Uma Estratégia de Versionamento de Workflows Científicos em Granularidade Fina

Bruno Costa[1], Eduardo Ogasawara[2], Leonardo Murta[1], Marta Mattoso[2]

[1] Instituto de Computação – UFF
Rua Passo da Pátria 156, Niterói, RJ, 24210-240

[2] Programa de Engenharia de Sistemas e Computação – COPPE / UFRJ
Caixa Postal 68.511, Rio de Janeiro, RJ, 21945-970

{brccosta,leomurta}@ic.uff.br, {ogasawara,marta}@cos.ufrj.br

Abstract. *The use of scientific workflow management systems (SWfMS) has become a reality in various scientific experiments. They support the controlled enactment of sequences of scientific activities, called scientific workflows, responsible for shaping the flow of data through programs. However, workflows are evolving entities and SWfMS should provide support for controlling this evolution. This paper presents a strategy for fine grained versioning of scientific workflows, which is based on solid configuration management (CM) principles such as separation of versioning and product spaces and bubble-up method, also used by popular CM systems such as Subversion.*

Resumo. *O uso de sistemas de gerenciamento de workflows científicos (SGWfC) se tornou uma realidade em diversos experimentos científicos. Eles apóiam a execução controlada de seqüências de atividades científicas, denominadas workflows científicos, responsáveis por modelar o fluxo de dados através de programas. Contudo, workflows são entidades em constante evolução e SGWfC devem fornecer apoio ao controle dessa evolução. Esse artigo apresenta uma estratégia de versionamento de workflows científicos em granularidade fina baseada em princípios sólidos de gerência de configuração (GC) como a separação dos espaços de versão e de produto e o método bubble-up, também usados por sistemas de GC populares, como o Subversion.*

1. Introdução

Atualmente, a ciência da computação tem permitido um avanço considerável em diversas áreas do conhecimento. O processamento e a execução automatizada de procedimentos anteriormente desenvolvidos manualmente oferece oportunidades para promover a otimização e eficiência na execução de diversos processos. Dentre estes processos se encontra o experimento científico. A ciência vem evoluindo com a utilização de procedimentos *in-vitro* e *in-vivo*, porém, o avanço descrito permite que sejam possíveis novas formas de experimento, dentre elas o *in-silico* (Taylor et al. 2007), onde os objetos de análise dos experimentos são usualmente processados por simulações computacionais e analisados via técnicas de visualização (Deelman et al. 2009). Neste contexto surge o termo e-Ciência (do inglês, e-Science), que caracteriza o desenvolvimento de ciência em larga escala utilizando infra-estrutura computacional correspondente (Mattoso et al. 2009).

Segundo Mattoso et al. (2009), os passos do ciclo de vida dos experimentos *in-silico* podem ser agrupados em três etapas, a saber, composição, execução e análise, sendo respectivamente, a fase responsável pela elaboração dos workflows que devem fazer parte do experimento; execução dos workflows propriamente ditos; e avaliação dos resultados experimentais obtidos das execuções dos workflows. A concepção, que é a principal etapa da composição de um experimento, engloba as atividades de especificação e modelagem, bem como a definição do empacotamento do experimento. Nesta fase, é construído o encadeamento de atividades para a experimentação.

Para o gerenciamento da execução das atividades de um workflow, os SGWfC passaram a ser utilizados (Deelman et al. 2009). O objetivo dos SGWfC é propiciar a orquestração de vários processamentos computacionais, fazendo-se valer de processamento paralelo e distribuído, bancos de dados, inteligência artificial, dentre outros, construindo, assim, um arcabouço para experimentação através de simulação (Taylor et al. 2007). Contudo, os SGWfC atuais atuam sobre workflows concretos, que normalmente passam por um longo ciclo de tentativa e erro, implicando em diversas modificações, até que o experimento atinja um resultado aceitável. Esse cenário se complica devido a essas modificações nem sempre serem feitas de forma seqüencial. Em diversas situações, pode ser necessário recuperar versões passadas do workflow para tentar alternativas de modificações. Assim, o controle sobre a evolução das diferentes versões dos workflows em granularidade fina, para permitir a identificação precisa de quais partes do workflow forma modificadas em cada versão, se torna um requisito fundamental para SGWfC (Ogasawara et al. 2009b).

