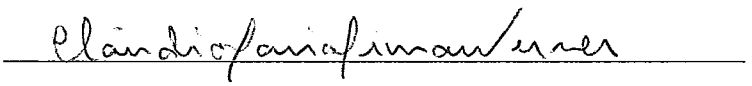
ODYSSEY-CCS: UMA ABORDAGEM PARA O CONTROLE DE MODIFICAÇÕES NO CONTEXTO DO DESENVOLVIMENTO BASEADO EM COMPONENTES

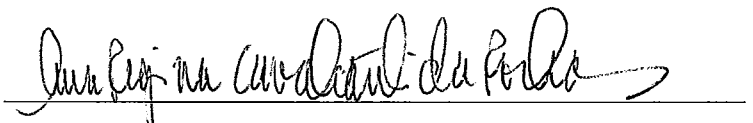
Luiz Gustavo Berno Lopes

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

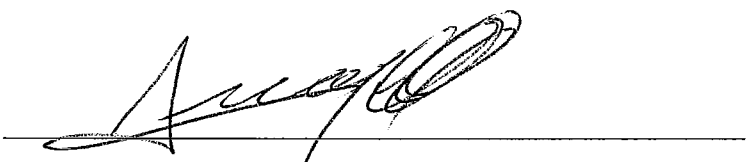
Aprovada por:



Prof. Cláudia Maria Lima Werner, D.Sc.



Prof. Ana Regina Cavalcanti da Rocha, D.Sc.



Prof. Nicolas Pierre François Anquetil, Ph.D.

RIO DE JANEIRO, RJ – BRASIL

ABRIL DE 2006

LOPES, LUIZ GUSTAVO BERNO

Odyssey-CCS: Uma Abordagem Para o Controle De Modificações No Contexto Do Desenvolvimento Baseado em Componentes [Rio de Janeiro] 2006

XII, 108 p., 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2006)

Dissertação - Universidade Federal do Rio de Janeiro, COPPE

1. Sistemas de Controle de Modificações
2. Gerência de Configuração de Software
3. Desenvolvimento Baseado em Componentes

I. COPPE/UFRJ II. Título (série)

À minha família.

Agradecimentos

À minha esposa Camila, pelo amor, carinho e incentivo durante essa caminhada, por entender minha constante falta de tempo, e pelo nosso filho que está a caminho.

Aos meus pais, pelo amor, dedicação, incentivo e confiança depositada em mim, e às minhas irmãs, pelo amor e amizade.

À professora Cláudia Werner, por me receber no grupo de braços abertos, pela orientação, paciência e dedicação que teve comigo durante todo o tempo de desenvolvimento deste trabalho. Muito obrigado.

Ao amigo Leonardo Murta, meu co-orientador informal, por me apresentar à GCS, pelos conselhos, pela disponibilidade, sempre pronto a ajudar, pela troca de idéias durante a escrita desta dissertação, pelas revisões realizadas neste trabalho e em artigos, pela paciência e também pelos churrascos da turma. Obrigado por tudo.

Aos amigos Sindi Yamamoto, Lisandro Pavie Cardoso e Paulo Cardieri, por terem participado de minha formação e pelo incentivo para realização do mestrado.

Ao amigo Bruno Santos da Cunha, pela paciência e por todo apoio dado sempre que precisei. Também aos demais amigos do Banco Central do Brasil, em especial ao Cleomar, Junio e Fernando de Abreu Faria, por acreditarem em mim e por terem concordado com a realização deste trabalho.

Aos amigos do Grupo de GCS, Hamilton, Cristine, ao Anderson e a todos os amigos do Grupo de Reutilização e da COPPE.

A todos os meus amigos, principalmente Rodrigo e Elina, por entenderem minha ausência em muitas ocasiões.

Ao Banco Central do Brasil, pelo apoio financeiro.

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

ODYSSEY-CCS: UMA ABORDAGEM PARA O CONTROLE DE MODIFICAÇÕES NO CONTEXTO DO DESENVOLVIMENTO BASEADO EM COMPONENTES

Luiz Gustavo Berno Lopes

Abril / 2006

Orientadora: Cláudia Maria Lima Werner

Programa: Engenharia de Sistemas e Computação

Um software está sujeito a modificações em qualquer etapa de seu ciclo de vida. Quando estas modificações são realizadas sem controle, diversos problemas podem surgir, como a perda de informações em artefatos alterados de forma concorrente, a falta de coordenação das pessoas envolvidas e a falta de informações sobre quais versões de artefatos estão presentes em cada cliente. No contexto do Desenvolvimento Baseado em Componentes (DBC), novas questões surgem com respeito ao controle de modificações, como a identificação do responsável por uma modificação quando esta atinge um componente reutilizado, problema chamado de cadeia de responsabilidades de manutenção, e quem será afetado por uma modificação realizada em um determinado componente.

Este trabalho apresenta uma abordagem de controle de modificações adequada ao contexto do DBC, sendo configurável com relação aos processos de controle de modificações e à coleta de informações, fornecendo informações que auxiliam a resolução do problema da cadeia de responsabilidades de manutenção, como o produtor, os consumidores e os contratos firmados entre eles para a aquisição de cada componente. Foi implementado um protótipo que demonstra a viabilidade de automação da abordagem proposta.

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

ODYSSEY-CCS: AN APPROACH FOR CHANGE CONTROL IN THE COMPONENT-BASED DEVELOPMENT CONTEXT

Luiz Gustavo Berno Lopes

April / 2006

Advisor: Cláudia Maria Lima Werner

Department: Computer and Systems Engineering

Changes may occur at anytime in a software lifecycle. When these changes are performed without control, many problems may arise, such as information loss in concurrently changed artifacts, lack of workflow coordination and lack of knowledge about which artifact version is deployed in each client. In Component-Based Development (CBD) context, new issues emerge, such as the identification of the responsible for a change when a component is impacted by a change request, named reuse chain of responsibility problem, and who is affected by a change in a specific component.

