

Controle de Modificações em Software no Desenvolvimento Baseado em Componentes

Luiz Gustavo Lopes, Leonardo Murta, Cláudia Werner

COPPE/UFRJ – Programa de Engenharia de Sistemas e Computação

Universidade Federal do Rio de Janeiro – Caixa Postal 68511

CEP 21945-970 – Rio de Janeiro – RJ – Brasil

{luizgus, murta, werner}@cos.ufrj.br

Abstract. *Changes may occur at anytime in a software lifecycle. In order to avoid rework and information loss, among other problems, these changes must be controlled. This paper discusses change control in the context of the Component-Based Development paradigm, discussing some problems in this area, suggesting an approach to mitigate some of these problems and presenting a computational implementation to support this approach.*

Resumo. *Um software está sujeito a modificações em qualquer etapa de seu ciclo de vida. Para evitar re-trabalho e perda de informações, entre outros problemas, estas modificações devem ser controladas. Este artigo trata do controle de modificações no contexto de Desenvolvimento Baseado em Componentes, discutindo problemas existentes na área, propondo uma abordagem para amenizar alguns destes problemas e apresentando uma implementação computacional de apoio a essa abordagem.*

1. Introdução

Um software está sujeito a modificações em qualquer etapa de seu ciclo de vida. Quando estas são realizadas sem controle, podem surgir diversos problemas, como perda de informações em artefatos alterados de forma concorrente, perda de informações sobre quais versões de artefatos estão presentes em cada cliente, perda da memória das modificações, entre outros. Para permitir que o software evolua de maneira controlada, existe a Gerência de Configuração de Software (GCS), que é a disciplina de controle de modificações em sistemas de software grandes e complexos (Estublier *et al.*, 2002), sendo um elemento crítico no processo de manutenção de software (IEEE, 1998b).

Um dos tipos de sistemas que auxiliam a aplicação da GCS é o sistema de controle de modificações. Este sistema é encarregado de dar suporte ao processo de controle de modificações, de manter as informações coletadas durante a execução do processo e disseminá-las a todos os interessados e autorizados (Murta, 2004).

Porém, o controle de modificações em software desenvolvido através do paradigma de Desenvolvimento Baseado em Componentes (DBC), ou seja, desenvolvido reutilizando-se componentes já existentes (Brown, 2000), não é adequadamente auxiliado pelos sistemas existentes. As principais deficiências destes sistemas estão relacionadas à ausência de informações necessárias para a manutenção de

software, como informações sobre os consumidores dos componentes e de dados contratuais, e à pouca flexibilidade existente para a definição de processos de controle de modificações e das informações que devem ser coletadas durante a execução destes processos (Murta, 2004).

Um problema identificado refere-se à dificuldade de conhecimento sobre os responsáveis pela manutenção dos componentes, o que pode ser complexo, principalmente, em um contexto de produção de componentes compostos desenvolvidos por equipes diferentes. Este problema pode ser amenizado identificando-se os produtores dos componentes e os contratos firmados entre as partes produtora e consumidora. Neste trabalho, é introduzida uma abordagem e uma implementação computacional para o controle de modificações que, apesar de poder ser utilizada em um cenário de desenvolvimento convencional de software, se torna ainda mais importante no contexto do DBC, pois visa amenizar as deficiências detectadas, como, por exemplo, a falta de apoio à resolução do problema da cadeia de responsabilidades de manutenção.

Este artigo está organizado em seis seções, incluindo esta primeira. Na Seção 2, são apresentados conceitos relacionados a GCS e ao DBC, as características dos sistemas de controle de modificações atuais e as dificuldades encontradas na utilização destes sistemas no contexto do DBC. Na Seção 3, é proposta uma abordagem que visa contornar algumas destas dificuldades encontradas. Na Seção 4, é apresentado um protótipo implementado para dar suporte à utilização da abordagem proposta. Na Seção 5, estão descritos trabalhos relacionados a este, e, finalmente, são apresentadas as contribuições e trabalhos futuros na Seção 6.

2. Cenário Atual

A GCS é definida pela IEEE (1990) como “uma disciplina que visa identificar e documentar as características de itens de configuração, controlar as suas alterações, armazenar e relatar as modificações aos interessados e garantir que foi feito o que deveria ter sido feito”. Desta forma, a GCS não se propõe a definir quando e como devem ser executadas as modificações nos artefatos de software, papel este reservado ao próprio processo de desenvolvimento de software. A sua atuação ocorre como processo auxiliar de controle e acompanhamento dessas atividades.

A GCS pode ser tratada sob diferentes perspectivas em função do papel exercido pelo participante do processo de desenvolvimento de software (Murta, 2004). Na perspectiva gerencial, a GCS é dividida em quatro funções, que são (IEEE, 1990; ISO, 1995): identificação da configuração, controle da configuração, acompanhamento da configuração e auditoria da configuração. Contudo, sob a perspectiva de desenvolvimento, a GCS é dividida em três sistemas principais, que são: controle de modificações, controle de versões e controle de construção (*building*) e liberação (*release*).

O sistema de controle de modificações é encarregado de executar a função de controle da configuração de forma sistemática, armazenando todas as informações geradas durante o andamento das requisições de modificação e relatando essas informações aos participantes interessados e autorizados, assim como estabelecido pela função de acompanhamento da configuração. O Bugzilla (Barnson *et al.*, 2003) é um exemplo destes sistemas.

O sistema de controle de versões permite que os itens de configuração sejam identificados, segundo critérios estabelecidos pela função de identificação da configuração, e que eles evoluam de forma distribuída e concorrente, porém disciplinada. Um exemplo destes sistemas é o CVS (Cederqvist, 2003).

O sistema de controle de construção e liberação automatiza o complexo processo de transformação dos diversos artefatos de software que compõem um projeto no sistema executável propriamente dito, de forma aderente aos processos, normas, procedimentos, políticas e padrões definidos para o projeto. Um exemplo destes sistemas é o Ant (Hatcher *et al.*, 2002).