SGWfC como Taverna (Hull et al. 2006), Kepler (Altintas et al. 2004) apresentam pouco ou nenhum suporte ao versionamento de workflows científicos. Dentre os SGWfC mais utilizados, o VisTrails (Callahan et al. 2006) é o que apresenta algum suporte a versionamento. Contudo, diversos princípios de gerência de configuração ainda não foram incorporados, como, por exemplo, a separação entre os espaços de versionamento e de produto, abrindo oportunidades para pesquisas adicionais no tema.

O presente trabalho apresenta uma estratégia de versionamento para workflows científicos, de maneira a contribuir com sua gerência e evolução, promovendo eficiência e controle para o cientista no momento da composição do experimento. Essa estratégia está fundamentada em princípios sólidos de gerência de configuração, como separação entre espaços de versionamento e de produto e método *bubble-up* (Subversion 2009) de versionamento. A estratégia proposta foi implementada na ferramenta para gerência de composição de workflows científicos GExpLine (http://gexp.nacad.ufrj.br).

Este artigo está organizado em quatro seções, além desta introdução. A Seção 2 faz a introdução ao conceito de concepção de workflows científicos. A Seção 3 apresenta brevemente a gerência de configuração aplicada a workflows científicos. A Seção 4 detalha a estratégia de gerência de configuração proposta, apresentando um modelo que separa o espaço de versão do espaço de produto e um algoritmo baseado no método *bubble-up* para versionamento em granularidade fina. Finalmente, a Seção 5 conclui o artigo com algumas perspectivas futuras.

2. Concepção de Workflows Científicos

Uma experiência científica caracteriza-se pela montagem de uma estratégia concreta a partir da qual se organizam diversas ações observáveis direta ou indiretamente, de for-

ma a provar a plausibilidade de uma dada hipótese ou de forma a estabelecer relações de causa e efeito entre fenômenos (Jarrard 2001). Esta montagem é feita na fase de concepção do workflow científico, onde as atividades necessárias para atingir o suposto objetivo do experimento são mapeadas, assim como as respectivas entradas e saídas de dados. Na Figura 1 é mostrado um workflow científico criado no SGWfC Kepler.

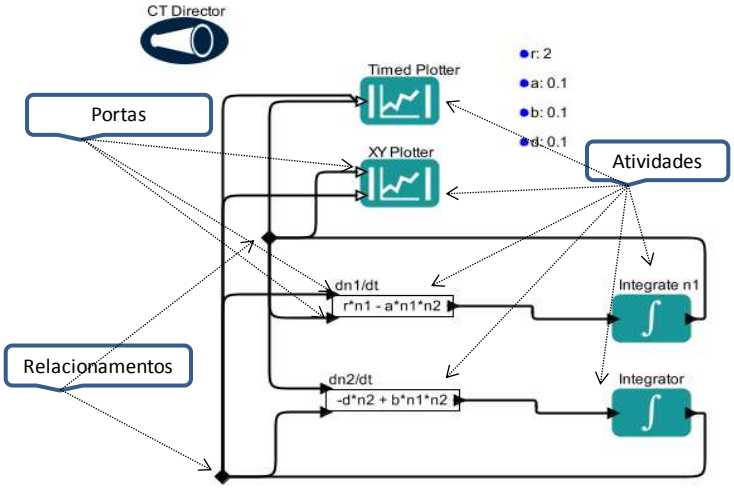

Figura 1: Workflow desenvolvido no Kepler

Baseados em grafos direcionados, os workflows científicos mapeiam as atividades a serem executadas na experiência científica. De forma simplificada, é possível verificar que a saída de uma atividade torna-se a entrada para a outra atividade. Isto é feito de forma sistemática até que a resposta ou o objetivo a ser alcançado seja atingido.

3. Gerência de Configuração de Workflows Científicos

A Gerência de Configuração de Software (GCS) é uma disciplina responsável pelo controle da evolução de sistemas de software (Estublier 2000), e engloba técnicas de versionamento amplamente utilizadas em projetos de grande porte de desenvolvimento de software. Para que tais técnicas sejam aplicadas de forma eficiente no contexto de e-Science, as seguintes funcionalidades devem estar presentes em um sistema de apoio à composição de experimentos (Ogasawara et al. 2009b):

- O sistema deve fornecer controle de acesso e controle de concorrência sobre o repositório de workflows, de modo que seja possível armazenar workflows e registrar a versão estável (i.e., de produção) e versões em desenvolvimento;
- O sistema deve manter um registro das versões de atividades que estão sendo utilizadas por cada composição de workflows; e
- O sistema deve fornecer um espaço de trabalho (*workspace*) para apoiar a edição de workflows. O espaço de trabalho precisa também apoiar a publicação de versões do workflow no repositório.