This work aims to develop a change control approach adjusted to the CBD context, providing resources to model change control processes and information gathering, a set of information that assists the solution of the reuse chain of responsibility problem, such as the producer and the consumers identification and the contracts firmed by them for each component acquisition. A prototype was implemented to demonstrate the viability to automate the proposed approach.

Índice:

Índice de Figuras

Índice de Tabelas

Capítulo 1 - Introdução

1.1 - Motivação

Um software está continuamente sujeito a modificações (LEHMAN, 1996; LEON, 2000; PRESSMAN, 2005), que podem ser motivadas por diversos fatores. Exemplos são a correção de erros encontrados por desenvolvedores e usuários, o surgimento de novas tecnologias, as alterações no ambiente de operação do software, o desenvolvimento incremental de software, entre outros.

Segundo LEHMAN (1980), quanto mais os requisitos do software dependerem do mundo real, maior será a probabilidade do software ser modificado, já que o mundo real é dinâmico e possui incertezas e conceitos que não são totalmente compreendidos. Por exemplo, alguns sistemas de software do mercado financeiro são dependentes da legislação tributária. Assim, alterações nestas leis tributárias podem implicar na necessidade de modificação destes sistemas. Alterações do sistema monetário do país, o que já ocorreu algumas vezes no Brasil, é um outro exemplo de fato que pode implicar em modificações nestes sistemas. Portanto, mesmo os sistemas de software que estão funcionando perfeitamente podem ser alvo de modificação.

Quando as modificações nos sistemas de software são realizadas sem controle, diversos problemas podem surgir (LEON, 2000; PRESSMAN, 2005). Por exemplo, com a falta de coordenação das pessoas envolvidas, uma mesma tarefa de modificação pode ser realizada mais de uma vez por pessoas diferentes, causando desperdício de recursos. Mais ainda, as modificações podem ser realizadas em uma ordem de prioridade errada e pode ocorrer demora no atendimento às requisições de modificação, entre outros.

Além disso, sem controle, o gerente do projeto perde informações, como o número de modificações em andamento, finalizadas e em espera, quais foram os motivos de cada modificação, quem foi que as realizou e quais artefatos foram afetados. Os artefatos também podem ser modificados de forma concorrente, podendo ser sobrescritos, causando perda de informações. Se não forem mantidas as informações sobre quais versões de artefatos estão presentes em cada cliente, quando um determinado cliente requisitar uma modificação, será muito difícil atender à sua solicitação. Estes são apenas alguns dos vários problemas que podem ocorrer.

Para permitir que o software evolua de maneira controlada, existe a Gerência de Configuração de Software (GCS) (ESTUBLIER, 2000). Ela deve ser aplicada tanto durante o desenvolvimento quanto durante a manutenção de software (LEON, 2000; PFLEEGER, 2004; PRESSMAN, 2005). A IEEE, através da norma 610.12 (1990), define a GCS como uma disciplina responsável pela identificação e documentação das características físicas e funcionais de um Item de Configuração[1] (IC), pelo controle das modificações sobre estas características e pelo armazenamento, acompanhamento e verificação da conformidade destas modificações com os requisitos especificados.

A GCS é considerada essencial para o sucesso dos projetos de desenvolvimento de software (ESTUBLIER *et al.*, 2005), sendo contemplada por uma área de processo do modelo de maturidade CMMI (CHRISSIS *et al.*, 2003) e por um processo do Modelo de Referência do MPS.BR (SOFTEX, 2005). Além disso, há várias normas internacionais que abordam a GCS, como a IEEE Std 828 (2005), a ANSI/EIA 649 (1998), a ISO 10007 (1995a) e a ISO/IEC 12207 (1995b). Uma visão geral da GCS é apresentada no Capítulo 2 deste trabalho.

Uma das preocupações da GCS está relacionada ao controle de modificações vista sob a perspectiva gerencial. Devem ser definidas quais são as atividades a serem executadas desde o recebimento de uma requisição de modificação (RM) até a finalização desta modificação, quais são as pessoas responsáveis pela execução de cada uma destas atividades, quando e como elas devem ser executadas, quais são as informações necessárias para a sua execução, quais são as informações geradas, quem deve receber e ter acesso a estas informações, entre outros. Todas estas definições devem estar registradas no Plano de Gerência de Configuração de Software do projeto (ISO, 1995a; EIA, 1998; IEEE, 2005).

Um sistema de controle de modificações é um tipo de software que auxilia a aplicação deste subconjunto de características da GCS, atuando como um maestro das equipes envolvidas com as modificações. Ele auxilia a comunicação, a coordenação do trabalho e mantém as informações sobre as modificações realizadas, como, por exemplo, quem fez o quê, quando, onde, por que e como.

O uso de um sistema de controle de modificações também auxilia a atingir alguns objetivos de modelos para a melhoria de processos de software, como o CMMI e

---

[1] Item de Configuração (IC): Agregação de software, hardware ou ambos, passível de GCS e tratado como um único item no processo de gerência de configuração (IEEE, 1990).

o MPS.BR. Com relação ao CMMI, o uso destes sistemas facilita que sejam cumpridas as práticas específicas 1.2 (Estabelecer linha básica), 2.1 (Acompanhar requisições de modificação) e 2.2 (Controlar itens de configuração) da área de processo chamada Gerência de Configuração. Com relação ao MPS.BR, o uso destes sistemas auxilia a obtenção de alguns resultados do processo Gerência de Configuração (GCO), como: parte do GCO 4, que corresponde ao estabelecimento de um sistema de GCS, parte do GCO 7, no que se refere ao acompanhamento e controle das modificações, parte do GCO 8, no que diz respeito à disponibilização das modificações para todos os envolvidos e parte do GCO 9, com relação ao registro, relato e análise de impacto de requisições de modificação.

A maneira como a GCS deve ser aplicada é especificada para cada projeto de software, adequando-se às suas características, como o tamanho e o paradigma utilizado no desenvolvimento do software. Por exemplo, construção de software no paradigma do Desenvolvimento Baseado em Componentes (DBC) possui suas particularidades.

DBC é uma disciplina de desenvolvimento de sistemas através da reutilização de componentes bem definidos, produzidos independentemente (BROWN, 2000; LARSSON, 2000).

