Algoritmo para Geração Automática de Ações de Rollback em Sistemas de Gerenciamento de Mudanças em TI

Guilherme Sperb Machado[1], Weverton Luis da Costa Cordeiro[1], Alan Diego dos Santos[1], Cristiano Bonato Both[1], Luciano Paschoal Gaspary[1], Lisandro Zambenedetti Granville[1], Claudio Bartolini[2], Akhil Sahai[2], David Trastour[3], Katia Saikoski[4]

[1]Instituto de Informática, UFRGS - Porto Alegre, RS
{gsmachado, weverton.cordeiro, adsantos
cbboth, paschoal, granville}@inf.ufrgs.br

[2]HP Laboratories Palo Alto - [3]HP Laboratories Bristol - [4]HP Brazil R&D
{claudio.bartolini, akhil.sahai,
david.trastour, katia.saikoski}@hp.com

Abstract. *The current research on IT change management has been exploring several aspects of this new discipline, but it usually assumes that changes expressed in Request for Changes (RFC) documents will be successfully executed over the managed IT infrastructure. This assumption, however, is not realistic in actual IT systems because failures during the execution of changes do happen and cannot be ignored. In order to address this issue, this paper uses a model where change plan activities can be expressed as atomic transactions. Once change plan activities are marked as atomic, associated rollback actions must be present to avoid inconsistent states in case of system failures. Therefore, this paper focuses in the generation of these rollback actions, presenting the algorithm for such computation.*

Resumo. *As atuais pesquisas em gerência de mudança em um ambiente de TI (Tecnologia de Informação) têm explorado diferentes aspectos, porém normalmente assumindo que as mudanças expressas em documentos de Requisição de Mudanças (RFC - Request for Change) são sempre executadas com sucesso sobre uma determinada infra-estrutura. Esse cenário, muitas vezes, não reflete a realidade em sistemas de TI, pois falhas durante a execução de mudanças podem ocorrer e não devem ser simplesmente ignoradas. Para abordar esta questão, este artigo propõe uma solução onde atividades em um plano de mudança podem ser definidas como transações atômicas. Sempre que marcadas como atômico, ações de rollback (atreladas à atividades) serão disparadas quando ocorrerem erros de execução. Este artigo foca na geração de ações de* rollback *apresentando o algoritmo para a computação destas ações.*

1. Introdução

Atualmente, empresas e organizações de grande porte não podem oferecer serviços de qualidade sem empregar uma sofisticada infra-estrutura de TI para suportar seus negócios. Por sua vez, infra-estruturas de TI geralmente possuem complexidade de gerência considerável, trazendo assim custos elevados para sua manutenção. Neste contexto, adotar políticas de gerência racionais para infra-estruturas de TI torna-se um ponto crítico para

as organizações. Para alcançar uma gerência apropriada – e assim reduzir custos – o ITIL (*Information Technology Infrastructure Library*) [ITIL 2006] compilou um conjunto de processos e boas práticas que ajudam as organizações à manter suas infra-estruturas de maneira adequada.

Entre os processos propostos pelo ITIL, o gerenciamento de mudanças [IT Infrastructure Library 2000] é aquele que define como mudanças em TI devem ser planejadas, agendadas, implementadas e avaliadas. A importância da gerência de mudanças reside no fato de que as mudanças em uma infra-estrutura de TI devem ser executadas de forma que não levem o sistema gerenciado a um estado desconhecido ou inconsistente. Assim, mudanças em infra-estruturas de TI são expressas primeiramente em documentos intitulados requisições de mudança (*Request for Change* - RFC), que definem quais mudanças são necessárias, mas não especifica porém como elas devem ser executadas. A definição de uma RFC é o primeiro passo do processo que irá gerar um plano de mudança (*change plan*), que essencialmente é um *workflow* composto por atividades concretas e de um nível de abstração mais baixo. O papel do plano de mudança, uma vez executado, é o de levar o sistema gerenciado para um novo estado de execução consistente que reflita as mudanças solicitadas na RFC original.

Apesar de a área de gerenciamento de mudanças ser nova e ainda pouco explorada, alguns problemas em potencial já foram investigados [Keller et al. 2004] [Bartolini et al. 2006] [Rebouças et al. 2007]. Devido à complexidade intrínseca do assunto, as pesquisas desenvolvidas até o momento se basearam em algumas premissas que permitiram chegar a diversas conclusões importantes sobre vários aspectos em gerenciamento de TI. Uma destas premissas é a de que uma vez aprovadas, as atividades de uma RFC irão sempre ser implantadas com sucesso, levando a infra-estrutura de TI para o próximo estado consistente. Na verdade, essa suposição não reflete a realidade dos ambientes de TI, já que falhas durante a execução das mudanças efetivamente ocorrem, e assim não podem ser ignoradas.

Este artigo aborda a necessidade de tratar falhas durante a execução de um plano de mudança, evitando assim que a infra-estrutura gerenciada evolua para um estado inconsistente e desconhecido. Para atacar este problema, esta pesquisa propõe uma solução que garanta que depois da implantação de uma RFC, a infra-estrutura gerenciada evoluirá para um novo estado consistente, ou então retornará ao estado imediatamente anterior à RFC. Em outras palavras, objetiva-se que a implantação de uma RFC seja tratada como uma transação atômica. Em trabalhos passados, desenvolvidos por este grupo de pesquisa [Machado et al. 2008], foi apresentado um modelo para suportar *rollback* em planos de mudança, onde o conceito de atomicidade foi estendido a diferentes níveis de uma RFC, concedendo então uma maior flexibilidade na descrição de ações suscetíveis a *rollback*. Com este modelo que provê suporte a *rollback*, toda vez que uma falha ocorrer durante a execução de um plano de mudança, um procedimento de *rollback* será disparado de forma a reverter as mudanças já executadas e abortar o plano de mudança corrente.