3.1 Itens de configuração

Um item de configuração corresponde a um artefato de software ou hardware que é passível de gerência de configuração e tratado como um elemento único. No que tange a workflows científicos, o workflow como um todo pode ser considerado um item de configuração, propiciando versionamento em granularidade grossa, ou cada parte interna do

workflow pode ser considerada um item de configuração, propiciando versionamento em granularidade fina.

3.2 Repositório

O repositório é o local onde são armazenados os itens de configuração **(Dart 1991)**. O repositório pode fazer uso de diferentes tecnologias para persistir os itens de configuração. Na literatura, as tecnologias mais adotadas são sistemas de arquivos do sistema operacional, bancos de dados relacionais e bancos de dados orientados a objetos. Independentemente da estratégia adotada, num primeiro momento, após a importação do workflow, todos os itens de configuração são armazenados na versão 1. Para a edição de um workflow, é necessário que seja feito o *check-out* (ação de copiar todos os itens de configuração, a fim de que sejam modificados). Após a modificação, é feito o *check-in* quando as modificações são gravadas novamente no repositório.

3.3 Estratégias de Versionamento

De acordo com Conradi et al. (1998), as duas estratégias clássicas de versionamento externo são versionamento por componente e versionamento total. Contudo, devido à facilidade de implementação, a primeira estratégia adotada por SGWfC tende a ser versionamento por componente. Nessa estratégia, a partir do *check-out* e um posterior *check-in*, é atribuída nova versão para todos os itens de configuração. Ou seja, o workflow como um todo passa para uma nova versão. Utilizando um gráfico AND/OR (Conradi e Westfechtel 1998), a Figura 2 mostra como se comporta um SGWfC utilizando a estratégia de versionamento por componente onde uma nova versão é atribuída à todas as atividades quando algo é modificado no Workflow (nome de uma atividade, por exemplo).

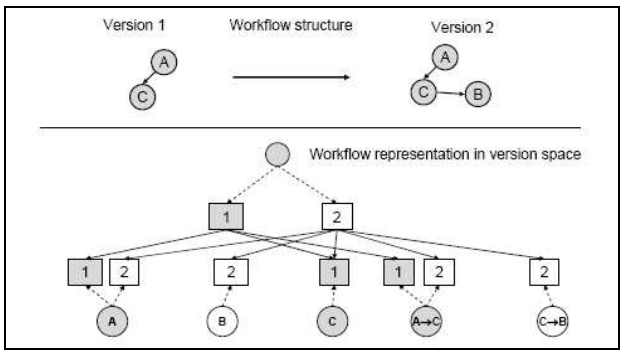

Figura 2: Gráfico AND / OR do modelo de versionamento global

Essa estratégia de versionamento apresenta ineficiência, pois neste caso o versionamento é baseado apenas no *check-out/check-in*, e não nas modificações concretas que foram efetuadas sobre os itens de configuração. O versionamento total, por sua vez, consiste em atribuir uma nova versão somente para os elementos que foram de fato modificados. Com essa estratégia, juntamente com a aplicação do método *bubble-up*, é possível propagar a modificação somente para a região pertinente do workflow, como detalhado na seção seguinte.

4. Uma Estratégia de Versionamento para SGWfC

Uma estratégia de versionamento para SGWfC deve levar em consideração tanto as características do problema quanto as tecnologias de gerência de configuração disponí-

veis. Esta seção detalhará aspectos referentes à separação do espaço de versionamento e de produto e o algoritmo utilizado para permitir o versionamento total em workflows.

4.1 Separação entre os Espaços de Versionamento e de Produto

Um modelo de versão define os objetos a serem versionados, identificação da versão e organização, bem como as operações para recuperação de versões existentes e construção de novas versões. Para a estratégia de versionamento proposta, é desnecessário explicitar a organização estrutural do workflow com seus elementos de controle, pois a versão é atribuída segundo a modificação do item de configuração. Assim, independente da estrutura, a estratégia se baseia nos itens modificados. Na Figura 3 é mostrado o modelo adotado na estratégia proposta.

