo P que é a causa da dependência causal *wasTriggeredBy*. Na Figura B.11, este parâmetro é preenchido com o valor AddFunction.pProcess, que é uma referência para o nome do processo associado à tarefa AddFunction do workflow, conforme padronização descrita na seção anterior.
- processEffect: deve ser preenchido com os dados do processo P que é o efeito da dependência causal *wasTriggeredBy*. Na Figura B.11, este parâmetro é preenchido com o valor AbsoluteFunction.pProcess, que é uma referência para o nome do processo associado à tarefa AbsoluteFunction do workflow, conforme padronização descrita na seção anterior.
- taskCause: deve ser definida a tarefa ou atividade T do workflow ao qual o Processo P está relacionado. Na Figura B.11, este parâmetro é preenchido com o valor AddFunction.pTaskName, que é uma referência para o nome da tarefa descrita no workflow, conforme padronização definida na seção anterior.
- taskEffect: deve ser definida a tarefa ou atividade T do workflow ao qual o Processo P está relacionado. Na Figura B.11, este parâmetro é preenchido com o valor AbsoluteFunction.pTaskName, que é uma referência para o nome da tarefa descrita no workflow, conforme padronização definida na seção anterior.

- initialParameters: neste parâmetro deve ser informado o WorkflowId e o OPMGraphId, no formato “workflowId#OPMGraphId” retornado pela instância do serviço Web configurado com o método *InitialConfiguration* para aquele workflow. Neste parâmetro é necessário apenas uma referência ao valor retornado pela instância do serviço Web configurado com o método *InitialConfiguration*.
- Account: neste parâmetro é definido o Account ao qual essa dependência causal, juntamente com entidades do modelo OPM relacionados nesta dependência, fazem parte. Na Figura B.11 este parâmetro é preenchido com o valor pAccount que é uma referência para o parâmetro do workflow denominado pAccount com o valor “Black”, que pode ser visualizado na Figura 3.10.
- Annotation: anotações segundo o modelo OPM das entidades envolvidas nesta dependência causal, inclusive a própria dependência causal. Na Figura B.11 este parâmetro é expresso sem valor, significando que o usuário não deseja registrar nenhuma anotação das entidades nem da dependência causal modelada nesta instância do serviço Web.
- performerCause: deve ser preenchido com o nome da pessoa ou ferramenta que realiza a tarefa que é a causa da dependência causal modelada nesta instância do serviço Web instrumentalizado, tal como um cientista ou uma ferramenta de workflow. Na Figura B.11, este parâmetro é preenchido com o valor AddFunction.pPerformerName, que é uma referência para o nome do performer da tarefa AddFunction descrita no workflow, conforme padronização definida na seção anterior.
- performerEffect: deve ser preenchido com o nome da pessoa ou ferramenta que realiza a tarefa que é o efeito da dependência causal modelada nesta instância do serviço Web instrumentalizado, tal como um cientista ou uma ferramenta de workflow. Na Figura B.11, este parâmetro é preenchido com o valor AbsoluteFunction.pPerformerName, que é uma referência para o nome do performer da tarefa AddFunction descrita no workflow, conforme padronização definida na seção anterior.

Se o usuário quiser diminuir o trabalho de instrumentalização do workflow e ele já tiver instrumentalizado as tarefas envolvidas na dependência causal *wasTriggeredBy* em outras dependências causais, ele pode preencher os parâmetros taskCause, taskEffect, performerCause, performerEffect com a string “”, representando campo vazio, conforme apresentado na Figura B.12.