Este trabalho está contextualizado na função de controle da configuração, quando analisado sob a perspectiva gerencial, e no sistema de controle de modificações, quando analisado sob a perspectiva de desenvolvimento. A seguir, são apresentadas as principais características dos sistemas de controle de modificações, o paradigma DBC, e as principais dificuldades existentes na utilização destes sistemas para o controle de modificações em software desenvolvido com base em componentes.

2.1. Sistemas de Controle de Modificações

A GCS é fortemente calcada em processos e normas (IEEE, 1987; ISO, 1995), que definem os procedimentos necessários para permitir a evolução controlada do software. Para automatizar a execução desses processos, são utilizados os sistemas de controle de modificações.

Inicialmente, os sistemas de controle de modificações tinham seu foco principal em modificações corretivas (Pressman, 2005), mas, com o amadurecimento da área, esses sistemas passaram a ser utilizados para qualquer tipo de modificação. Em alguns casos, quando o processo de desenvolvimento prioriza a geração precoce de liberações, os sistemas de controle de modificações passam a ser utilizados desde o início do desenvolvimento, exercendo a função de máquina de processo e recebendo o nome de sistema de controle de requisições.

Um dos sistemas de controle de modificações mais conhecidos e utilizados em projetos de software livre é o Bugzilla (Barnson *et al.*, 2003). O Bugzilla provê suporte à busca detalhada de modificações, criação de vínculos entre modificações, controle do estado das modificações, relacionamento entre modificações e os artefatos alterados propriamente ditos, envio de mensagem de notificação de eventos, e geração de gráficos e relatórios para análise das informações armazenadas. Além disso, uma preocupação constante no projeto é a necessidade de um alto desempenho da ferramenta. Para atingir esse objetivo, o Bugzilla faz uso de banco de dados relacional e interfaces HTML (*Hypertext Markup Language*) (W3C, 1999).

Contudo, existe atualmente um número relativamente grande de sistemas com características semelhantes ao Bugzilla (CMCrossroads, 2005). Alguns exemplos são o Roundup (2005), que permite incluir e alterar requisições de modificação por e-mail, característica também presente no Bugzilla, e o sistema comercial ClearQuest (White, 2000), que faz parte da iniciativa da IBM Rational de prover um ambiente integrado de GCS, conhecido como UCM (*Unified Change Management*).

O sistema ClearQuest é um dos mais flexíveis com relação à configuração do processo de manutenção, que se baseia em um diagrama de estados. Além de possuir alguns modelos de processo pré-cadastrados, ele permite a alteração e criação de novos modelos, através de sua ferramenta administrativa, disponibilizada para o sistema operacional Windows. Para os usuários finais, este sistema é disponibilizado pela Web ou como aplicativo Windows.

Outro sistema que provê certa flexibilidade para customização do processo de manutenção e dos formulários de requisição de modificação é o JIRA (Atlassian, 2005), sistema comercial que possui licença gratuita quando utilizado por projetos que sejam de Código Aberto ou por empresas sem fins lucrativos.

Estes sistemas foram construídos tendo como foco o controle de modificações em software desenvolvido de maneira convencional, sem explorar características importantes para a manutenção de software desenvolvido através do paradigma DBC. A seguir, são apresentados alguns conceitos envolvidos neste paradigma e algumas das limitações dos sistemas de controle de modificações atuais quando utilizados no contexto do DBC.

2.2. Desenvolvimento Baseado em Componentes

DBC, ou Engenharia de Software Baseada em Componentes, é uma disciplina de desenvolvimento de sistemas através da reutilização de componentes bem definidos, produzidos independentemente (Brown, 2000; Larsson, 2000).

O DBC faz uso de componentes, interfaces e conectores como elementos de estruturação de sistemas. Componentes, que são partes reutilizáveis de software (D'Souza *et al.*, 1998), fazem uso de interfaces descritas de forma contratual para interagir com os demais elementos de software (Szyperski, 2002). A ligação propriamente dita entre os componentes, que pode ser simples ou complexa dependendo de questões como distribuição, adaptação e coordenação, é obtida através dos conectores (Shaw *et al.*, 1996; Larsson, 2000; OMG, 2002).

No DBC, as equipes de desenvolvimento de software podem ser diferenciadas quanto ao foco do desenvolvimento. Há as equipes que desenvolvem componentes, chamadas de equipes produtoras, que atuam como fornecedoras de componentes a serem reutilizados e são as mais similares às equipes de desenvolvimento convencional, há as equipes que desenvolvem software reutilizando componentes, chamadas de equipes consumidoras, e há as que tanto desenvolvem componentes quanto reutilizam componentes no desenvolvimento, chamadas de equipes híbridas.

Existem diferentes métodos para apoiar o DBC. Dentre os mais conhecidos estão Catalysis (D'Souza *et al.*, 1998), UML Components (Cheesman *et al.*, 2000) e Kobra (Atkinson *et al.*, 2001).

Apesar da existência desses métodos de DBC, a carência de processos, padrões e ferramental de suporte ainda é grande. Com o uso de DBC, a quantidade de artefatos produzidos durante o processo de desenvolvimento aumenta substancialmente em comparação ao desenvolvimento convencional, já que cada componente possui, além dos artefatos convencionais (e.g. documento de requisitos, modelos de análise e projeto, etc.), interfaces e conectores. Além disso, esses artefatos, que se situam em diferentes

níveis de abstração, estão intimamente relacionados através do conceito de componente. Ao serem reutilizados, os componentes podem evoluir, paralelamente à evolução dos componentes originais. Os sistemas de GCS devem manter os rastros entre os componentes originais e os reutilizados, possibilitando o envio de notificações para os produtores e os consumidores (Atkinson *et al.*, 2001).

Tanto a GCS quanto o DBC têm como principais características o aumento da produtividade, o aumento da qualidade e a redução de custos (Kwon *et al.*, 1999). Entretanto, para potencializar essas características, é necessário que ambas as técnicas sejam adotadas de forma consistente e integrada.