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 reutilizáveis. Há as equipes que desenvolvem software reutilizando componentes, chamadas de equipes consumidoras. Há ainda as que reutilizam componentes para desenvolver componentes mais complexos, chamadas de equipes híbridas, que atuam tanto como produtoras quanto como consumidoras de componentes.

Nesta configuração de trabalho, os pedidos de modificação deixam de ser apenas uma relação entre o cliente da aplicação e o seu desenvolvedor, como acontece no desenvolvimento convencional de software. Ela passa a ser entre o cliente da aplicação, a equipe consumidora que a desenvolveu e as equipes produtoras dos componentes reutilizados. Este cenário torna-se mais complexo quando os componentes produzidos são compostos de outros componentes, e neste caso as equipes híbridas também se envolvem nas atividades de modificação do software.

Várias questões surgem em um contexto de intensa reutilização e composição de componentes, no que se refere ao controle de modificações. Por exemplo, deve-se identificar os responsáveis por cada modificação, que pode ser a equipe consumidora,

uma equipe produtora ou uma equipe híbrida. É necessário também saber quem são os consumidores de cada componente, para que eles possam ser contatados e informados sobre defeitos encontrados, melhorias previstas e novas versões dos componentes disponibilizadas, entre outros. Além disso, os consumidores podem ter um nível de importância diferenciado para o produtor, por exemplo, por questões estratégicas ou pela quantidade de componentes adquiridos. Os consumidores também podem participar do controle de modificações, por exemplo, opinando sobre quais são as principais carências dos componentes consumidos.

Os sistemas de controle de modificações encontrados no mercado e na literatura não são adequados para utilização no contexto de DBC, pois não fornecem os mecanismos necessários para auxiliar a resolução das questões aqui discutidas. A motivação deste trabalho consiste na carência de apoio à aplicação da GCS no contexto do DBC, no que diz respeito aos sistemas de controle de modificações.

1.2 - Caracterização do problema

No contexto do DBC, ao receber uma RM de um cliente, a equipe consumidora pode identificar que um componente reutilizado, produzido por outra equipe, precisa de manutenção. Ela deverá analisar e identificar quem realizará 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 aquisição 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[2] do componente.

É possível que o produtor seja uma equipe híbrida e tenha reutilizado um componente de terceiros na construção de seu componente. Mais ainda, ela pode ter identificado que a modificação requisitada 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

[2] Liberação (*release*) é a notificação e distribuição formal de uma versão aprovada do software para o cliente (IEEE, 2005).

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 RM, 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 (ISO, 1995a; EIA, 1998; KWON *et al.*, 1999; IEEE, 2005).

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 necessária novamente a disseminação desta informação para os seus consumidores. Em resumo, componentes podem ser compostos de outros componentes (APPERLY *et al.*, 2003), 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 deve conhecer o contrato firmado com seus consumidores para decidir em que condições a atualização do componente deverá ocorrer. Dependendo do tipo da modificação e das cláusulas contratuais, a atualização poderá ser feita de forma gratuita ou não.

Outra situação que pode ocorrer é o produtor decidir pela modificação de um componente à revelia dos consumidores, por exemplo, por questões estratégicas. 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 liberação do componente.

Existe também o caso no qual um consumidor utiliza dois ou mais componentes que devem ser utilizados em conjunto, um comunicando-se com o outro, sendo que eles são fornecidos por diferentes produtores. Se existir alguma incompatibilidade entre as interfaces dos componentes, deve-se identificar quem será o responsável pela modificação necessária, se será um dos produtores ou o próprio consumidor. É

importante que o contrato seja detalhado o suficiente para auxiliar a identificação do responsável pela modificação neste caso. Conhecendo os produtores dos componentes, é possível também que o consumidor os contate para negociar a realização da modificação necessária.

Neste contexto, torna-se necessária a manutenção dos rastros dos componentes com relação a seus produtores e consumidores, ou seja, para cada liberação 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* (BRERETON, BUDGEN, 2000; MURTA, 2004) não é considerado pelos atuais sistemas de controle de modificação.

Um outro problema refere-se ao fato de processos de controle de modificações aplicados ao DBC (KWON *et al.*, 1999; ATKINSON *et al.*, 2000; DANTAS *et al.*, 2003) ainda serem imaturos e não possuírem diretrizes específicas das principais normas de GCS (ISO, 1995a; EIA, 1998; IEEE, 1998). Isso sugere a necessidade de configuração destes processos, recurso não fornecido por muitos dos sistemas de controle de modificações. É válido ressaltar que esta necessidade não é exclusiva ao contexto do DBC, já que os processos de controle de modificações convencionais também devem poder ser adaptados às necessidades específicas das empresas e dos projetos.

Além disso, na maioria dos sistemas que fornecem este recurso não é possível modelar o processo graficamente, o que facilitaria esta tarefa, e também não é possível modelar explicitamente os artefatos consumidos e produzidos pelas atividades. A maioria também não permite a criação de subatividades, o que seria interessante para processos mais extensos. Portanto, os processos de controle de modificações poderiam ser modelados de forma mais completa e compreensível.

1.3 - Objetivo

Tendo em vista os problemas e necessidades citados anteriormente, o objetivo deste trabalho é fornecer uma abordagem para o controle de modificações que seja mais adequada ao contexto do DBC, sendo configurável com relação aos processos de controle de modificações e à coleta de informações, que auxilie a resolução do problema da cadeia de responsabilidades de manutenção, e que apresente uma implementação computacional que demonstre a sua viabilidade de automação.

A abordagem deve englobar mecanismos de modelagem gráfica de processos de controle de modificações, facilitando o trabalho do gerente de configuração. Deve ser utilizada uma linguagem para modelagem de processos, que possibilite expressar o processo de controle de modificações de forma mais completa em comparação com as abordagens de controle de modificações atuais. Além disso, a linguagem de processo utilizada deve ser padronizada, permitindo que os processos modelados sejam compreendidos por um número maior de pessoas e sistemas.