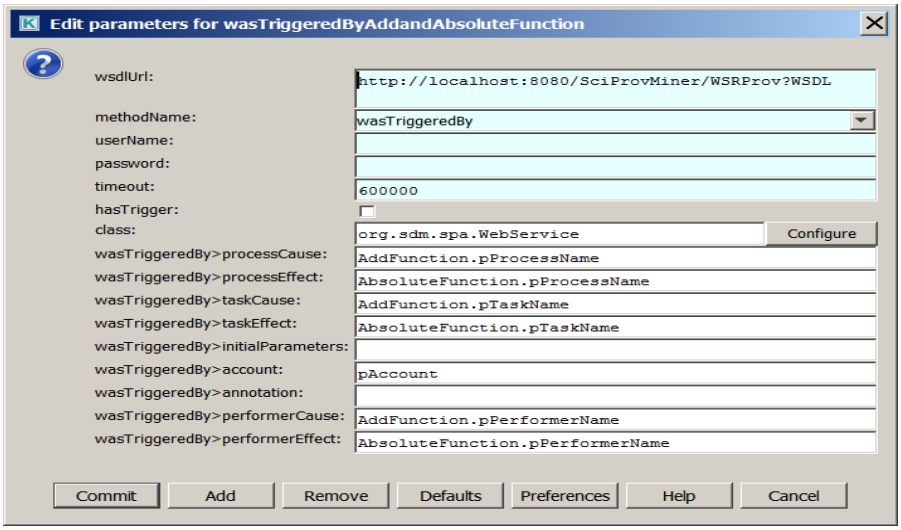


Figura B.11. Parâmetros do Método *wasTriggeredBy*

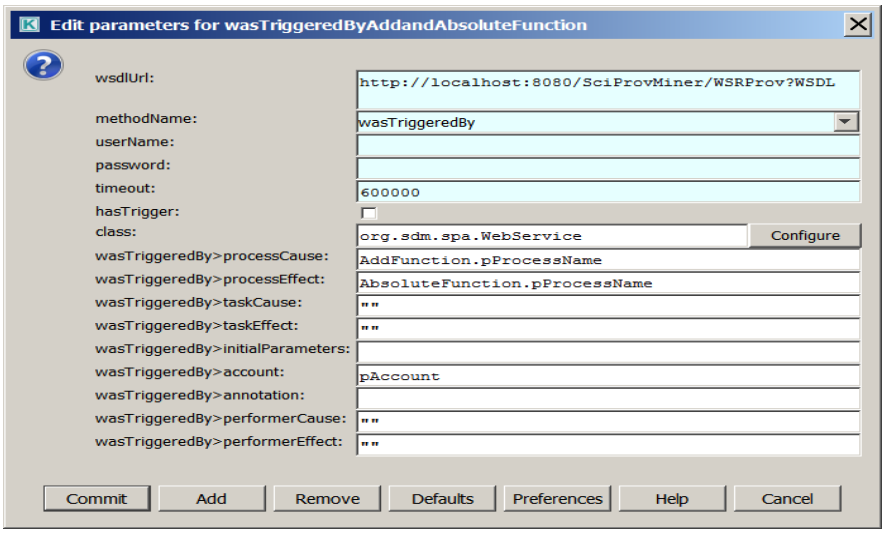


Figura B.12. Parâmetros do Método *wasTriggeredBy* com os parâmetros taskCause, taskEffect, PerformerCause e performerEffect preenchidos com o valor vazio simbolizado por abre e fecha aspas duplas(“”).

O SciProvMiner permite também a captura da proveniência de workflows que possuem laços de repetição. Para que isso seja possível, foi adicionado o atributo *Fire* as tabelas *Process*, *Used*, *WasDerivedFrom*, *WasGeneratedBy*, *WasGeneratedBy e WasTriggeredBy*. Este atributo é responsável por guardar a qual iteração do workflow a entidade ou a dependência causal está relacionada. O algoritmo implementado para trabalhar com workflows com laços de repetição se comporta da seguinte maneira:

1. No caso da captura de dados das dependências causais *used* e *wasGeneratedBy*, que usam um artefato relacionado com um processo, o algoritmo executa os seguintes passos:

   a. Verifica na dependência causal D para a qual a proveniência está sendo captura qual é a maior Fire desta dependência causal que relaciona o processo P com um artefato de mesmo label A que o artefato que está sendo relacionado com o processo P.
   b. Caso existam relacionamentos prévios, significa que a dependência causal do processo P com o artefato de label A já foi capturada anteriormente, e o algoritmo retornará o *currentFire* como sendo o maior Fire encontrado nesta consulta adicionado de uma unidade.
   c. Se não existirem relacionamentos prévios, o retorno desse algoritmo será 0.