2.3. Dificuldades existentes

No contexto do DBC, ao receber uma requisição de modificação de um cliente, a equipe consumidora pode identificar que um componente reutilizado, produzido por outra equipe, precisa de manutenção. Ela deverá avaliar e decidir quem fará a manutenção: se a equipe produtora do componente, se ela própria, ou se ela optará pela aquisição de um novo componente, de um outro produtor. Informações importantes para esta decisão são os dados do produtor e do contrato de manutenção do componente, que deve explicitar os direitos e as obrigações de cada parte. Caso o contrato seja desrespeitado, a parte lesada pode requerer os seus direitos judicialmente.

Caso o contrato garanta a manutenção do componente para o tipo de modificação necessária, o consumidor envia uma requisição de modificação para o produtor e este gera uma nova liberação do componente.

É possível que o produtor também tenha reutilizado um componente de terceiros na construção de seu componente, podendo identificar que a manutenção requerida está relacionada ao componente reutilizado. Neste caso, o produtor do componente passa a ocupar o papel de consumidor e a situação descrita anteriormente se repete para a realização da modificação. Isso ocorrerá até que se encontre o responsável pela manutenção. Quando houver vários níveis de reutilização, este processo de identificação dos responsáveis e liberação dos componentes em cada nível pode ser lento. Uma opção dos consumidores seria implementar uma solução temporária até que os produtores liberassem uma solução definitiva para a modificação.

Tendo recebido a solicitação de modificação, o produtor responsável pelo componente deverá levar em consideração a sua importância e o interesse das outras equipes consumidoras nesta modificação, para fins de priorização e decisão pela implementação da modificação (Dantas *et al.*, 2003). Uma manutenção corretiva certamente será de interesse dos outros consumidores, o que não é necessariamente verdade para uma manutenção evolutiva.

Uma vez gerada uma nova liberação do componente, deve-se propagar esta informação aos seus consumidores, segundo a função da GCS de acompanhamento da configuração (ISO, 1995; IEEE, 1998a).

No caso em que equipes híbridas reutilizam este componente, a atualização deste resultará na geração de uma nova liberação dos componentes que o reutilizam, sendo necessário novamente a disseminação desta informação para os seus consumidores. Em resumo, componentes podem ser compostos de outros componentes, em vários níveis de

composição, e a rede de consumidores de um determinado componente pode se tornar complexa, o que impacta na disseminação de informações.

Também podem incidir questões legais sobre a atualização de um componente. O produtor do componente deve conhecer o contrato firmado com seus consumidores para decidir em que condições a atualização deverá ocorrer. Dependendo do tipo da modificação e das cláusulas contratuais, a atualização poderá ser feita de forma gratuita ou não.

Outra possibilidade é o produtor decidir pela modificação de um componente à revelia dos consumidores, por exemplo, devido a imposições de mercado. Neste caso, ele deve estar ciente das restrições contratuais. Se o contrato obrigá-lo a manter a liberação antiga do componente, ele terá que ser capaz de dar manutenção a esta liberação em paralelo ao desenvolvimento da nova versão do componente.

Portanto, torna-se necessária a manutenção dos rastros dos componentes com relação a seus produtores e consumidores, ou seja, para cada versão de componente, é preciso saber quem é seu produtor, quem são seus consumidores, quais são os seus dados para contato, e qual é o contrato assumido entre o produtor e o consumidor para o determinado componente. Este problema, que será referenciado como *cadeia de responsabilidades de manutenção*, não é considerado pelos atuais sistemas de controle de modificação.

Um outro problema é referente aos processos de GCS. Para o desenvolvimento convencional de software, há normas internacionais (IEEE, 1987; ISO, 1995; IEEE, 1998a) que fornecem diretrizes sobre o processo de controle de configuração. Porém, elas não tratam especificamente de DBC e suas particularidades.

Estas normas devem ser utilizadas como guias, e o processo de controle de modificações deve poder ser alterado de acordo com as mudanças nos papéis dos desenvolvedores e mudanças no próprio processo, o que é essencial para qualquer atividade colaborativa (Weber, 2001). Para DBC, a necessidade de customização do processo de controle de modificações é ainda maior, visto o estado imaturo em que estes processos se encontram e a falta de diretrizes específicas para DBC por parte das normas de GCS. Portanto, é importante que os sistemas de controle de modificações forneçam facilidades para customização dos processos.

Além disso, novas informações precisam ser coletadas nos processos de GCS para DBC, como, por exemplo, quem são os produtores dos componentes e como localizá-los. Porém, não há consenso com relação às informações que devem ser coletadas. Por isso, um sistema de controle de modificações deve permitir a customização de quais informações serão coletadas, e em que momento do processo.

A maior parte dos sistemas de controle de modificações não permite a customização de processos e formulários de coleta de dados. Os sistemas encontrados que possuem características deste tipo são o IBM Rational ClearQuest (White, 2000) e o JIRA (Atlassian, 2005). Porém, eles possuem algumas limitações, como a impossibilidade de representar claramente os produtos gerados em cada atividade e de definir subatividades em vários níveis, o que tornaria o processo mais completo e compreensível.

3. Auxílio ao Controle de Modificações no Contexto do DBC

Com base nas dificuldades existentes para controlar as modificações de software desenvolvido através do DBC, discutidas na Seção 2.3, propomos uma abordagem de controle de modificações mais adequada a este paradigma, que seja capaz de manter e disponibilizar as informações necessárias às equipes envolvidas na manutenção e seja flexível no que se refere à customização dos processos utilizados. Esta abordagem é chamada de Odyssey-CCS (*Change Control System*).

Na Fig. 1 são apresentadas as principais atividades e a arquitetura com os principais componentes presentes na abordagem. As atividades são representadas por elipses e os componentes por caixas, e abaixo de cada atividade estão os componentes que a apóiam. A seguir, são detalhadas cada uma das atividades exibidas na Fig. 1.