A implementação computacional da abordagem deve apoiar a execução do processo modelado, auxiliando a coordenação do trabalho envolvido nas modificações. Assim, as atividades relacionadas às modificações devem ser atribuídas às pessoas responsáveis, no momento correto, de acordo com o processo de controle de modificações definido. Deve ser utilizado um mecanismo de notificação, responsável por disseminar as informações relevantes a cada pessoa envolvida.

A coleta de informações também deve ser configurável. Devem poder ser definidas quais informações serão requisitadas em cada momento do processo de controle de modificações, e quem são os responsáveis pelo preenchimento destas.

Também devem ser mantidas e disponibilizadas as informações relativas à reutilização de componentes, para auxiliar a resolução do problema da cadeia de responsabilidades de manutenção.

1.4 - Organização

Esta dissertação é composta de cinco capítulos, incluindo este primeiro.

O Capítulo 2 apresenta a análise de algumas abordagens e sistemas de controle de modificações encontrados na literatura, comparados segundo critérios definidos no próprio capítulo. São analisados tanto sistemas de código aberto quanto sistemas comerciais. Algumas abordagens de DBC que englobam o controle de modificações também são discutidas.

No Capítulo 3, é apresentada a abordagem proposta neste trabalho, que visa definir e implementar um sistema de controle de modificações apropriado para o contexto do DBC. Este sistema deve atender aos critérios apresentados no Capítulo 2.

No Capítulo 4, é apresentado o protótipo implementado, que mostra a viabilidade de automação das idéias propostas na abordagem. São fornecidos detalhes técnicos e de utilização deste protótipo, e também são apresentados dois casos de utilização do protótipo por outras abordagens, a Odyssey-WI e a Odyssey-VCS.

No Capítulo 5, são discutidas as contribuições e as limitações deste trabalho e são apresentados possíveis trabalhos futuros.

Capítulo 2 - Apoio ao Controle de Modificações em Software

2.1 - Introdução

Este capítulo apresenta alguns sistemas e abordagens para o controle de modificações em software descritos na literatura. Há uma grande quantidade destes sistemas, tanto de código aberto quanto comerciais (CMCROSSROADS, 2006). Os sistemas de código aberto selecionados para análise são aqueles que se mostram mais utilizados e mais referenciados na literatura, ou que possuem funcionalidades interessantes e diferentes. Os sistemas comerciais analisados são aqueles com maior participação no mercado (WEBSTER, 2004), com exceção de alguns que possuem documentação restrita, impossibilitando sua análise.

A maioria destes sistemas foi projetada para o controle de modificações de software desenvolvido em qualquer paradigma. Entretanto, como software desenvolvido em DBC possui um processo de desenvolvimento diferente do convencional (LARSSON, 2000), é natural que o processo de controle de modificações também possua particularidades e tenha que ser adaptado. Alguns dos sistemas comerciais são flexíveis o suficiente para que o processo de controle de modificações seja adaptado às necessidades dos projetos, mas eles não fornecem todas as informações necessárias para o controle de modificações no contexto do DBC.

Na Seção 2.2 deste capítulo, é apresentada uma visão geral da GCS, em termos de suas funções e sistemas, para contextualização dos sistemas de controle de modificações. Na Seção 2.3, são analisados alguns sistemas de controle de modificações de código aberto e comerciais. Existem também algumas abordagens de apoio ao DBC que se preocupam com a evolução dos componentes e sistemas baseados em componentes. Na Seção 2.4, estas abordagens são analisadas quanto ao apoio fornecido ao controle de modificações. Na Seção 2.5, os sistemas e abordagens analisados nas Seções 2.3 e 2.4 são comparados segundo um conjunto de critérios, identificados a partir da análise das contribuições e limitações dos próprios sistemas e abordagens descritos. Na Seção 2.6, é discutida a forma de modelagem dos processos de controle de modificações dos sistemas atuais, e, na Seção 2.7, são apresentadas as considerações finais deste capítulo.

2.2 - Gerência de Configuração de Software

A GCS pode ser tratada sob diferentes perspectivas em função do papel exercido pelo participante do processo de desenvolvimento de software (ASKLUND, BENDIX, 2002). Na perspectiva gerencial, a GCS é dividida em cinco funções principais, que são (ISO, 1995a; EIA, 1998; SCOTT, NISSE, 2001; IEEE, 2005): Identificação da Configuração, Controle da Configuração, Acompanhamento da Configuração (*Configuration Status Accounting*), Auditoria e Revisão da Configuração e Gerenciamento de Liberação (*Release*).

A função de Identificação da Configuração engloba a seleção dos elementos passíveis de GCS, ou seja, os itens de configuração (IC), a definição do esquema de nomenclatura para identificação única dos ICs, a documentação física e funcional dos ICs e o estabelecimento e identificação de linha básica[3] (*baseline*).

A função de Controle da Configuração possui um processo para controle de modificações nos ICs presentes nas linhas básicas. As seguintes atividades devem ser contempladas: Identificação e documentação do motivo da modificação, análise e avaliação da requisição de modificação, aprovação ou não da modificação, implementação, verificação e liberação da modificação.

A função de Acompanhamento da Configuração envolve a aquisição e disseminação das informações coletadas durante as tarefas de GCS, o controle de acesso às informações armazenadas, a coleta de métricas de GCS e a geração de relatórios gerenciais.

A função de Auditoria e Revisão da Configuração engloba a verificação funcional da linha básica, assegurando que a mesma cumpre corretamente o que foi especificado, e também a verificação física, para certificar que a mesma é completa em relação ao que foi acordado em contrato. A auditoria deve acontecer ao menos antes da liberação de uma linha básica do produto.

A função de Gerenciamento de Liberação é responsável pelos procedimentos de controle de construção (*building*), liberação (*release*) e entrega de produtos. A construção do produto significa combinar e transformar um conjunto de versões de itens em um programa executável, que será entregue aos usuários, que podem ser externos ou

---

[3] Linha básica (*baseline*) é um conjunto de ICs aprovados formalmente que servem de base para as etapas seguintes de desenvolvimento, e que devem ser alterados somente segundo os procedimentos formais de controle de modificações (IEEE, 1990).

internos à organização. Os produtos a serem entregues devem conter, além do programa executável, informações suficientes para que o consumidor possa compreendê-lo e utilizá-lo. Em geral, são providas informações como manual de instalação e de uso, novidades contempladas na nova versão, quais são os problemas já identificados e ainda não corrigidos, quais problemas foram corrigidos, entre outras. A GCS deve verificar se o produto a ser entregue está completo e pronto para ser utilizado pelo consumidor.