Para fornecer suporte à transações atômicas no contexto de gerenciamento de TI, empregamos um conjunto de técnicas em um protótipo desenvolvido para avaliar nossa solução. Em particular, exploramos alguns mecanismos relacionados ao disparo de exceções definidos na *Business Process Execution Language* (BPEL) [OASIS Standards 2007]. No protótipo implementado, transações atômicas no nível de

RFC e planos de mudança associados, definidos por operadores do sistema, são traduzidos em construções BPEL por um algoritmo de mapeamento. Este algoritmo de mapeamento é de extrema importância no processo, pois adiciona construções BPEL que identificam falhas na execução, invocando então ações de *rollback*. As ações de *rollback* por sua vez, são geradas automaticamente por um algoritmo que leva em consideração a especificação de atomicidade dos planos de mudança. Portanto, este artigo está focado em apresentar detalhadamente o algoritmo de geração de ações de *rollback* e como ele se relaciona com outros componentes do sistema proposto. Tendo este cenário, experimentos foram explicitados para observar a funcionalidade do algoritmo implementado.

Este artigo está organizado como segue. Na Seção 2 descrevemos rapidamente a relação do suporte a *rollback* em sistemas computacionais. A solução proposta como um todo, com o detalhamento do algoritmo da geração de ações de *rollback*, é apresentado na Seção 3, enquanto que o protótipo implementado é descrito na Seção 4. Um estudo de caso e análise são apresentados na Seção 5. Por fim, encerramos este artigo na Seção 6, onde conclusões e trabalhos futuros são discutidos.

2. Trabalhos Relacionados

Suporte a *rollback* é um assunto complexo em diversas áreas da ciência da computação. Alguns aspectos de sistemas computacionais (e.g., comunicação tolerante a falhas, dependências entre componentes distribuídos e indisponibilidade de serviços) fazem com que tarefas executadas possam ou não serem executadas com sucesso. Um exemplo de tarefas que possuem esta característica é a ação de salvar estados consistentes de um sistema para que em um momento subseqüente, o sistema possa executar *rollback*. Porém, esta situação nem sempre é possível. Ainda assim, alguns mecanismos para suportar *rollback* já foram propostos, investigados e implementados. Nesta seção iremos relacionar trabalhos que inspiraram e motivaram o projeto da solução apresentada.

No nível de dispositivos, uma forma comum de implementar *rollback* é obter a configuração do dispositivo através da rede (*download*), implementar a nova configuração, e usar a configuração prévia caso a mudança torne o dispositivo instável. Uma evolução para esta solução pode ser vista em dispositivos que a configuração candidata de mudança pode ser armazenada dentro dos próprios dispositivos gerenciáveis, dispensando então a necessidade de um servidor de configuração externo. Recentemente, o protocolo NETCONF [Enns 2006], proposto pelo *Internet Engineering Task Force* (IETF), incorporou a noção de transações em uma tarefa de configuração, que evita que dispositivos gerenciáveis evoluam para estados desconhecidos.

Isoladamente, o processo de *rollback* em nível de dispositivos pode não ser suficiente para ambientes de TI complexos, pois geralmente diferentes dispositivos e serviços dependem de outros fatores para o seu funcionamento. Por exemplo, se a instalação de um novo servidor Web necessitar configurações adicionais no *gateway* de borda, e esta última ação de configuração falhar, não só a instalação/configuração do servidor precisa ser desfeita como também a ação sobre o *gateway*. Com isso, a utilização de *rollback* em nível de rede (acima do nível de dispositivos) se faz necessário. Em trabalhos passados, propusemos um sistema PBNM (*Policy-Based Network Management*) [Alves et al. 2006] onde falhas na implantação de políticas de QoS (*Quality of Service*) geram ações que levam os dispositivos de rede para um estado anteriormente conhecido, usando uma versão

adaptada do protocolo *two-phase commit*.

Andrzejak [Andrzejak et al. 2005] investigou como gerar automaticamente um *workflow* que pode se adaptar em reação a falhas em um sistema de TI. Porém, este último trabalho não apresenta muitos detalhes na geração de *workflows* em resposta a falhas parciais. Além disso, os autores reconhecem que a solução proposta possui algumas limitações como a complexidade existente relacionada entre o número de objetos e os operadores relacionados, custos elevados, e a especificação das ações.

Na área de gerenciamento de mudanças, até onde os autores deste artigo estão cientes, não existem trabalhos que abordem a questão do uso de *rollback* como um mecanismo para manter uma infra-estrutura de TI em um estado consistente. A importância desse tópico é ainda mais evidente se for observado, nos documentos do ITIL, as citações aos planos de contingência (*back-out*) como um requisito básico na área de gerenciamento de mudanças. Porém, atualmente esta funcionalidade não é encontrada em sistemas de gerenciamento. Nas próximas seções iremos introduzir a solução proposta com um detalhamento maior do algoritmo de geração de planos de *rollback*.

3. Solução para suporte e execução de *rollback*

Nesta seção apresentamos a solução para suporte a *rollback* em sistemas de gerenciamento de mudanças em TI. Primeiramente, iremos descrever uma arquitetura genérica de gerenciamento de TI, onde o suporte a *rollback* é introduzido. Em seguida explicitaremos como ações atômicas são descritas e como administradores/operadores podem agrupar atividades críticas em grupos atômicos. Por fim, apresentaremos a modelagem de classes da proposta, seguido do algoritmo de geração de ações de *rollback*.

3.1. Arquitetura de gerenciamento de mudanças e componentes de *rollback*

Apesar de não existir uma única arquitetura amplamente utilizada para gerenciamento de mudanças em TI, é possível identificar um conjunto básico de componentes funcionais que, agrupados, formam uma arquitetura genérica. No passado [Machado et al. 2008], propomos uma arquitetura com componentes complementares para explicitar o suporte a *rollback* em planos de mudança em tal arquitetura genérica. Assim, a Figura 1 mostra a arquitetura para gerenciamento de mudanças resultante, ressaltando os componentes necessários para o suporte a *rollback*.