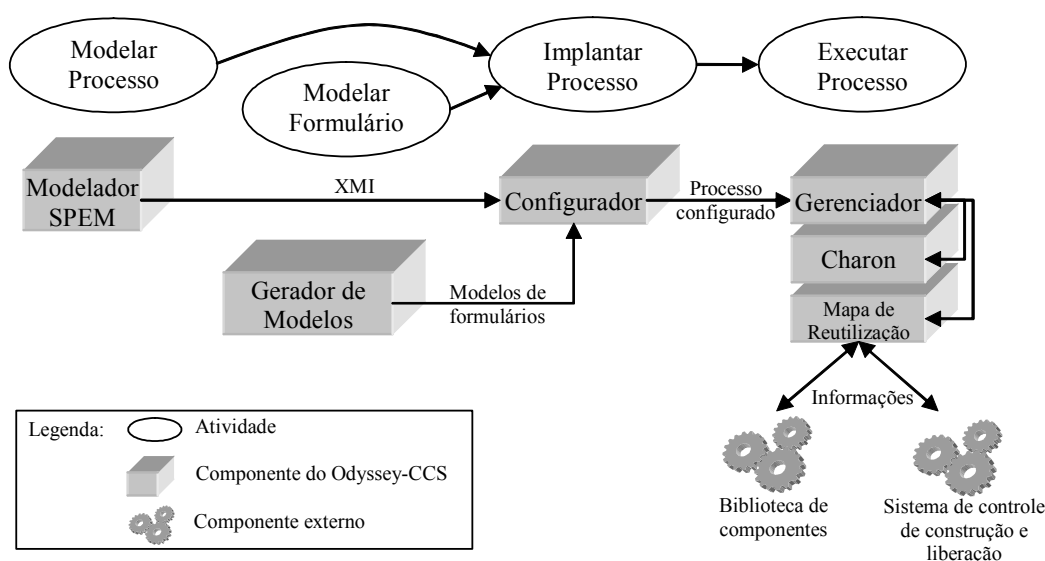


Fig. 1 - Abordagem Odyssey-CCS

3.1. Modelagem de processos

Uma das dificuldades discutidas na Seção 2.3 refere-se à necessidade de adaptação dos processos de GCS para DBC, e ao pequeno número de sistemas de controle de modificações que fornecem algum apoio a este tipo de trabalho. Na expectativa de amenizar este problema, a abordagem proposta inclui uma atividade de modelagem de processos.

A principal função desta atividade é modelar processos de GCS utilizando alguma notação padronizada de processos, sendo de responsabilidade do Gerente de Configuração. Para facilitar a modelagem e a compreensão dos processos, deve ser possível modelar e visualizar os processos graficamente, sem a necessidade de alterações em códigos-fonte ou extensos arquivos de configuração. Além disso, o processo modelado deve poder ser exportado.

Para dar apoio a esta atividade, a abordagem possui um componente para modelagem de processos chamado Modelador SPEM. Ele permite a modelagem gráfica de processos seguindo a notação SPEM (*Software Process Engineering Metamodel*) (OMG, 2005b), definida pela OMG (*Object Management Group*).

SPEM é uma notação própria para a modelagem de processos de software e é baseada no MOF (*Meta-Object Facility*) (OMG, 2005a), um metamodelo padrão. Ela possui vários diagramas adaptados da UML para modelagem de processos. Por exemplo, para modelagem do fluxo de atividades do processo e seus produtos, pode ser utilizado o Diagrama de Atividades da UML, e para a modelagem dos papéis associados a cada atividade, pode ser utilizado o Diagrama de Casos de Uso.

Na modelagem do fluxo de atividades do processo, pode-se utilizar vários elementos, como início, fim, sincronismo e decisões, e elementos da SPEM, como atividades (*Activity*), macro-atividades (*WorkDefinition*), que são compostas, produtos (*WorkProduct*) gerados pelas atividades e papéis (*ProcessPerformer* e *ProcessRole*). Cada elemento da SPEM possui um ícone sugerido na especificação da notação.

O Modelador SPEM também permite a exportação do processo modelado em arquivo no padrão XMI (*XML Metadata Interchange*) (OMG, 2005c), que é uma especificação que possibilita a geração de modelos baseados no MOF em XML. Assim, sistemas que sejam compatíveis com a notação SPEM poderão ler o processo descrito neste arquivo.

3.2. Criação de modelos de formulários

Assim como não existe um único processo de controle de modificações para DBC que possa ser utilizado em todos os casos, também não existe um conjunto único de informações que devam ser coletadas nos processos, conforme discutido na Seção 2.3. Por isso, a abordagem possui uma atividade de criação de modelos de formulários para coleta de informações, possibilitando a definição de quais informações serão solicitadas em cada etapa do processo. Esta customização deve poder ser realizada de maneira simples, idealmente de forma gráfica, sem a necessidade de alterações em código-fonte. O responsável por esta atividade é o Gerente de Configuração.

Um modelo de formulários (*template*) é composto por vários tipos de campos utilizados para a coleta de informações. Exemplos destes campos são: caixas de texto (*text-box*), caixas de seleção (*combo-box*), caixas de marcação (*check-box*), áreas de texto (*text-area*) e campos para importação de arquivos.

Os modelos de formulários definidos podem ser associados a cada produto das atividades de um processo de controle de modificações, ou às próprias atividades, sendo preenchidos pelos usuários durante a execução do processo. Ao final de cada atividade, é gerado um documento que agrega todas as informações coletadas na atividade realizada. Este documento é então associado à modificação em vigor.

Para auxiliar esta atividade, a abordagem possui o componente Gerador de Modelos. Este componente é um compositor de modelos de formulários, que permite a geração e alteração de modelos de formulários graficamente. Através dele, é possível montar telas de formulários contendo vários tipos de campos para aquisição de informações necessárias nos processos de controle de modificações.

3.3. Implantação de processos

Uma vez modelado, o processo deve ser implantado pelo Gerente de Configuração para poder ser executado. A implantação de um processo é composta por

quatro subatividades, apoiadas pelo componente Configurador: importação do processo, atribuição de responsabilidades, configuração da coleta de informações e configuração do envio de notificações.

O processo a ser importado deve ser baseado na SPEM e deve ser transportado em arquivo no padrão XMI. De forma a manter a política de flexibilidade proposta pela abordagem, é possível que haja projetos seguindo processos diferentes. Dessa forma, ao importar um processo, deve-se selecionar ou criar um novo projeto que seguirá o processo importado. Uma vez interpretado, o processo é verificado e armazenado. Caso haja alguma falha, o Gerente de Configuração é avisado e a importação é cancelada.