Contudo, sob a perspectiva de desenvolvimento, a GCS é apoiada por três tipos de sistemas principais: Controle de Modificações, Controle de Versões e Controle de Construção e Liberação.

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.*, 2005) é um exemplo deste tipo de sistema.

O sistema de controle de versões permite que os ICs sejam identificados, segundo critérios estabelecidos pela função de Identificação da Configuração, e que eles evoluam de forma distribuída e concorrente, porém disciplinada. Um exemplo deste tipo de sistema 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, LOUGHRAN, 2002).

Este trabalho, no contexto da GCS, se enquadra nas funções de Controle e Acompanhamento da Configuração, na perspectiva gerencial, e nos Sistemas de Controle de Modificações, na perspectiva de desenvolvimento. A seguir, são discutidos alguns sistemas e abordagens para o controle de modificações de software, incluindo abordagens específicas para o DBC.

2.3 - Sistemas de controle de modificações

O sistema de controle de modificações apóia principalmente a execução das funções da GCS de Controle e Acompanhamento da Configuração. Ele provê recursos para a execução do processo de controle de modificações definido para o projeto,

permitindo a inserção de requisições de modificação (RM) e o acompanhamento de todo o seu ciclo de vida. Durante a execução do processo, informações são coletadas, armazenadas e disseminadas aos participantes interessados e autorizados. Estas informações são de grande valia para o gerenciamento do projeto, permitindo a identificação dos artefatos modificados, a análise das causas das modificações e a coleta de métricas para o acompanhamento do projeto (CRNKOVIC, WILLFÖR, 1998). Este tipo de sistema também gera relatórios e gráficos para análise das informações coletadas.

As normas IEEE Std 828 (2005), ISO 10007 (1995a) e EIA 649 (1998) definem como principais atividades de um processo de controle de modificações: 1) Identificação e documentação da necessidade de modificação; 2) Análise e avaliação da requisição de modificação; 3) Aprovação ou rejeição da requisição de modificação; 4) Implementação e verificação da modificação. Estas normas fornecem diretrizes para a elaboração do processo de modificação, que deverá ser definido para cada empresa e projeto no Plano de Gerência de Configuração de Software.

Encontramos na literatura diversos sistemas de controle de modificações disponíveis para o desenvolvimento convencional de software (CMCROSSROADS, 2006). Estes sistemas são conhecidos por diversos nomes, como Sistemas de Rastreamento de Defeitos (*Bug/Defect Tracking Systems*), Sistemas de Rastreamento de Problemas (*Problem Tracking Systems*), Sistemas de Rastreamento de Questões (*Issue Tracking Systems*) e Sistemas de Controle de Modificações, entre outros. Especificamente para DBC, são encontrados poucos trabalhos e sistemas nesta área.

A seguir, são apresentados alguns dos sistemas de controle de modificações encontrados na literatura, escolhidos por serem muito utilizados em projetos de código aberto, por serem produtos das empresas líderes de mercado na área de GCS (WEBSTER, 2004), ou por possuírem características interessantes e diferentes. Também foi levada em consideração a disponibilidade de documentação para a análise dos sistemas.

2.3.1 - Bugzilla

O Bugzilla (BARNSON *et al.*, 2005) é um sistema de código aberto (OSI, 2006) para controle de modificações muito utilizado pela comunidade que desenvolve sistemas de código aberto, estando instalado em centenas de empresas (BUGZILLA,

2006). Este sistema possui interface para Internet, o que facilita sua utilização por equipes geograficamente distribuídas, situação freqüente nos projetos de código aberto.

No Bugzilla, todas as alterações solicitadas são chamadas de *Bugs*, mesmo que seja requisitada uma evolução ou uma adaptação do software. O processo de controle de modificações é definido através de um diagrama de transição de estados, implementado no código-fonte do software. Portanto, não é trivial a configuração deste processo. Ele é apresentado na Fig. 2-1, e exibe todos os estados que uma RM pode assumir. Cada elipse representa um estado, e cada seta representa qual o próximo estado que a RM pode assumir a partir do estado corrente. A cada seta é associado um rótulo, que representa o evento que dispara a mudança de estado. Quando a RM assume o estado *FECHADO*, o processo chega ao fim. Porém, nota-se que uma RM no estado *FECHADO* pode ir para o estado *RE-ABERTO*.

Analisando este processo definido, percebe-se a ausência de atividades importantes do processo de controle de modificações, definidas em normas internacionais (ISO, 1995a; EIA, 1998; IEEE, 2005), como a Análise e a Avaliação da RM. Assim, para inclusão destas atividades, o processo deve ser remodelado. Contudo, o Bugzilla não provê recursos de apoio à modelagem do processo.

O Bugzilla também possui dois formulários para coleta de informações, um para inserção de uma nova RM e outro para visualização e modificação das informações sobre as RMs. Estes formulários podem ter a aparência alterada, mas não há recursos para inclusão de novos campos. Para isso, o código-fonte do Bugzilla deve ser alterado. Também é possível criar grupos de usuários e definir quem pode realizar cada mudança de estado da RM. Porém, esta tarefa também não é apoiada pelo sistema, sendo necessária a alteração de código-fonte.

Uma característica interessante do sistema é a criação de vínculos (*links*) automaticamente para determinados trechos de texto inseridos, que referenciem modificações, comentários e URL's. Por exemplo, quando um usuário insere um comentário fazendo referência a uma RM, o sistema cria automaticamente um vínculo entre o identificador da RM inserido e o conteúdo da RM referenciada.

O sistema também possui recursos para manter informações sobre dependências entre RM. Assim, pode-se definir quais RMs bloqueiam a resolução de uma determinada RM. Além disso, ele dissemina informações por e-mail, exporta e importa listas de RMs e gera gráficos e relatórios para análise das informações armazenadas.