2. No caso da dependência *causal wasControlledBy*, o algoritmo é executado da mesma maneira, apenas substituindo o Artefato de label A, pelo agente de AgentValue AV.

3. No caso da dependência causal *wasDerivedFrom*, o algoritmo também é executado da mesma maneira, apenas substituindo o processo pelos dados do segundo artefato de label A2, e fazendo os devidos relacionamentos entre as entidades.

4. No caso da dependência causal *wasTriggeredBy*, o algoritmo também é executado da mesma maneira, considerando, porém que a dependência causal envolve dois processos, e que ambos devem ser passados na consulta para a averiguação da existência de um relacionamento prévio entre esses processos na dependência causal *wasTriggeredBy*.

APÊNDICE C - CLASSES E RESTRIÇÕES DA ONTOLOGIA OPMO

As classes da ontologia OPMO original são apresentadas na Tabela C.1, bem como as restrições adicionadas a elas.

Tabela C.1: Classes da Ontologia OPMO original

Nome	Restrições	Annotations
Entity **Named Class**		The class of all constituents of an OPM graph.
AValue **Subclass of Entity** **Named Class**	content some Literal encoding some anyURI	The serial representation of an artifact Value
Account **Subclass of Entity** **Named Class**		The class representing an OPM Account.
Annotable **Subclass of Entity** **Defined Class**	Equivalent To Account or Annotation or Edge or Node or OPMGraph or Role	The set of OPM entities that can be annotated.
Annotation **Subclass of Entity** **Named Class**		OPM class *used* to annotate Annotable entities.
Edge **Subclass of Entity** **Named Class**		A (causal) relationship is represented by an arc and denotes the presence of a dependency between the source of the arc (the effect) and the destination of the arc (the cause).
Used **Subclass of Edge** **Named Class**	causeUsed only *Artifact* causeUsed some *Artifact* effectUsed only *Process* effectUsed some *Process* role some Role Disjoint *WasGeneratedBy*, *WasTriggeredBy*, *WasGeneratedBy*, *WasDerivedFrom*	A *used* edge from process to an artifact is a relationship intended to indicate that the process required the availability of the artifact to be able to complete its execution. When several artifacts are connected to a same process by multiple *used* edges, all of them were required for the process to complete. *Used* is a class that encompasses all the properties defined by OPM for this kind of edge. It is a reification of the opmv:*used* property.
WasGeneratedBy	causeWasControlledBy only *Agent* causeWasControlledBy some *Agent* effectWasControlledBy only *Process*	An edge *wasControlledBy* from a process P to an agent Ag is a dependency that indicates that the start and end of process P