A atribuição de responsabilidades consiste em associar usuários da abordagem a papéis do processo importado. Como cada atividade possui um conjunto de papéis responsáveis pela sua execução, ao fazer a associação com usuários, sabe-se quem é responsável por cada atividade do processo. Essa informação é muito importante para a execução do processo, no momento de verificar se um determinado usuário pode executar determinada atividade, e no momento de disseminar informações sobre o andamento da execução de um processo.

A configuração da coleta de informações consiste em associar modelos de formulários a produtos de atividades, ou às próprias atividades. Isso significa especificar quais informações deverão ser coletadas em cada atividade do processo. Assim, na execução de determinada atividade do processo, o formulário apresentado ao usuário deverá ser baseado no modelo associado.

Outra configuração a ser feita está relacionada ao envio de notificações. Devem ser informadas quais atividades do processo referem-se ao envio de notificações, e, para cada uma, devem ser configurados os destinatários, o assunto e a mensagem a ser enviada.

3.4. Execução de processos

A atividade de execução dos processos possui como requisitos básicos uma máquina de processos para controlar a sequência de atividades definidas e de uma base de informações disponíveis aos usuários para auxiliar a execução das atividades do processo. Para atender estes dois requisitos básicos, são utilizados os componentes Gerenciador, Charon (Murta, 2002) e Mapa de reutilização.

O componente Gerenciador tem como funções: exibir o formulário adequado a cada atividade em execução para os papéis responsáveis, coletar e armazenar os dados informados pelos usuários e dar andamento ao processo. Para isso, ele recebe o processo implantado como entrada e utiliza a Charon para saber qual atividade deve ser executada em cada momento.

A máquina de processos Charon possui diversas características importantes para a execução de processos, como (Murta, 2002):

- A possibilidade de execução de processos contendo ciclos e recursão em sua representação, o que é possível modelar através da notação SPEM;
- O mapeamento automático da definição gráfica do processo na representação executável, ou seja, sem necessidade de intervenção humana;

- A possibilidade de reutilização de processos, tanto como um processo global quanto como uma atividade dentro de outro processo; e
- A possibilidade de extensão de suas funcionalidades.

É ela quem fornece as atividades pendentes de execução, infere as próximas atividades a serem executadas e informa quando um processo chegou ao final.

Quando um usuário, membro de uma equipe produtora ou consumidora, se identifica no Odyssey-CCS, são exibidas quais atividades estão pendentes para os papéis que ele desempenha e que ele está autorizado a executar. Ao selecionar uma atividade, é exibido o formulário que foi associado ao produto desta atividade, ou a ela própria. O usuário preenche as informações solicitadas e a Charon continua a execução do processo, procurando e disponibilizando a próxima atividade a ser executada.

Como já foi identificado, as equipes consumidoras e produtoras precisam ter acesso a determinadas informações que viabilizem a manutenção de seu software desenvolvido em um contexto de reutilização, auxiliando na resolução do problema da cadeia de responsabilidades de manutenção. Estas informações são mantidas por um *Mapa de Reutilização*.

O mapa de reutilização é um artefato que possui informações de contato das equipes e de quais componentes elas produziram ou consomem. Um componente possui informação de sua versão e associa-se aos componentes que ele reutiliza e pelos quais é reutilizado. Além disso, o mapa mantém as informações dos contratos firmados entre os produtores e consumidores, para cada componente reutilizado. O mapa possui um cadastro de licenças de software, e cada contrato está associado a uma licença, que descreve de forma mais genérica como um componente pode ser utilizado. O modelo conceitual do mapa de reutilização está apresentado na Fig. 2.

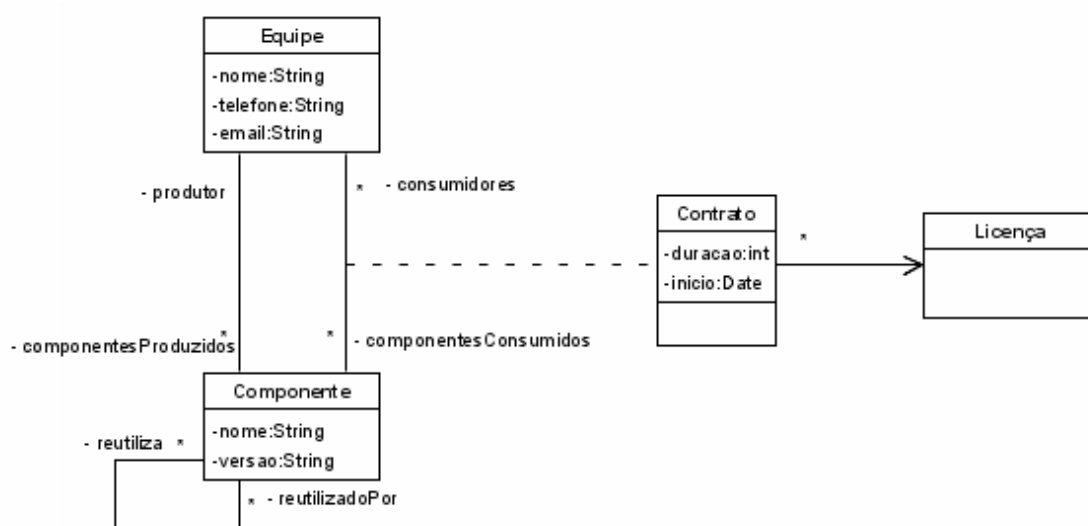


Fig. 2 - Mapa de Reutilização - Modelo conceitual

De posse destas informações, as equipes envolvidas no processo de manutenção podem determinar de quem é a responsabilidade por uma modificação específica em um componente, através da identificação do produtor e das cláusulas contratuais referentes àquele componente. Também podem consultar quem são os consumidores, identificar

quais são os que têm direito a uma nova liberação, e disseminar informações a respeito de novas versões e modificações realizadas nos componentes de interesse.