Basicamente, a especificação de uma nova RFC começa quando o *change requester* descreve suas necessidades em um documento de alto nível. Esse processo de formalização é feito com a interação no componente *change designer*, que é uma ferramenta que ajuda o *change requester* a preencher a RFC de uma forma clara e consistente. É importante lembrar que uma RFC expressa o que é necessário para a mudança, mas não como implantá-la. Isso é inicialmente definido quando o *operator*, também interagindo com o *change designer*, gera um *change plan*.

A saída do *change designer* é um *change plan* preliminar que necessita ser complementado posteriormente. O *change planner* é responsável por automaticamente computar um *workflow* de ações que define o *change plan* final. O algoritmo que computa este *workflow* está fora do escopo deste artigo, porém este já foi explicitado em outros trabalhos [Keller et al. 2004] [Cordeiro et al. 2008].

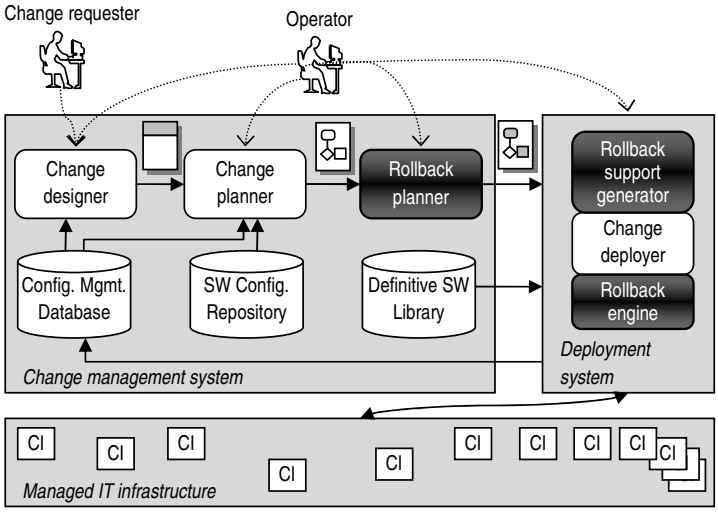

Figura 1. Arquitetura de gerenciamento de mudanças de TI

Em um sistema de mudanças sem suporte a *rollback*, o *workflow* computado pelo *change planner* estaria pronto para ser submetido, com uma simples ordem do *operator*, para o *deployment system*. Então, neste caso, as mudanças sobre a infra-estrutura de TI seriam executadas. Com este cenário, se alguma falha ocorrer durante a implantação do *change plan*, o sistema possivelmente irá entrar em um estado inconsistente, pois ações de reação à falhas não foram definidas. Como proposto anteriormente, contornamos este problema com informações de *rollback*, complementando o *change plan*. Essas informações são expressas através de um componente chamado *rollback planner*. Como podemos observar na Figura 1, o *rollback planner* tem como entrada um *workflow* de ações, e gera uma saída que incluirá marcas de *rollback* para suportar ações de *rollback* se caso alguma falha ocorrer.

Internamente, o *deployment system* é formado pelos seguintes componentes: *rollback support generator*, *change deployer* e *rollback engine*. O papel do *rollback support generator* é criar estruturas internas para suportar eventuais ações de *rollback* durante a execução do *change deployer* que, de fato, implanta as mudanças sobre os *Configuration Items* (CIs). Se alguma falha ocorrer no processo de mudança, a *rollback engine* é chamada e executará os procedimentos de *rollback*, seguindo as marcas definidas no *change plan* com suporte a *rollback*.

3.2. Marcando *Change Plans*

Como mencionado anteriormente, o *operator* é responsável, em um sistema de gerenciamento com suporte a *rollback*, por marcar o *change plan* original para complementá-lo com informações de *rollback*. Para entendermos como estas marcas são definidas, precisamos primeiro observar como RFCs e *change plans* são internamente organizados.

Uma única RFC é composta de uma ou mais operações. Cada operação é um elemento independente que precisa ser executado para cumprir a mudança requisitada na RFC. Como operações são independentes uma das outras, duas ou mais operações de uma mesma RFC podem ser executadas em paralelo. Internamente, para cada operação um único *change plan* é definido, o que significa que um *change plan* é associado em uma operação. Na verdade, todos os *change plans* de uma mesma RFC poderiam ser expressos

em uma só requisição, otimizando assim o processo de implantação. Porém, por motivos de simplificação, assumimos neste artigo que uma RFC com mais de uma operação irá apresentar um *change plan* para cada operação. Finalmente, cada *change plan* é composto por um conjunto de atividades que, encadeadas, formam o *workflow* final de ações.

Com o objetivo de que RFCs tenham a habilidade de suportar ações de *rollback*, em um trabalho anterior [Machado et al. 2008] foi definido que alguns elementos (RFCs, operações, ou atividades) podem ser marcados como *elementos atômicos* usando *marcadores de atomicidade*. Se uma RFC for marcada como atômica, todas as operações e atividades associadas irão ter atividades de *rollback*. Já na marcação de operações, cada operação que for marcada como atômica irá ter ações de *rollback* de forma independente, tendo em vista que são executadas de forma paralela. Finalmente, a atomicidade também pode ser definida no nível de atividades. Se uma atividade marcada como atômica falhar, esta irá ter ações de *rollback*, mas as atividades subseqüentes não. Se uma atividade não atômica falhar, esta não terá plano de *rollback*, e o plano de mudança associado será abortado. Assim, foi introduzido o conceito de *grupos de atomicidade*. Uma vez que ocorra falha de uma atividade que participe de um grupo atômico, todas as atividades participantes deste grupo também irão executar *rollback*.