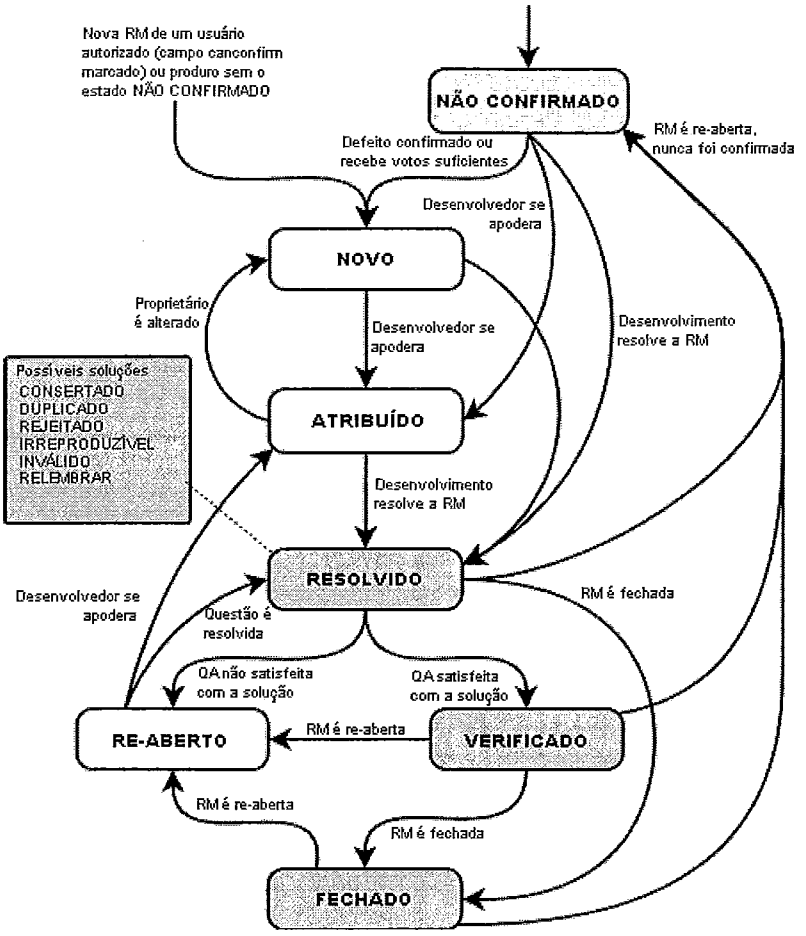


Fig. 2-1 - Diagrama de transição de estados do Bugzilla (BARNSON et al., 2005)

O Bugzilla é organizado em projetos compostos por componentes. Para cada componente, pode-se atribuir um responsável, que no contexto de DBC pode ser identificado como o produtor do componente. Porém, não há informações sobre os consumidores destes componentes, tampouco dos contratos para utilização destes.

Existe um software chamado Scmbug (MKGNU.NET, 2006) que integra o Bugzilla ao sistema de controle de versões CVS (CEDERQVIST, 2003). O Scmbug procura na base de dados do CVS mensagens de *check-in* inseridas pelos usuários que contenham um identificador para alguma das RMs cadastradas no Bugzilla. Estas mensagens são adicionadas aos comentários das RMs referenciadas, podendo ser acessadas a partir da tela de visualização da RM. Dessa forma, o Scmbug consegue agrupar logicamente os artefatos alterados a cada RM do Bugzilla.

Porém, esta técnica depende da disciplina de cada mantenedor em preencher os comentários do *check-in*, o que nem sempre ocorre, e que ele saiba qual é o identificador da RM na qual está trabalhando. Dessa forma, ao fazer o *check-in*, ele deverá sair do ambiente do CVS para consultar a RM, no Bugzilla, e depois retornar ao

ambiente do CVS para retomar a transação de *check-in*. Além de causar a interrupção do trabalho do mantenedor, esta técnica é bastante suscetível a erros. Caso ocorram estes problemas, eles serão detectados pela função de auditoria da GCS.

Este sistema não possui flexibilidade para configuração de formulários e do processo de controle de modificações, que é único para todos os projetos, e não se preocupa em manter informações sobre a reutilização de componentes.

2.3.2 - Mantis

O Mantis (2006) é um sistema de controle de modificações semelhante ao Bugzilla, também muito utilizado pela comunidade de software de código aberto. Ele apresenta um conjunto de funcionalidades básicas de apoio ao processo de controle de modificações, mas com algumas particularidades interessantes, discutidas a seguir.

O Mantis facilita o acesso às RMs de interesse de cada usuário. Após se autenticar no sistema, o usuário pode acessar uma página personalizada, onde são apresentados ao usuário alguns quadros contendo, cada um, uma lista de RMs filtradas pelos seguintes critérios: RMs relatadas pelo usuário, RMs atribuídas ao usuário, RMs modificadas recentemente, RMs ainda não atribuídas a nenhum usuário e RMs monitoradas pelo usuário. Ou seja, o usuário recebe de forma concentrada as principais RMs de seu interesse, sem precisar fazer buscas no sistema.

Outra característica do Mantis é permitir a configuração do formulário de coleta de informações, sendo possível inserir novos campos, inclusive podendo-se associar a eles expressões regulares para validação das informações inseridas. Porém, esta configuração não pode ser feita para projetos específicos, ela é aplicada a todos os projetos mantidos pelo Mantis. Além disso, o processo de controle de modificações não é configurável.

É introduzido também um serviço de mensagens para apoiar a disseminação de informações e a comunicação entre os usuários do sistema. Podem ser enviadas mensagens públicas ou privadas, que são exibidas na página inicial do sistema, após a autenticação dos usuários. Além disso, o Mantis possui um fórum para troca de mensagens. Estas funcionalidades são bastante interessantes, já que o controle de modificações é realizado em equipe e a troca de informações é fundamental.