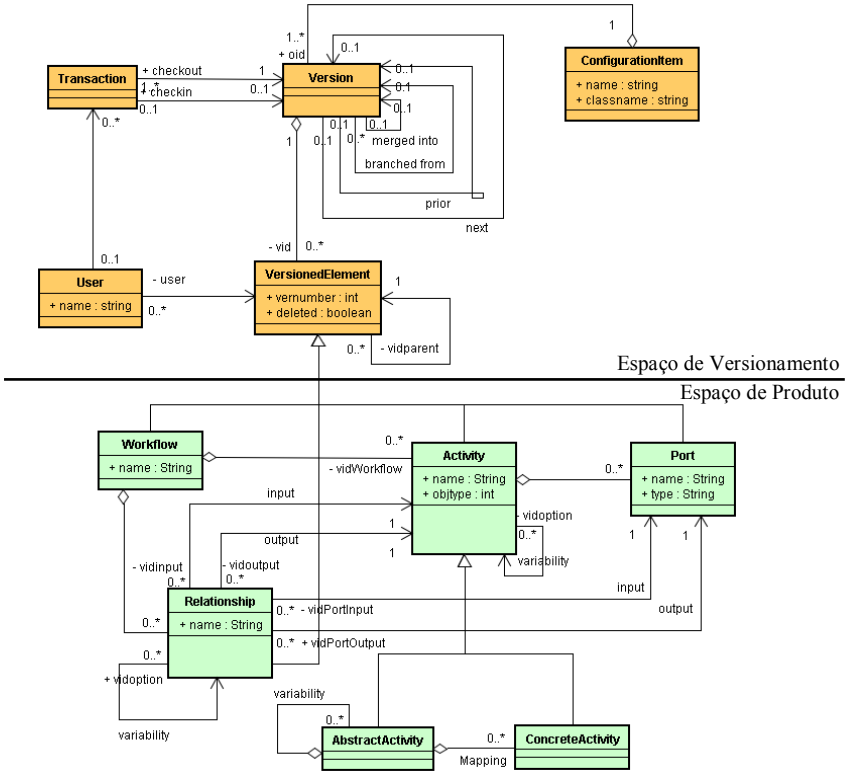

Figura 3: Evolução do Modelo de Versionamento de Workflows de (Ogasawara et al. 2009b) apoiando diferentes níveis de abstração

As classes da parte superior da Figura 3 correspondem ao espaço de versionamento, enquanto que as classes da parte inferior da Figura 3 representam o espaço de produto. No espaço de produto, um workflow (classe *Workflow*) é composto por atividades (classe *Activity*) e relacionamentos (classe *Relationship*). Uma atividade num workflow comporta-se como um componente, possuindo portas de entrada e saída (classe *Porta*). O relacionamento entre atividades é uma aresta diretamente conectada, estabelecendo uma relação de dependência entre atividades e também definindo o encadeamento do workflow. Além disto, foram criadas especializações de atividades para representar as atividades abstratas (classe *AbstractActivity*) e concretas (*ConcreteActivity*) para apoiar a composição de workflows científicos em diferentes níveis de abstração (Ogasawara et al. 2009a).

No espaço de versionamento, cada item de configuração (classe *ConfigurationItem*) é composto por versões (classe *Version*). Cada versão tem relacionamentos com a

próxima versão e com a versão anterior, que pode ser nula para a última e a primeira versão de um item de configuração, respectivamente. Um atributo específico diferencia versões que foram apagadas por usuários. Versões são consultadas ou criadas por transações (classe *Transaction*) feitas por usuários (classe *User*). Finalmente, versões têm relações com elementos versionados (classe *VersionedElement*). Essa relação é a ponte entre o espaço de versionamento e o espaço de produto.

4.2 Estratégia de Versionamento Total

Conforme citado anteriormente, o versionamento por componente se apresenta ineficiente, pois atribui nova versão conforme o *check-out* e *check-in* e não conforme modificações. Desta forma, o versionamento total traz características positivas para o e-Science, onde o acompanhamento detalhado da evolução de um workflow, em granularidade fina, se faz necessário, como discutido na seção 1.