O mapa de reutilização pode ser preenchido por qualquer pessoa da equipe que desempenhe um papel gerencial do mapa. A cada nova aquisição, atualização, renegociação contratual ou liberação de um novo componente, o mapa deve ser atualizado.

Para auxiliar a preencher os dados, pode haver uma integração com o sistema de construção e liberação. Quando uma nova liberação for gerada, o sistema de construção e liberação solicita as informações de quais são os consumidores desta nova versão do componente e as disponibiliza para o mapa de reutilização. Havendo uma biblioteca de componentes, ela pode fornecer certas informações, como nome e versão dos componentes e dados dos produtores. Ela também pode ser monitorada para a coleta de informações de quem está publicando ou recuperando componentes.

Outras características desta atividade são o envio automático de notificações e a visualização das informações coletadas durante a execução.

4. Protótipo

Com base na abordagem descrita na Seção 3, foi implementada a primeira versão de um sistema de controle de modificações que possui as características identificadas como importantes para o contexto do DBC.

O Modelador SPEM foi implementado como um *plugin* do Ambiente Odyssey (2005), que é uma infra-estrutura de suporte à reutilização baseada em modelos de domínio. O Modelador SPEM é baseado na notação SPEM, e implementa a maior parte de seus elementos. Ele permite a modelagem gráfica de processos, utilizando-se os ícones sugeridos pela notação.

A Fig. 3 exibe uma tela do Modelador SPEM com um exemplo de processo modelado. Do lado esquerdo da tela, são exibidos todos os elementos SPEM que fazem parte do processo modelado. Do lado direito, é exibido um exemplo de processo modelado.

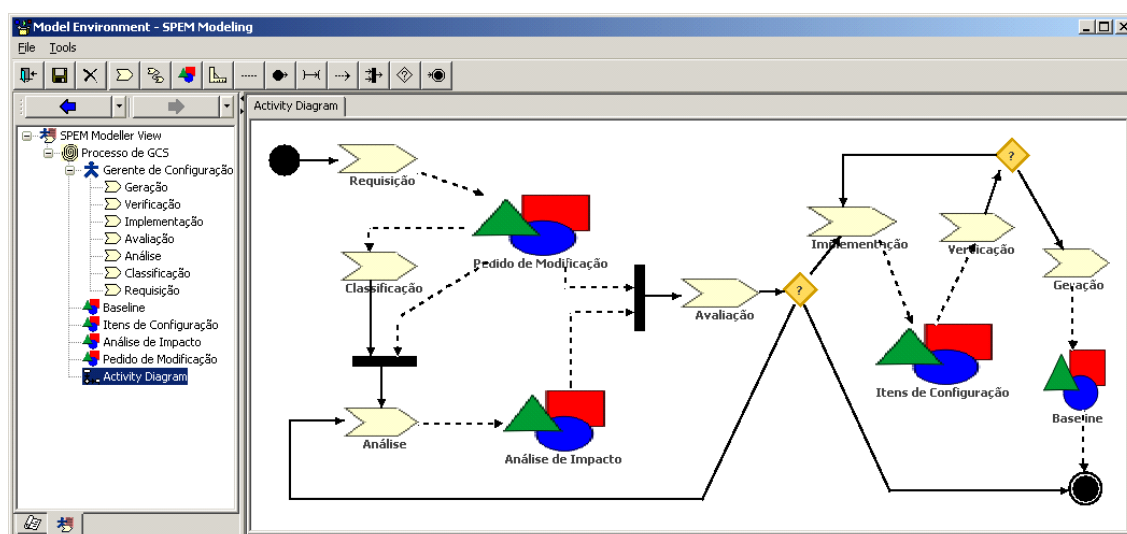


Fig. 3 – Modelador SPEM – Modelador de processos baseados na SPEM

Os ícones em forma de seta representam atividades do processo, como “Requisição”, por exemplo. Os ícones representados pelo agrupamento de um triângulo, um círculo e um quadrado representam um produto de uma atividade. Um exemplo é “Pedido de Modificação”.

As setas contínuas, entre atividades, representam fluxo de controle, e as setas tracejadas representam fluxo de controle e dados, e indicam a participação de um produto na associação. Ao selecionar o processo do lado esquerdo da tela, é possível exportá-lo em arquivo no padrão XMI.

O componente Gerador de Modelos, para criação de modelos de formulários, foi implementado para ser utilizado pela *Web*. A tela para criação destes modelos é apresentada na Fig. 4.a. Os campos são inseridos sucessivamente, e uma prévia do formulário é exibida na metade de baixo da tela. A tela para criação de campos do tipo área de texto é exibida na Fig. 4.b.