Além disso, é possível manter um rastreamento entre RMs através da definição de relacionamentos entre elas. Os relacionamentos podem indicar o parentesco, duplicidade ou simplesmente que as RMs possuem uma relação qualquer. No caso da

relação de parentesco, uma RM pode ser subdividida em outras RMs, que manterão uma relação de pai e filhos, sendo que a análise individual de uma destas RMs indicará o seu relacionamento com as outras. Esta funcionalidade é interessante quando uma RM possui uma grande quantidade de tarefas embutidas, que podem ser quebradas em RMs menores e mais concisas, facilitando o trabalho de controle das modificações e de manutenção.

O Mantis também pode operar de forma integrada ao sistema de controle de versões CVS (CEDERQVIST, 2003), da mesma forma que o Scmbug faz para o Bugzilla, mas sem a necessidade de instalação de outro software. Assim, o Mantis também consegue identificar quais foram os artefatos alterados no contexto de cada RM, através da leitura dos identificadores de RMs inseridos pelos usuários nas mensagens de *check-in* do CVS.

2.3.3 - GConf

Um outro sistema existente é o GConf (FIGUEIREDO *et al.*, 2004). Ele implementa um processo de GCS completo, incluindo as atividades de identificação da configuração, controle da configuração, relato da situação, auditoria da configuração e gerência da liberação e entrega. Portanto, este sistema também implementa as funcionalidades de um sistema de controle de modificações. Ele é voltado ao controle de artefatos de granularidade grossa, como documentos, por exemplo. Por estar inserido no contexto de um Ambiente de Desenvolvimento de Software Orientado a Organização (ADSOrg) (VILLELA *et al.*, 2003), ele possui uma característica muito interessante, que é a integração com uma base de conhecimentos, onde são armazenadas lições aprendidas, melhores práticas, idéias, dúvidas e problemas encontrados durante a execução do processo de GCS.

Uma limitação do GConf é o fato de uma RM só poder ser gerada quando se sabe qual o IC que deve ser modificado, o que nem sempre é verdade. Em geral, uma RM está associada a uma modificação de mais alto nível de abstração, que pode demandar a alteração de vários ICs, que somente serão identificados após análise da RM pelos analistas de sistema. Por exemplo, quando um cliente encontra um erro no software e lança uma RM, ele não sabe quais são os ICs que serão impactados, que podem ser vários, como modelos de análise, projeto e arquivos de código-fonte.

Uma outra limitação do sistema é o fato de não ser possível cadastrar mais de uma RM sobre um mesmo IC, ao mesmo tempo. Por exemplo, se dois clientes

encontram problemas diferentes que afetam um mesmo IC, e um deles inclui uma RM, o outro deverá esperar que esta RM seja finalizada para incluir a sua RM.

O GConf se mostra mais adequado ao controle de modificações requisitadas internamente pela equipe de desenvolvimento e manutenção do software, devido às limitações descritas anteriormente. As RMs originadas pelos clientes podem afetar vários artefatos, geralmente desconhecidos por eles, como arquivos de código-fonte e diagramas de classes. Os clientes possuem uma visão de mais alto nível de abstração do software, detectando problemas e possíveis melhorias do software, mas não conseguem detectar os ICs impactados pelas modificações requisitadas.

2.3.4 - IBM Rational ClearQuest

A IBM possui um sistema de controle de modificações chamado IBM Rational ClearQuest (WHITE, 2000), que pode trabalhar de forma integrada a um sistema de controle de versões, o IBM Rational ClearCase. Em uma análise de mercado realizada em 2003 pela IDC (WEBSTER, 2004), a IBM foi apontada como sendo a líder em vendas de sistemas de GCS, com uma fatia de 36,7% do mercado.

O Rational ClearQuest é composto por uma aplicação cliente, um conjunto de ferramentas administrativas, um módulo central e uma base de dados. A aplicação cliente possui uma versão para Internet, e as ferramentas de administração são específicas para o sistema operacional MS Windows.

Os processos de controle de modificações são modelados através de diagramas de transição de estados. O sistema disponibiliza alguns modelos de processo pré-definidos, mas também possui uma grande flexibilidade para configuração dos processos, permitindo a configuração destes para cada projeto específico. Assim, pode-se alterar e criar novos estados e ações que disparam a mudança de um estado para outro.

A modelagem dos diagramas de estados não pode ser feita graficamente. Ela é realizada através de uma matriz de transição de estados, onde as linhas representam os estados de destino e as colunas os estados de origem, sendo que na intersecção é configurada a ação que dispara a transição entre os estados. Nota-se que esta matriz torna-se difícil de ser compreendida quando o diagrama possui muitos estados. Por exemplo, um diagrama com dez estados é representado por uma matriz 10x10, com cem intersecções entre estados. Para processos extensos, é necessária uma estratégia que

permita a visualização do processo em diferentes níveis de abstração, recurso não fornecido por este sistema.

O sistema também não fornece auxílio à verificação da estrutura dos processos de controle de modificações modelados. Se for modelado um processo com um estado *adiado*, por exemplo, e este estado não possuir nenhum responsável associado, as RMs que atingirem este estado serão acumuladas e nunca serão resolvidas.

Uma das funcionalidades do ClearQuest é a possibilidade de configurar responsabilidades no processo, através da associação de grupos de usuários a cada ação de mudança de estado, restringindo aos usuários destes grupos a execução destas ações. Além disso, o sistema possui um mecanismo de notificação por e-mails, que são enviados quando ações satisfazem determinadas regras.

Outra característica do sistema é a possibilidade de customizar os formulários utilizados para a exibição e coleta de informações durante a execução do processo. Pode-se, graficamente, configurar o formulário, incluindo, excluindo e alterando seus campos.

O ClearQuest permite ainda a geração de consultas e gráficos para análise das informações sobre as modificações. Além disso, ele permite a importação e exportação de informações sobre as RMs, possibilitando a troca de dados entre diferentes instalações do ClearQuest e com outros sistemas de controle de modificações. Porém, pode ser trabalhoso colocar os dados de um outro sistema no formato de importação do ClearQuest.