3.3. Modelo para suporte a *rollback*

Para suportar *rollback* em *change plans*, é necessário ter um modelo que possa agregar informações do *change plan* com as possibilidades descritas nas seções anteriores. Neste trabalho usamos o modelo proposto anteriormente por este grupo de pesquisa, que expressa um *workflow* de ações que incluem suporte a *rollback* [Machado et al. 2008]. Este modelo é fortemente baseado no gerenciamento de mudanças apresentado no livro ITIL Service Support [ITIL 2006], e na abordagem para especificar *workflows* do Workflow Management Coalition (WfMC) [WfMC 2007]. A Figura 2 apresenta uma visão parcial do modelo definido, salientando os elementos necessários para o suporte a *rollback*.

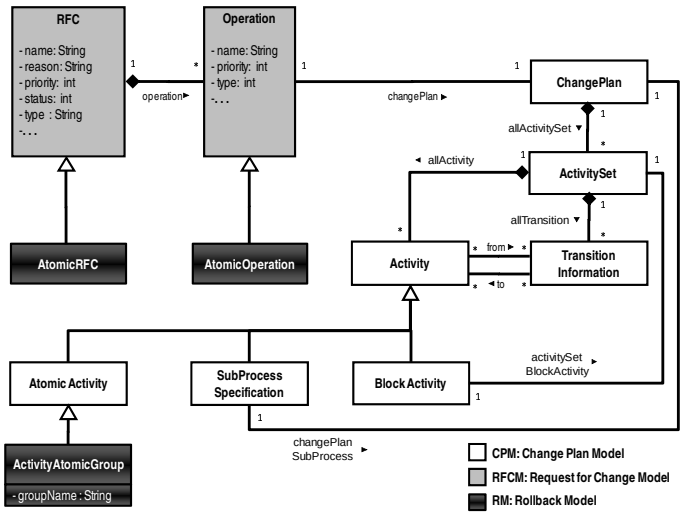

Figura 2. Modelo de RFC e *change plan* com suporte a rollback

Uma `RFC` é composta por `Operations` que, por sua vez, são compostos de um ou mais `ChangePlans`. As classes `RFC` e `Operation` possuem informações mais

abstratas de uma requisição de mudança, e então formam a parte de *Requests for Change* do modelo. Uma `AtomicRFC` é uma RFC especializada na qual ações precisam ser tratadas como uma única transação pelo *deployment system*. Exemplificando, para marcar uma RFC como atômica basta usar a classe `AtomicRFC`. Da mesma maneira, uma `AtomicOperation` é uma operação na qual possui ações que irão executar ações de *rollback* através do *deployment system* em casos onde ocorram falhas.

Cada operação em uma RFC tem um *change plan* composto por `ActivitySets`, que são grupos de uma ou mais atividades que servem para implementar um *change plan*. Uma `Activity` pode ser de mais baixo nível, sem a granularidade de refinamentos (`LeafActivity`), ou então atividades agrupadas (`SubProcessDefinition` e `BlockActivity`), que são atividades compostas por outro conjunto de atividades ou por um novo *change plan*.

Para marcar uma atividade como atômica, é preciso usar a classe `AtomicActivity`. Como algumas atividades atômicas podem ser agrupadas em grupos atômicos [Machado et al. 2008], cada atividade atômica precisa indicar em uma *string* a qual grupo atômico participa. Se nenhum nome for especificado, a atividade pertencerá por um único grupo formado somente por ela mesma. Note que não é possível uma atividade participar de mais de um grupo atômico. Se este caso em particular fosse possível, as atividades em questão funcionariam como um mecanismo que repassariam o comportamento de um grupo para todos os outros que a atividade fizesse parte. Isso aconteceria pois se um grupo atômico executasse suas ações de *rollback*, a atividade com mais de um grupo atômico também deveria executar este plano, e então levaria todos os outros grupos atômicos à desfazerem suas atividades (*rollback* em cascata).

3.4. Algoritmo de geração de ações de *rollback*

Para produzir um *workflow* que reflita uma RFC com suporte a *rollback* é necessário um mecanismo automático para a geração de ações de *rollback*. Na solução proposta, um *workflow* pode ser marcado como atômico, criando assim atividades associadas a grupos atômicos. Porém, planos que desfaçam as atividades marcadas precisam ser gerados. Após o processo de marcação ser concluído, podemos assumir que o resultado final é como o mostrado na Figura 3-a, possuindo dois grupos atômicos: AG1 e AG2.

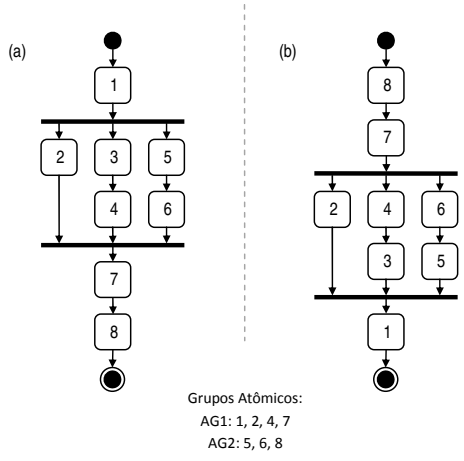

Figura 3. Exemplo de *workflow* com grupos atômicos e o seu *workflow* inverso

O primeiro passo para a geração das ações de *rollback* é a inversão do *workflow* original. A inversão é um passo importante pois é gerado um *workflow* que preserva as dependências em uma eventual falha. Ou seja, caso a atividade 8 falhar no *workflow* original, então esta atividade vai ter a ação de *rollback* executada, para depois as dependências serem revertidas gradativamente respeitando as suas relações. A inversão é feita utilizando a classe `TransitionInformation` do modelo apresentado na Seção 3.3. Esta classe possui todas as transições entre atividades de um *change plan*, tendo atributos *from* e *to* que apontam, respectivamente, para as atividades de origem e de destino. Assim, para o algoritmo de geração de ações de *rollback* processar e inverter o *workflow*, basta que sejam trocadas as origens pelos destinos, gerando então o que é mostrado na Figura 3-b.