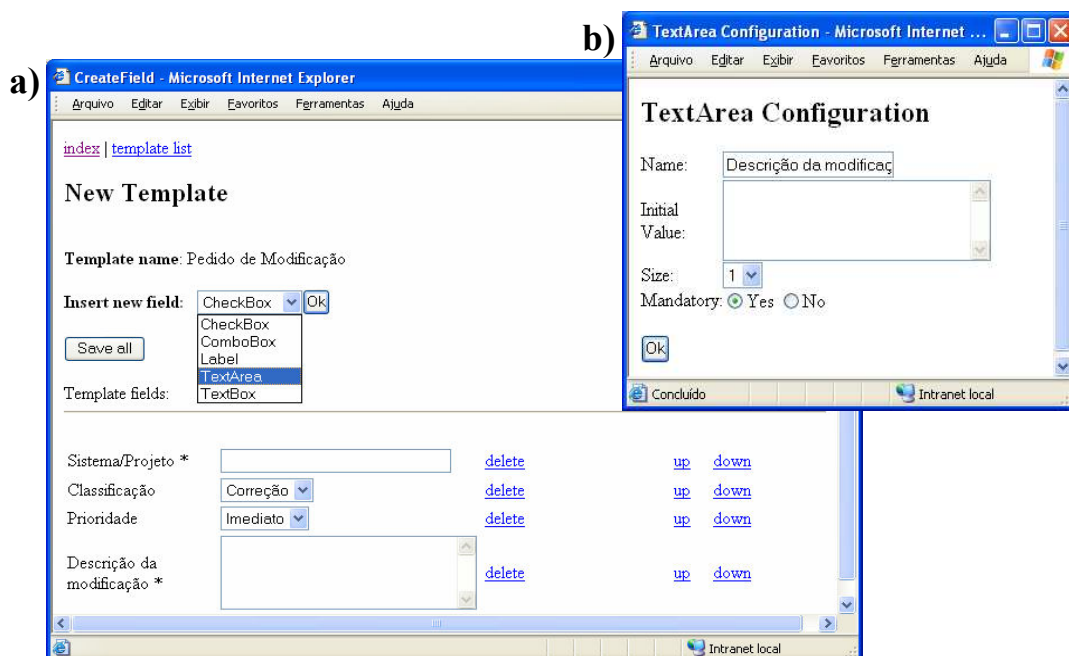


Fig. 4 – a) Criação de modelos de formulários; b) Criação de campo do tipo área de texto (text-area)

Com o processo modelado e transportado por arquivo XMI, e com os modelos de formulários criados, é feita a implantação do processo, através do componente Configurador, também implementado para *Web*. O arquivo XMI é importado, os usuários são associados aos papéis definidos no processo, e os modelos de formulários são associados aos produtos das atividades, ou às próprias atividades. Por exemplo, o modelo de formulário “Pedido de Modificação” exibido na Fig. 4.a pode ser associado ao produto de mesmo nome, presente no processo exibido na Fig. 3.

Uma vez implantado, o processo está pronto para ser executado. Sua execução é realizada pelo componente Gerenciador e pela máquina de processos Charon. Originalmente, a Charon executava processos definidos utilizando uma notação estendida do Diagrama de Atividades da UML, e teve que ser adaptada para executar também processos definidos com base na SPEM. A Fig. 5 apresenta um formulário

sendo exibido durante a execução de um processo. O modelo deste formulário é o exibido na Fig. 4.a.

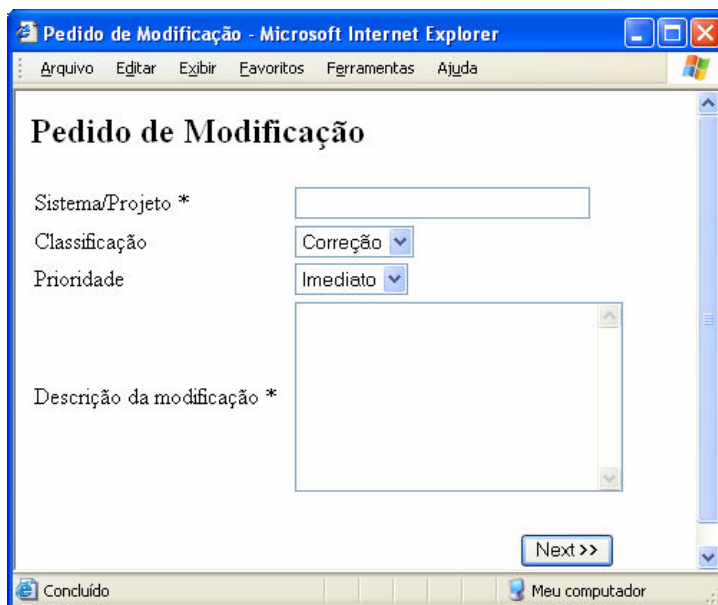


Fig. 5- Exemplo de formulário exibido durante a execução de um processo

Durante a execução do processo, as equipe produtoras e consumidoras de componentes têm acesso ao mapa de reutilização para consultar informações importantes na identificação de responsabilidades de manutenção.

5. Trabalhos relacionados

Poucos trabalhos que abordam especificamente o uso de sistemas de controle de modificações no paradigma DBC foram encontrados na literatura. A seguir são apresentados dois deles (Kwon *et al.*, 1999; Atkinson *et al.*, 2001).

Kwon *et al.* (1999) criaram o modelo de processo MwR, que dá apoio à manutenção de um sistema legado através da reutilização de software e de funcionalidades de GCS. Dentre as suas características, está uma atividade de gerenciamento de modificações e versões.

Para dar suporte ao MwR, foi desenvolvido o protótipo TERRA. Ele possui as seguintes funcionalidades: busca e registro de componentes, registro de requisições de modificação, registro de relatório de reutilização e registro de aprovação de modificações. Existem formulários próprios para cada uma destas funcionalidades, onde informações são coletadas e armazenadas no sistema.

O processo MwR está implementado no código fonte do sistema TERRA, o que o torna muito pouco flexível. No registro dos componentes, não há campos para a identificação da versão do componente e dos seus rastros para seus consumidores. Apesar do processo se preocupar com a disseminação de informações, o protótipo não implementa um mecanismo que mantenha o rastro dos componentes e de seus usuários.

Outro trabalho relacionado é o método Kobra (Atkinson *et al.*, 2001), uma abordagem para linha de produtos que abrange a área de gerência de configuração. Ele utiliza um metamodelo para representar o controle de modificações.

O modelo estabelece o conceito de conjunto de modificações (*change set*), que encapsula um conjunto de modificações realizadas no desenvolvimento de um componente. Ele provê informações essenciais para a integração destas modificações em outros componentes, ou para a troca de informações de modificações entre eles.

Neste modelo, uma modificação também pode estar relacionada a outras modificações por uma associação de causa. Assim, um rastro de modificações consecutivas pode ser registrado. Estes rastros podem ser úteis para, por exemplo, encontrar defeitos.

Porém, ele não trata de processos de controle de modificações e do problema da cadeia de responsabilidades de manutenção, e não existe uma implementação computacional que dê suporte à utilização deste método.

6. Contribuições e Trabalhos Futuros

Como contribuições deste trabalho, podemos citar:

- A identificação de deficiências dos sistemas de controle de configuração atuais quando utilizados no contexto do paradigma DBC;
- A introdução de uma abordagem que visa ajudar a contornar algumas das dificuldades existentes no controle de modificações de software desenvolvido na perspectiva de reutilização;
- A implementação de um protótipo, que facilitará a utilização da abordagem proposta.