Sub class of Edge **Named Class**	effectWasControlledBy some *Process* role some Role Disjoint *WasGeneratedBy*, *WasTriggeredBy*, *Used*, *WasDerivedFrom*	*wasControlledBy* agent Ag. *WasGeneratedBy* is a class that encompasses all the properties defined by OPM for this kind of edge. It is a reification of the opmv:*wasGeneratedBy* property.
WasDerivedFrom **Subclass of Edge** **Named Class**	cause*WasDerivedFrom* only *Artifact* cause*WasDerivedFrom* some *Artifact* effect*WasDerivedFrom* only *Artifact* effect*WasDerivedFrom* some *Artifact* Disjoint *WasGeneratedBy*, *WasTriggeredBy*, *Used*, *WasGeneratedBy*	An edge *wasDerivedFrom* from artifact A2 to artifact A1 is a relationship that indicates that artifact A1 needs to have been generated for A2 to be generated. The piece of state associated with A2 is dependent on the presence of A1 or on the piece of state associated with A1 . *WasDerivedFrom* is a class that encompasses all the properties defined by OPM for this kind of edge. It is a reification of the opmv:*wasDerivedFrom* property.
WasGeneratedBy **Subclass of Edge** **Named Class**	causeWasGeneratedBy only *Process* causeWasGeneratedBy some *Process* effectWasGeneratedBy only *Artifact* effectWasGeneratedBy some *Artifact* role some Role Disjoint *WasGeneratedBy*, *WasTriggeredBy*, *Used*, *WasDerivedFrom*	A *wasGeneratedBy* edge from an artifact to a process is a relationship intended to mean that the process was required to initiate its execution for the artifact to have been generated. When several artifacts are connected to a same process by multiple *wasGeneratedBy* edges, the process had to have begun, for all of them to be generated. *WasGeneratedBy* is a class that encompasses all the properties defined by OPM for this kind of edge. It is a reification of the opmv:*wasGeneratedBy* property.
WasTriggeredBy **Subclass of Edge** **Named Class**	cause*WasTriggeredBy* only *Process* cause*WasTriggeredBy* some *Process* effect*WasTriggeredBy* only *Process* effect*WasTriggeredBy* some *Process* Disjoint *WasGeneratedBy*, *WasGeneratedBy*, *Used*, *WasDerivedFrom*	An edge *wasTriggeredBy* from a process P2 to a process P1 is a causal dependency that indicates that the start of process P1 was required for P2 to be able to complete. *WasTriggeredBy* is a class that encompasses all the properties defined by OPM for this kind of edge. It is a reification of the opmv:*wasTriggeredBy* property
EventEdge **Subclass of Entity** **Defined Class**	Equivalent to *Used* or *WasGeneratedBy*	An EventEdge denotes an Edge associated with a time instant.
Node **Subclass of Entity**		Node is the class of nodes in an OPM graph. Nodes can be a source or effect of edges.
Agent **Subclass of Node**	Equivalent Class *Agent* Disjoint *Agent*, *Artifact*	comment "*Agent* is a contextual entity acting as a catalyst of a process, enabling, facilitating, controlling, or affecting its execution. "
Artifact **Subclass of Node**	Disjoint *Agent*, *Process*	*Artifact* is a general concept that represents immutable piece of state, which may have a physical embodiment in a physical object, or a digital representation in a computer system.

Process **Subclass of Node** **Named Class**	Disjoint *Agent*, *Artifact*	*Process* refers to an action or series of actions performed on or ca*used* by artifacts, and resulting in new artifacts.
OPMGraph **Subclass of Entity** **Named Class**		The class of all OPM graphs.
OTime **Subclass of Entity** **Named Class**		Observed time.
Property **Subclass of Entity** **Named Class**	value only Literal value some Literal	Building block allowing for the construction of annotations. It consists of key-value pair.
Role **Subclass of Entity** **Named Class**	value only string	A role designates an artifact's or agent's function in a process. Roles are constituents of *used*, *wasGeneratedBy*, and *wasControlledBy* edges, aimed at distinguishing the nature of the dependency when multiple such edges are connected to a same process.

As propriedades da ontologia OPMO original estão representadas na Tabela C.2, e modelam os conceitos expressos no modelo OPM.

Tabela C.2:*Object properties* da ontologia OPMO original

Nome	***Domain***	***Range***	**Comment**
Account	Annotation or Edge or Node	Account	Object Property to express the member of an OPM entity to some Account.
Annotation	Annotable	Annotation	Object property to associate an Annotable entity and an Annotation
Avalue	*Artifact*	Avalue	Denotes a serialization of an application value associated with an *Artifact*. Such serialization should have a type (expressed in a type system suitable for the serialization). Serialization technologies include XML, JSON, and ntriples.
Cause (Functional)	Edge	Node	The cause of an Edge.
cause→causeUsed	*Used*	*Artifact*	The cause of a *Used* edge.
cause→ cause*wasControlledBy*	*WasGeneratedBy*	*Agent*	The cause of a *WasGeneratedBy* edge.
cause→ cause*wasDerivedFrom*	*WasDerivedFrom*	*Artifact*	The cause of a *WasDerivedFrom* edge.
cause→ causewasGeneratedBy	*WasGeneratedBy*	*Process*	The cause of a *WasGeneratedBy* edge.