Para exemplificar essa estratégia de versionamento, como primeira ação, um workflow é criado no sistema Kepler. Após a importação do workflow, são identificados os itens de configuração em granularidade fina (i.e., workflow, atividades, relacionamentos e portas) e atribuída uma identificação única (OID) a cada um, conforme o modelo de versão. Além da identificação única, é atribuída a versão 1 a cada item. A Figura 4.a exibe essa situação, onde é possível verificar que existe uma relação pai - filho, onde, conforme as características de árvores, o pai de um nó N é o no imediatamente acima, de altura N-1.

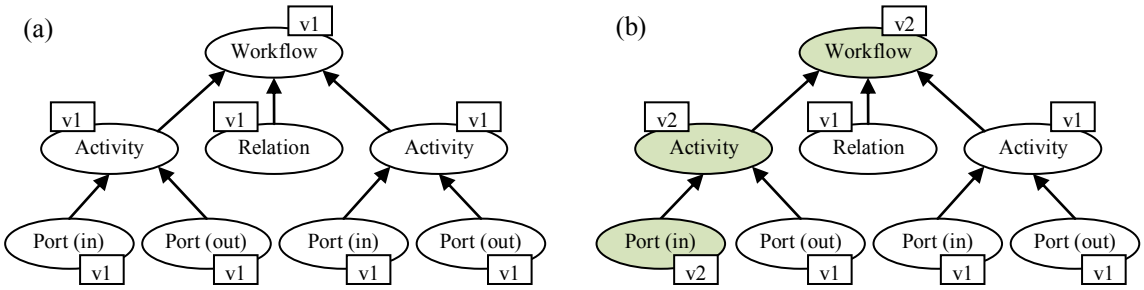

Figura 4: Itens de configuração (a) e Atribuição de versão para o elemento Port(in) e para seus pais (b)

Para a estratégia de versionamento proposta, foi desenvolvido o algoritmo apresentado na Figura 5. O método diff (linha 1) inicia verificando se os atributos primitivos das duas versões do elemento (a do repositório e a do *check-in*) são equivalentes (linha 2). Após essa verificação, é feita uma análise para verificar se os filhos da versão do *check-in* são novos ou se já existiam no repositório. Esta identificação é feita pela busca de OID *compara()* (linha 4). Se o elemento filho for novo (linha 5), o pai é considerado modificado. Caso o elemento filho já exista, o método é executado recursivamente (linha 8), e a informação de modificação dos filhos é propagada para o pai (método *bubble-up*). No caso em que algum filho foi removido o pai também é considerado modificado (linha 10). Ao final, caso o elemento não tenha sido modificado, permanece com a versão atual. Caso contrário, uma nova versão (que está sendo utilizada no *check-in* em questão) é atribuída ao elemento.

É importante notar que quando algum elemento é modificado, o seu pai deve ser modificado também, e isto deve se propagar até o elemento raiz, ou seja, o workflow. A Figura 4.b contém um exemplo. Nela, foi modificado um dos elementos *Port (in)*, e uma nova versão foi atribuída e propagada até o *Workflow*.

```
1  diff(VersionedElement repositorio, VersionedElement checkin) : boolean
2     modificado ← comparaAtributos(repositorio, checkin)
3     Para cada filho filhoCheckin de checkin
4        Busca filho filhoRepos de repositorio onde compara(filhoRepositorio, filhoCheckin) == true
5        Se (filhoRepositorio == null) // elemento novo
6           modificado ← true
7        Senão // elemento  já existente -> comparar recursivamente até não ter mais filho a explorar
8           modificado ← modificado OR diff(filhoRepositorio, filhoCheckin)
9           filhosProcessados  ← filhoRepositorio
10    Se (modificado OR (filhos de repositório ≠ filhosProcessados)) // modificado ou filho deletado
11       Cria uma nova versão para este VersionedElement checkin
12    Senão
13       Associa ao VersionedElement checkin a mesma versão de repositorio

14 compara(VersionedElement e1, VersionedElement e2) : boolean
15    retorna (e1.getConfigurationItem().getOID() == e2. getConfigurationItem().getOID())

16 comparaAtributos(VersionedElement e1, VersionedElement e2) : boolean
17    Retorna (e1.getname() == e2.getname()) AND ...
```

Figura 5: Algoritmo de diferença

A estratégia se mostrou eficiente se comparada ao versionamento por componente, pois as versões foram atribuídas apenas aos elementos que de fato foram modificados, juntamente com seus pais. Com a criação do método *comparaAtributos*(), é possível que sejam analisadas quaisquer modificações no elemento, retornando um valor verdadeiro caso alguma modificação tenha sito feita. Este método pode ser implementado concretamente utilizando reflexão ou via polimorfismo, onde cada classe fornece uma implementação específica para esse método, explicitando os atributos que devem ser utilizados para comparação.