Após a inversão do *workflow* original, é necessário analisar os grupos atômicos existentes. Essa identificação é feita no momento do processo da inversão, para que processamento não seja desperdiçado. Já que o processo de inversão precisará percorrer todas as atividades relacionadas a um *change plan*, é plausível que para otimizar o processamento já se obtenha as informações de atomicidade. Estas informações, como já mencionado, ficam nas classes `AtomicRFC`, `AtomicOperation` e `AtomicActivity`. Porém, durante o processo de inversão, o algoritmo irá somente coletar os dados do atributo *atomicGroup* da classe `AtomicActivity`. A razão para a ocorrência deste fato, é as diversas maneiras de expressar atomicidade em níveis mais altos (RFC e operações) [Machado et al. 2008]. Para gerar ações de *rollback*, o segundo passo consiste em eliminar recursivamente do *workflow* inverso as ações de atividades que se encaixam nas seguintes premissas:

1. Atividades que não fazem parte do mesmo grupo atômico que a atividade que está sendo processada no momento;
2. Atividades que não possuem grupos atômicos;
3. Atividades do mesmo grupo atômico, mas que antecedem a atividade que se está gerando as ações de *rollback*.

A primeira premissa descarta a execução de qualquer ação de *rollback* referente a outro grupo atômico. Em outras palavras, esta premissa é o mecanismo usado para respeitar o conceito de grupos de atomicidade. Ou seja, se ocorrer alguma falha entre as atividades de um grupo atômico, somente as atividades daquele grupo terão *rollback*. A segunda premissa retira do *workflow* de ações de *rollback* qualquer atividade não marcada no *workflow* original. Esta premissa faz com que ações não marcadas como atômicas sejam descartadas, deixando a atividade sem *rollback* associado. Por conseqüência, esta atividade ficará desprotegida de eventuais falhas. E por fim, a terceira premissa garante que atividades que participam do grupo atômico, mas ainda não foram executadas no *workflow* original, não recebam ação de *rollback*. Por exemplo, se a atividade 4 do *workflow* original falhar (Figura 3-a), então mesmo participando do grupo atômico AG1, o algoritmo precisará gerar ações de *rollback* para as atividades 4, 2 e 1, sem acionar *rollback* da atividade 7 que ainda não foi executada. Na Figura 4 são mostradas ações de *rollback* para as atividades 7 (Figura 4-a), 8 (Figura 4-b) e 4 (Figura 4-c).

Pode-se notar que as atividades descartadas aparecem no *workflow* de *rollback* como uma atividade *dummy*. Uma atividade deste tipo é caracterizada por não possuir nenhum tipo de ação atrelada, apesar da existência da estrutura representativa. A escolha de não retirar completamente as atividades do *workflow*, se deve ao fato do alto processamento relacionado em percorrer e analisar todas as transições de um *workflow* quando

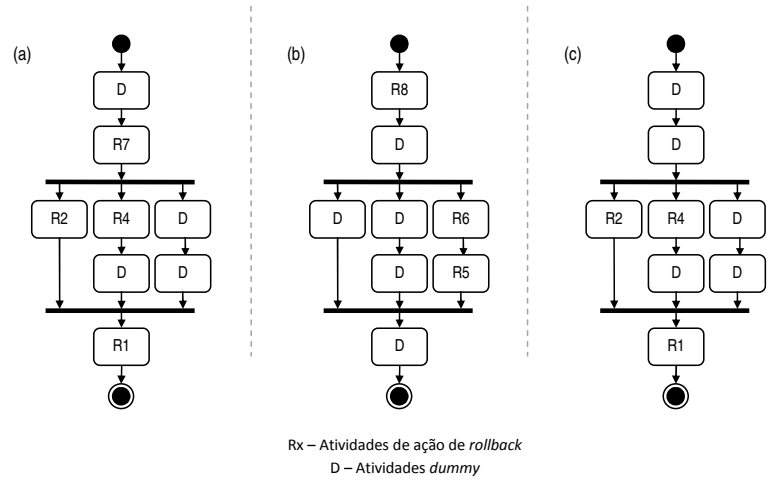

Figura 4. *Workflows* de ações de *rollback* para as atividades 7, 8 e 4, respectivamente

se quer gerar ações de *rollback* para uma determinada atividade. Vamos supor o não uso de atividades *dummy*. O resultado do *workflow* de ações de *rollback*, demonstrado na Figura 4-a, seria o seguinte: R7 com transições em paralelo para R2 e R4, e então uma *join transition* para R1. Para a geração deste, o algoritmo precisaria iterar sobre todas as transições a cada atividade do *workflow* original, pois necessitaria saber quais atividades do grupo são executadas em pararelo ou em sequência, para então montar um novo *workflow*. Com a inversão do fluxo em conjunto com o uso de atividades *dummy* , o algoritmo sempre reutiliza o *workflow* original, mantendo sua construção básica e poupando processamento. Assim, o único processamento que o algoritmo proposto apresenta é na execução da terceira premissa, quando é necessário percorrer as atividades precedentes substituindo-as por atividades *dummy*.

Por fim, é importante ressaltar que o segundo passo (eliminação de atividades), é executado sobre o *workflow* inverso para cada atividade de cada grupo atômico, gerando assim vários *workflow* de ações de *rollback*. No entanto, estes podem ser gerados de forma idêntica, dependendo da localização da atividade dentro do *workflow* original. Um exemplo deste fato ocorre no *workflow* de ação de *rollback* para a atividade 4 (Figura 4-c), que é idêntico ao plano gerado para a atividade 2. A detecção desta ocorrência para uma possível otimização é encarado como trabalho futuro.