cause→ cause*wasTriggeredBy*	*WasTriggeredBy*	*Process*	The cause of a *WasTriggeredBy* edge.
effect (Functional)	Edge	Node	The effect of an Edge.
effect→effectUsed	*Used*	*Process*	The effect of a *Used* edge.
effect→ effect*wasControlledBy*	*WasGeneratedBy*	*Artifact*	The effect of a *WasGeneratedBy* edge.
effect→ effect*wasDerivedFrom*	*WasDerivedFrom*	*Artifact*	The effect of a *WasDerivedFrom* edge.
effect→ effectwasGeneratedBy	*WasGeneratedBy*	*Artifact*	The effect of a *WasGeneratedBy* edge.
effect→ effect*wasTriggeredBy*	*WasTriggeredBy*	*Process*	The effect of a *WasTriggeredBy* edge.
effectInverse (Inverse Functional)	Node	Edge	Convenience class introduced to describe the inverse of an effect. It is *used* to express*property chain*s.
effectInverse→effectUsed	*Process*	*Used*	The cause of a *Process* by means of a *Used* edge.
effectInverse→ effect*wasControlledBy*	*Process*	*WasGeneratedBy*	The cause of a *Process* by means of a *WasGeneratedBy* edge.
effectInverse→ effect*wasDerivedFrom*	*Artifact*	*WasDerivedFrom*	The cause of an *Artifact* by means of a *WasDerivedFrom* edge.
effectInverse→ effectwasGeneratedBy	*Artifact*	*WasGeneratedBy*	The cause of an *Artifact* by means of a *WasGeneratedBy* edge.
effectInverse→ effect*wasTriggeredBy*	*Process*	*WasTriggeredBy*	The cause of a *Process* by means of a *WasTriggeredBy* edge.
hasConstituent	OPMGraph		(Abstract) Property that denotes the constituency relationship between an OPM graph and one of its constituent entity, meaning that the object of this property is a constituent of the subject.
hasConstituent→ hasAccount	OPMGraph	Account	Property that denotes the constituency relationship between an OPM graph and an account, meaning that the object of this property is an account of the subject.
hasConstituent→ hasAgent	OPMGraph	*Agent*	Property that denotes the constituency relationship between an OPM graph and an agent, meaning that the object of this property is an agent of the subject.
hasConstituent→ hasArtifact	OPMGraph	*Artifact*	Property that denotes the constituency relationship between an OPM graph and an artifact, meaning that the object of this property is an artifact of the subject.
hasConstituent→ hasDependency	OPMGraph	Edge	Property that denotes the constituency relationship between an OPM graph and an edge, meaning that the object of this property is an edge of the subject.
hasConstituent→ hasProcess	OPMGraph	*Process*	Property that denotes the constituency relationship between an OPM graph and a process, meaning that the object of this

			property is a process of the subject.
Key	Property		The key of a Property.
Property	Annotation	Property	Object Property that associates an Annotation instance with a (set of) Propery(ies).
Role	*Used*or *WasGeneratedBy* or *WasGeneratedBy*	Role	The role of an edge.
startTime	*WasGeneratedBy*	OTime	The time at which the agent began controlling a process.
endTime	*WasGeneratedBy*	OTime	The time at which the agent ended controlling a process.
Time	EventEdge	OTime	A piece of timing information associated with an EventEdge.
WasGeneratedBy	*Process*	*Agent*	*wasGeneratedBy* is an abstract property to express that a process was controlled an agent.