Com o armazenamento dos Itens de Configuração conforme a estratégia proposta é possível a recuperação do workflow a partir do item de configuração e versão desejados juntamente com as informações de quem modificou e quando foi modificado. Além disso, sendo capaz de identificar unicamente os elementos que foram modificados, é possível também executar consultas mais elaboradas, como, por exemplo, utilizando técnicas provenientes da mineração de dados.

5. Conclusões

Este artigo apresentou de forma concreta como conceitos provindos de gerência de configuração podem ser usados para permitir o controle sobre a evolução de workflows científicos. Foram apresentadas duas estratégias de versionamento, a saber, o versionamento por componentes, que não leva em conta as modificações, mas apenas as rotinas de *check-out/check-in*, e o versionamento total. Este último se mostrou mais eficiente, pois controla de fato a evolução dos Itens de Configuração.

Ademais, a estratégia implementada possibilitará a comparação por similaridade, onde a comparação entre Itens de Configuração não será feita apenas entre OIDs, mas contemplando os atributos definidos por cada classe. Esta forma de configuração permite que workflows trabalhem com o conceito de linhas de experimento científico em sua evolução (Ogasawara et al. 2009a). Após o desenvolvimento da comparação por similaridade, será possível a implementação de um algoritmo que realize a junção (do inglês *merge*) entre workflows que venham de linhagens diferentes. Neste caso, não pode ser feita apenas uma comparação por OID, pois, ao analisar dois workflows de sistemas diferentes, os Itens de Configuração serão, conseqüentemente, diferentes.

A estratégia de versionamento aqui apresentada foi implementada na ferramenta GexpLine e será submetida a avaliações futuras com dados reais, em especial nos domínios de Bioinformática e Engenharia de Petróleo.

Referências

Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludascher, B., Mock, S., (2004), "Kepler: an extensible system for design and execution of scientific workflows". In: *Proceedings. 16th International Conference on Scientific and Statistical Database Management*, p. 423-424, Santorini, Greece.

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

Conradi, R., Westfechtel, B., (1998), "Version Models for Software Configuration Management", *ACM Computing Surveys*, v. 30, n. 2

Dart, S., (1991), "Concepts in configuration management systems". In: *Proceedings of the 3rd international workshop on Software configuration management*, p. 1-18, Trondheim, Norway.

Deelman, E., Gannon, D., Shields, M., Taylor, I., (2009), "Workflows and e-Science: An overview of workflow system features and capabilities", *Future Generation Computer Systems*, v. 25, n. 5, p. 528-540.

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

Hull, D., Wolstencroft, K., Stevens, R., Goble, C., Pocock, M. R., Li, P., Oinn, T., (2006), "Taverna: a tool for building and running workflows of services", *Nucleic Acids Research*, v. 34, n. Web Server issue, p. 729-732.

Jarrard, R. D., (2001), *Scientific Methods*. Online book, Url.: http://emotionalcompetency.com/sci/booktoc.html.

Mattoso, M., Werner, C., Travassos, G., Braganholo, V., Murta, L., Ogasawara, E., Oliveira, F., Martinho, W., (2009), "Desafios no Apoio à Composição de Experimentos Científicos em Larga Escala". In: *SEMISH - CSBC*, Bento Gonçalves, RS - Brasil.

Ogasawara, E., Paulino, C., Murta, L., Werner, C., Mattoso, M., (2009a), "Experiment Line: Software Reuse in Scientific Workflows". In: *Proceedings of the 21th international conference on Scientific and Statistical Database Management*, p. 264–272, New Orleans, LA.

Ogasawara, E., Rangel, P., Murta, L., Werner, C., Mattoso, M., (2009b), "Comparison and Versioning of Scientific Workflows". In: *Proceedings of the 2009 international workshop on Comparison and versioning of software models*, p. 25-30, Vancouver, Canada.

Subversion, (2009), *Subversion*, http://subversion.tigris.org/design.html.

Taylor, I. J., Deelman, E., Gannon, D. B., Shields, M., (Eds.) , (2007), *Workflows for e-Science: Scientific Workflows for Grids*. 1 ed. Springer.