4. Protótipo

Para provar conceito, desenvolvemos um protótipo que implementa a solução proposta de *rollback*. Nossa implementação é baseada em tecnologias e padrões de *Web services*, principalmente por três motivos: a facilidade de comunicação entre processos na Internet, pela aceitação do padrão tanto na indústria quanto na área acadêmica, e pela possibilidade de composição de serviços como o BPEL, usado para coordenar ações distribuídas em uma infra-estrutura de TI. Neste contexto, a implementação é baseada na seguintes soluções de *Web services*:

- No lado dos CIs, assumimos que existe uma interface de gerência por *Web services*. Esta interface de gerenciamento pode seguir a especificação do

Configuration Description, Deployment, and Lifecycle Management (CDDLM) [CDDLM Working Group 2007].

- Para implantar as mudanças sobre a infra-estrutura de TI, *workflows* de ações são descritos em documentos BPEL utilizando uma BPEL *engine* como a ActiveBPEL [Active Endpoints 2007].

4.1. Construções BPEL para suporte a *rollback*

Enquanto um *change plan* estiver sendo executado, o *deployment system* precisa conhecer o estado e a maneira em que as ações estão sendo executadas. Para orquestrar o sistema como expressado em um *workflow*, quatro construções BPEL foram usadas: `sequence`, `flow`, `if`, que são classificados como *containers* [Juric 2006], e `links`, que é uma construção BPEL básica para criar transições entre atividades.

A atividade BPEL `invoke` permite chamar um *Web service* remoto para executar uma determinada ação. Com isso, pode-se usar um `invoke` para executar operações remotas usando comunicação na forma *two-way* (requisição-resposta), normalmente usadas em comunicações síncronas, ou *one-way* (somente uma mensagem), normalmente usada em comunicações assíncronas [Machado et al. 2008]. Desta maneira, o protótipo implementado suporta ambos tipos de comunicação (síncrono e assíncrono) e, combinado com as estruturas `sequence`, `flow`, `if` e `links`, pode implementar efetivamente o *workflow* descrito em um *change plan*. Outra atividade BPEL usada é a `empty`, que serve quando a ação de uma atividade necessita ser descartada pelo algoritmo de geração de ações de *rollback*. Esta atividade BPEL simplesmente representa uma ação *dummy*, não possuindo nenhum processamento para a execução do processo BPEL.

Finalmente, tendo em vista detectar problemas, e com isso chamar ações de *rollback*, algumas construções BPEL adicionais foram empregadas neste protótipo. A atividade BPEL `invoke` pode incluir construções de detecção de falhas para que o *workflow* seja suscetível a diversos erros relacionados ao serviço invocado. Neste caso, a atividade `invoke` precisa ser adicionado em um BPEL *scope*, e então erros poderão ser detectados em tempo de execução com uma estrutura chamada `catch all`. Esta estrutura desvia o fluxo normal para um fluxo diferente e específico para o tratamento de exceções.

4.2. Deployment System

O *deployment system* é implementado em Java e organizado internamente em três blocos que já foram introduzidos na Figura 1: *Rollback Support Generator*, *Change Deployer*, e *Rollback Engine*. O diagrama completo do *deployment system* é apresentado na Figura 5.

Primeiramente, o *rollback support generator* recebe um *change plan* com todas as marcações de *rollback*, e depois de ler suas informações, importa um conjunto de arquivos WSDL dos CIs que serão afetados pelo *change plan*. Os arquivos WSDL são então validados para garantir que todos os recursos e operações necessárias estão disponíveis nos elementos gerenciados. Neste passo, uma verificação é feita para determinar qual tipo de comunicação irá ser usada para executar cada atividade (síncrono ou assíncrono). Após o passo de análise dos arquivos WSDL, é feita uma conversão do *workflow* original com marcas de *rollback* para um documento BPEL. Na implementação feita, após o documento BPEL que reflete o *workflow* original estar pronto, o componente *add rollback support* é chamado para adicionar as estruturas de *rollback* utilizando as construções de

BPEL descritas anteriormente. Após este processo, o elemento *create rollback actions* irá criar as ações de *rollback* usando o algoritmo que foi detalhado neste artigo. Note que o elemento *create rollback actions* faz parte do componente *rollback support generator*, uma vez que o algoritmo que gera o *workflow* de *rollback* também é essencial para o suporte de *rollback*, tendo como produto um documento BPEL que é representado pelo componente *rollback engine*. Finalmente, com toda a informação pronta para ser levada à implantação, o componente *create deployment descriptor* cria um arquivo complementar chamado Process Deployment Descriptor (PDD), necessário pela ActiveBPEL *engine* (também usado na implementação) para executar os *workflows*. O conjunto completo de arquivos gerados é então encaminhado para o *change deployer*.

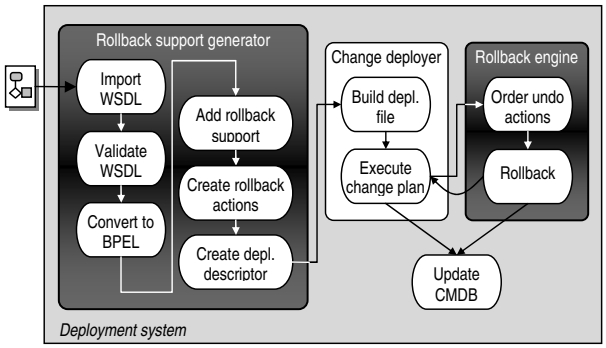

Figura 5. Deployment system

O *change deployer* expande a funcionalidade da BPEL engine, adaptando as necessidades do projeto em relação à intenção do *deployment system* como um todo. Primeiramente, ele empacota todos os arquivos gerados previamente (WSDL, BPEL e PDD) e então o ActiveBPEL é chamado para executar o *change plan*. Na verdade, o ActiveBPEL é responsável por executar o *change plan*, fazer a detecção das falhas, e executar as ações de *rollback*. Uma vez detectadas as falhas, o fluxo normal do *change plan* é desviado para a *rollback engine*. Na implementação corrente, diferentemente do trabalho realizado no passado [Machado et al. 2008], a *rollback engine* é implementada como um segundo documento BPEL, e também interpretado/executado pelo ActiveBPEL.