A abordagem pode ser estendida para incluir outras funções, como a busca de informações coletadas durante o processo de controle de modificações. Por exemplo, busca por texto e por autor das requisições de modificação, entre outros. Outra funcionalidade que pode ser explorada é a criação de gráficos e relatórios gerenciais, que apresentem informações como o número de processos em andamento, o número de processos finalizados em determinado período e o tempo de execução dos processos. Outra possível extensão da abordagem refere-se à análise de impacto, sobre como auxiliar a identificação dos componentes afetados por determinada requisição de modificação.

Apesar dos indícios teóricos de que a abordagem proposta auxiliará no controle de modificações para DBC, há necessidade de realização de estudos experimentais para a verificação prática das reais contribuições do uso da abordagem e das suas deficiências. O protótipo implementado facilitará a realização destes estudos.

Agradecimentos

Os autores desejam agradecer a CAPES, ao CNPq e ao Banco Central do Brasil pelo apoio financeiro.

Referências

- Atkinson, C., Bayer, J., Bunse, C., *et al.* (2001), *Component-Based Product Line Engineering with UML*, Addison-Wesley.
- Atlassian, "JIRA - Bug tracking, issue tracking and project management software". In: <http://www.atlassian.com/software/jira/docs/latest/>, accessed in 19/08/2005.
- Barnson, M.P, Steenhagen, J. (2003), *The Bugzilla Guide - 2.17.5 Development Release*, The Bugzilla Team.
- Brown, A.W. (2000), *Large Scale Component Based Development*, Prentice Hall PTR.
- Cederqvist, P. (2003), *Version Management with CVS*, Free Software Foundation.
- Cheesman, J., Daniels, J. (2000), *UML Components: A Simple Process for Specifying Component-Based Software*, Addison-Wesley.
- CMCrossroads, "Defect Tracking Software". In: <http://resources.cmcrossroads.com/cmcrossroads/search/tabbrowse/software/1715/1/index.jsp?vf=>, accessed in 19/08/2005.
- D'Souza, D., Wills, A. (1998), *Objects, components, and frameworks with UML: The catalysis approach*, Addison Wesley.
- Dantas, C.R., Oliveira, H.L.R., Murta, L.G.P., *et al.* (2003), "Um Estudo sobre Gerência de Configuração de Software aplicada ao Desenvolvimento Baseado em Componentes". In: *Terceiro Workshop de Desenvolvimento Baseado em Componentes*, São Carlos, SP.
- Estublier, J., Leblang, D., Clemm, G., *et al.* (2002), "Impact of the research community on the field of software configuration management: summary of an impact project report", *ACM SIGSOFT Software Engineering Notes*, v. 27, n. 5 (September), pp. 31-39.
- Hatcher, E., Loughran, S. (2002), *Java Development with Ant*, Greenwich, CT, Manning Publications Company.
- IEEE (1987), *Std 1042 - IEEE Guide to Software Configuration Management*, Institute of Electrical and Electronics Engineers.
- IEEE (1990), *Std 610.12 - IEEE Standard Glossary of Software Engineering Terminology*, Institute of Electrical and Electronics Engineers.
- IEEE (1998a), *Std 828 - IEEE Standard for Software Configuration Management Plans*, Institute of Electrical and Electronics Engineers.
- IEEE (1998b), *Std 1219-1998 - IEEE Standard for Software Maintenance*, Institute of Electrical and Electronics Engineers.
- ISO (1995), *ISO 10007, Quality Management - Guidelines for Configuration Management*, International Organization for Standardization.
- Kwon, O., Shin, G., Boldyreff, C., *et al.* (1999), "Maintenance with Reuse: An Integrated Approach Based on Software Configuration Management". In: *Asia Pacific Software Engineering Conference*, pp. 507-515, Takamatsu, Japan, December.

- Larsson, M. (2000), *Applying Configuration Management Techniques to Component-Based Systems*, Licentiate Thesis, Department of Information Technology, Uppsala University, Sweden.
- Murta, L.G.P. (2002), *Charon: Uma Máquina de Processos Extensível Baseada em Agentes Inteligentes*, M.Sc., COPPE, UFRJ, Rio de Janeiro, Brasil.
- Murta, L.G.P. (2004), *Odyssey-SCM: Uma Abordagem de Gerência de Configuração de Software para o Desenvolvimento Baseado em Componentes*, Exame de Qualificação, COPPE, UFRJ, Rio de Janeiro, Brasil.
- Odyssey (2005), "Odyssey". In: <http://reuse.cos.ufrj.br/odyssey/>, accessed in 18/10/2005.
- OMG (2002), *UML 2.0 Superstructure: Final Adopted Specification*, Object Management Group.
- OMG (2005), "Meta Object Facility (MOF) Specification, version 1.4". In: <http://www.omg.org/technology/documents/formal/mof.htm>, accessed in 25/07/2005.
- OMG (2005b), *Software Process Engineering Metamodel (SPEM), Version 1.1*, Object Management Group.
- OMG (2005), "XML Metadata Interchange (XMI) Specification, Version 2.0". In: <http://www.omg.org/technology/documents/formal/xmi.htm>, accessed in 25/07/2005.
- Pressman, R.S. (2005), *Software Engineering: A Practitioner's Approach*, 6a ed., McGraw-Hill.
- Roundup, "Roundup Issue Tracker". In: <http://roundup.sourceforge.net/index.html>, accessed in 19/08/2005.
- Shaw, M., Garlan, D. (1996), *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall.
- Szyperski, C. (2002), *Component Software: Beyond object-oriented programming*, Addison-Wesley.
- W3C (1999), *HTML 4.01 Specification*, World Wide Web Consortium.
- Weber, D.W. (2001), "Requirements for an SCM Architecture to Enable Component-Based Development". In: *Proceedings of Tenth International Workshop on Software Configuration Management (SCM 10)*, Toronto, Canada, May.
- White, B.A. (2000), *Software Configuration Management Strategies and Rational ClearCase: A Practical Introduction*, Addison-Wesley.