Quando a *rollback engine* é invocada, então ela simplesmente irá executar o plano de *rollback* gerado anteriormente. Cada ação de *rollback* possui uma operação reversa àquela que foi executada no *workflow* original. Por exemplo, se uma atividade original é descrita com a ação `install`, o reverso será uma ação de `uninstall` [Machado et al. 2008]. Depois da execução de uma ação de *rollback*, o fluxo do *workflow* pode retornar para o *change plan* original, ou então seguir diretamente para o seu fim, dependendo de como as atividades atômicas foram definidas pelo operador do sistema. Assim, o último passo então, é atualizar o CMDB (Configuration Management Database) para refletir o novo estado da infra-estrutura de TI gerenciada.

5. Estudo de Caso & Análise

Para gerar uma prova de conceito e testar a viabilidade técnica da nossa proposta, realizamos um estudo de caso com uma RFC para instalação de aplicativos, tendo atividades suficientes para que o algoritmo que foi descrito neste artigo fosse analisado. Para isso,

definimos uma RFC com a seguinte descrição: instalação da plataforma de gerenciamento de projetos *dotProject* em uma máquina dedicada para este fim. Neste contexto, logo depois do *change requester* descrever a RFC e especificar suas operações, o *operator* especifica quais atividades do *change plan* gerado serão marcadas como atômicas com ajuda do *rollback planner*. Como resultado destas etapas, obtivemos o *change plan* marcado com grupos atômicos como é mostrado na Figura 6. Na verdade, o *workflow* mostrado na Figura 6 é o *workflow* resultante do algoritmo de tradução, onde um *change plan* marcado é traduzido para um *workflow* BPEL executável. A representação é a mesma, já que ambos documentos precisam refletir as mesmas transições e atividades. As razões que levaram ao *operator* à definição de tais grupos atômicos não serão explicitadas neste artigo, até porque é possível marcar como atômico qualquer combinação de atividades, desde que uma não participe de dois grupos atômicos ao mesmo tempo. Portanto, o suporte a *rollback* funciona como uma linguagem de programação onde o programador tem a liberdade de escolha, construindo estruturas dependendo da sua necessidade.

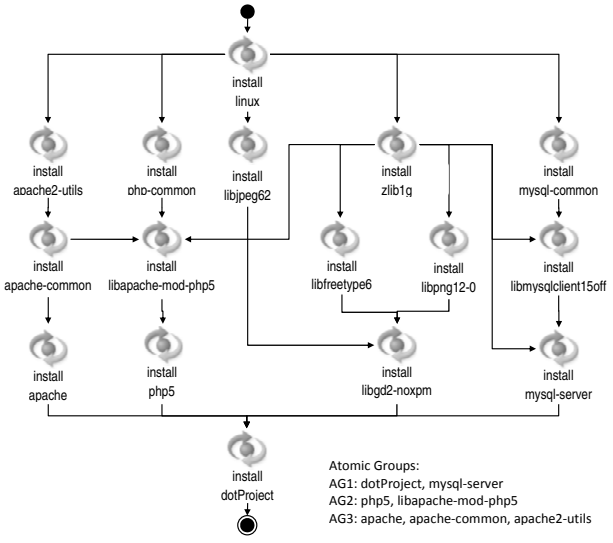

Figura 6. *Change plan* marcado com grupos de atômicidade entre atividades

O próximo passo no sistema de gerenciamento de mudança é a submissão do *change plan* marcado para o *deployment system*, onde mais especificamente irá ser tratado pelo *rollback support generator*. Nesta etapa, o algoritmo de geração de ações de *rollback* é executado. Após a sua execução, podemos constatar a geração de um documento BPEL que possui todas as ações de *rollback* para cada atividade participante de cada grupo atômico. Este documento BPEL, que representa a funcionalidade da *rollback engine*, é invocado cada vez que ocorre uma falha. O *workflow* de ações de *rollback* gerado para a atividade de instalação do pacote *dotProject*, pode ser observado na Figura 7. Observações sobre a execução do algoritmo e da invocação do *rollback* são:

- Tendo como entrada o *change plan* da Figura 6, o algoritmo de geração de ações de *rollback* gerou o documento BPEL em 145 milisegundos;
- Com base nos *logs* da ActiveBPEL *engine*, a execução do *workflow* de ações de *rollback* sempre teve tempo menor do que 1 segundo. Este *workflow* foi gerado a partir do *change plan* descrito na Figura 6, sendo testado falhas em todas as

atividades participantes de cada grupo atômico. É importante salientar que nos *logs* da *engine* não são representados tempos menores do que 1 segundo;

- A comunicação entre a *rollback engine* e os elementos gerenciáveis possui um tempo desprezível se executado em uma rede local, pois a troca de mensagens para cada ação de *rollback* consome aproximadamente 10 pacotes, com tamanho médio de 200,5 bytes.

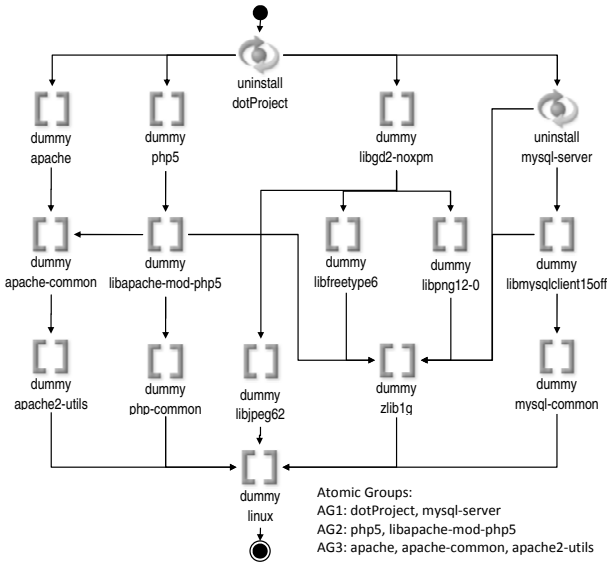

Figura 7. *Workflow* de ação de *rollback* para a instalação do *dotProject*

6. Conclusões e Trabalhos Futuros

Neste artigo, discutimos como organizações implantam suas mudanças e a importância de se ter um sistema de gerenciamento de mudanças com suporte a *rollback*. Muitas das organizações tem uma infra-estrutura de TI complexa, onde mudanças são implantadas por humanos, aumentando a probabilidade de ocorrerem falhas. Por esta razão, com base em trabalhos passados desenvolvidos por este grupo de pesquisa, propomos um algoritmo de geração de ações de *rollback* que se encaixa em uma arquitetura de gerenciamento de mudanças. Assim, nesta arquitetura, um administrador/operador possui a possibilidade de marcar diversas partes de uma RFC como atômica. Caso haja alguma falha na implantação desta RFC, um *workflow* de ações de *rollback* previamente gerado será executado para levar o estado do sistema para um estado consistente.

Os resultados obtidos demonstram coerência entre a especificação de grupos atômicos em um *change plan* e o *workflow* de ações de *rollback*. Ou seja, seguindo as regras de especificação de grupos atômicos, e executando os passos do algoritmo proposto, pode-se chegar em um *workflow* que reverte falhas de um *change plan* real. Também foi possível observar que apesar do grande tamanho dos arquivos BPEL gerados para o *workflow*, o uso de atividades *dummy* não influenciou no processamento destes. Além disso, podemos salientar a reutilização das estruturas do *workflow* original no algoritmo de geração de ações de *rollback*, otimizando o seu processamento.

Apesar disso, detectamos que otimizações adicionais são factíveis, e servem como trabalhos futuros. Uma destas otimizações é a detecção de *workflows* de ações de *rollback*

idênticos, que são gerados a partir de duas atividades diferentes. Assim, o algoritmo geraria um só *workflow* de ações de *rollback* que poderia ser apontado por várias atividades no *workflow* original, e invocado caso houvesse falhas.

Referências

Active Endpoints (2007). ActiveBPEL Open Source Engine. http://www.activebpel.org.

Alves, R. S., Granville, L. Z., Almeida, M. J. B., and Tarouco, L. M. R. (2006). A Protocol for Atomic Deployment of Management Policies in QoS-Enabled Networks. In *6th IEEE International Workshop on IP Operations and Management (IPOM 2006)*, LNCS 4268, pages 132–143.

Andrzejak, A., Hermann, U., and Sahai, A. (2005). FEEDBACKFLOW-An Adaptive Workflow Generator for Systems Management. In *2nd International Conference on Automatic Computing (ICAC 2006)*, pages 335–336. IEEE Computer Society.

Bartolini, C., Sauvé, J., and Trastour, D. (2006). It service management driven by business objectives - an application to incident management. In *11th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006)*, pages 45–55, Vancouver, Canada.

CDDLM Working Group (2007). Configuration Description, Deployment, and Lifecycle Management. http://forge.gridforum.org/projects/cddlm-wg.

Cordeiro, W. L. D. C., Machado, G. S., Daitx, F. F., Both, C. B., Gaspary, L. P., Granville, L. Z., Saikoski, K., Sahai, A., Bartolini, C., and Trastour, D. (2008). A template-based solution to support knowledge reuse in it change design. In *11th IEEE/IFIP Network Operations and Management Symposium (NOMS 2008)*, Salvador, Brazil. *(To appear)*.

Enns, R. (2006). NETCONF Configuration Protocol. RFC 4147.

IT Infrastructure Library (2000). *ITIL Service Support Version 2.3*. Office of Government Commerce.

ITIL (2006). Information Technology Infrastructure Library (ITIL).

Juric, M. B. (2006). *Business Process Execution Language for Web Services BPEL and BPEL4WS: 2nd Edition*. Packt Publishing.

Keller, A., Hellerstein, J. L., Wolf, J. L., Wu, K.-L., and Krishnan, V. (2004). The champs system: Change management with planning and scheduling. In *9th IEEE/IFIP Network Operations and Management Symposium (NOMS 2004)*, pages 395–408, Seoul, Korea.

Machado, G. S., Cordeiro, W. L. D. C., Daitx, F. F., Both, C. B., Gaspary, L. P., Granville, L. Z., Saikoski, K., Sahai, A., Bartolini, C., and Trastour, D. (2008). Enabling rollback support in it change management systems. In *11th IEEE/IFIP Network Operations and Management Symposium (NOMS 2008)*, Salvador, Brazil. *(To appear)*.

OASIS Standards (2007). Business process execution language version 2.0. http://docs.oasis-open.org/wsbpel/2.0/.

Rebouças, R., Sauvé, J., Moura, A., Bartolini, C., and Trastour, D. (2007). A decision support tool to optimize scheduling of it changes. In *10th IFIP/IEEE International Symposium on Integrated Network Management (IM 2007)*, pages 343–352.

WfMC (2007). Workflow Process Definition Interface. http://www.wfmc.org.