A IBM Rational define também o UCM (WHITE, 2000), um processo para gerenciamento de modificações que integra o ClearCase e o ClearQuest. Quando ele é utilizado, é possível manter o rastro entre uma modificação lógica e as versões dos arquivos geradas no contexto da modificação, o que eleva o nível de abstração de arquivos para atividades, facilitando a compreensão das modificações.

O ClearQuest é um sistema muito flexível quanto à modelagem de processos de controle de modificações e formulários, apesar de não ser possível modelar algumas características de processos, como os produtos gerados e consumidos em cada atividade. Ele possui uma grande quantidade de recursos, como a possibilidade de definição dos grupos de usuários responsáveis por cada atividade dos processos, permite que processos distintos sejam associados a tipos diferentes de RMs, entre outros. Como principais pontos fracos, destacamos a inexistência de recursos gráficos para modelagem de processos, a não utilização de uma linguagem padronizada de definição

de processos, a falta de informações sobre reutilização, importantes no contexto do DBC, e o elevado custo associado a sua aquisição.

2.3.5 - JIRA

JIRA (ATLASSIAN, 2006a) é um sistema de controle de modificações comercial utilizado por centenas de empresas (ATLASSIAN, 2006b). Ele foi implementado utilizando-se tecnologia J2EE, e todas as suas interfaces gráficas foram desenvolvidas para Internet. Ele é um sistema comercial, mas gratuito para projetos de código aberto.

O sistema permite a organização das informações em projetos que podem ser subdivididos em componentes. A cada projeto e componente podem ser atribuídos os usuários responsáveis. As RMs são geradas para um projeto e, opcionalmente, para um componente específico.

O processo de controle de modificações é definido através de diagramas de transição de estados. O sistema possui um processo sugerido, que pode ser configurado, e permite também a inclusão de novos processos. Um processo pode ser associado a um projeto ou a uma combinação de projeto e tipo de RM.

O sistema fornece um editor de processos de controle de modificações para auxiliar a modelagem, onde são inseridas informações de estados e transições, no formato textual. A interface para modelagem não possui recursos gráficos, o que facilitaria o trabalho de modelagem de processos, provendo uma visão geral de todo o processo sendo modelado e possibilitando a sua visualização em um formato mais intuitivo.

O sistema possui dois tipos de formulários, um para criação e outro para edição de uma RM, sendo que este último é exibido durante toda a execução de um processo de controle de modificações. Estes formulários podem ser adaptados, por exemplo, com a inclusão de novos campos. Cada campo pode ser configurado para ser exibido apenas em determinados projetos e tipos de RMs. Porém, a configuração não chega ao nível das atividades do processo, o que seria interessante para que um usuário só tivesse acesso às informações relevantes à atividade na qual ele está envolvido. Além disso, também é possível criar novos tipos de campos através de uma API fornecida, o que exige programação na linguagem Java.

Uma característica interessante do sistema é a possibilidade de criar sub-requisições a partir de uma RM. Cada sub-RM possui os mesmos campos de uma RM

comum, e é tratada como tal. Uma RM e suas filhas não possuem uma relação de precedência, ou seja, a RM mãe pode ser finalizada sem que suas filhas tenham sido finalizadas. No contexto de DBC, este mecanismo seria interessante para o caso onde uma equipe consumidora precisa encaminhar uma RM recebida para a equipe produtora, podendo assim manter o rastro entre as RMs.

É possível fazer a integração do JIRA com sistemas de controle de versão para o agrupamento lógico de artefatos modificados com uma RM, da mesma forma em que é feita no Mantis. Porém, além do CVS, ele também pode ser integrado ao Subversion (COLLINS-SUSSMAN *et al.*, 2004) e ao Perforce (WINGERD, 2005). Os mesmos problemas discutidos em relação ao Mantis repetem-se aqui.

Outras características conhecidas do sistema são a notificação por e-mails, a exibição de relatórios, a importação e exportação de dados através de arquivos XML e a importação de dados das ferramentas Bugzilla e Mantis.

O sistema JIRA, apesar de ser um sistema com diversas características interessantes, não é suficiente para ser utilizado no controle de modificações em DBC, já que não são mantidas as informações sobre os consumidores dos projetos e componentes, tampouco as informações sobre os contratos mantidos entre eles, fundamentais para que o problema da cadeia de responsabilidades de manutenção possa ser resolvido.

2.4 - Abordagens e sistemas de controle de modificações para DBC

DBC é uma disciplina de desenvolvimento de sistemas através da reutilização de componentes bem definidos e produzidos independentemente (BROWN, 2000; LARSSON, 2000).

Existem diferentes métodos para apoiar o DBC. Alguns exemplos são: Catalysis (D'SOUZA, WILLS, 1998), UML Components (CHEESMAN, DANIELS, 2000), KobrA (ATKINSON *et al.*, 2001) e Select Perspective (APPERLY *et al.*, 2003).

Alguns deles, como o Catalysis e o UML Components, não se preocupam com a aplicação da GCS em seus processos, ao contrário do KobrA e do Select Perspective, que discutem inclusive questões relacionadas ao controle de modificações aplicadas ao DBC.

A seguir, apresentamos as abordagens encontradas na literatura que contemplam o controle de modificações aplicado ao DBC.

2.4.1 - Terra/MwR

A abordagem MwR (*Maintenance with Reuse*) (KWON *et al.*, 1999) apóia a manutenção de sistemas legados através de funcionalidades de GCS e reutilização de software. Ela não foi modelada para o desenvolvimento de novos sistemas, mas para a manutenção de sistemas já existentes, que foram desenvolvidos reutilizando-se componentes.

A MwR integra quatro atividades principais: gerência de configuração, processo de reutilização, processo de manutenção e administração da biblioteca de reutilização. A gerência de configuração possui as subatividades de gerenciamento de modificações e versões, e acompanhamento (*status accounting*). O processo de reutilização e o processo de manutenção são os envolvidos na evolução de sistemas legados, que podem reutilizar componentes ou não. O processo de administração da biblioteca de reutilização consiste em povoar a biblioteca, decidir sobre a aquisição de novos componentes, controlar as modificações realizadas sobre os componentes e notificar os interessados.

Uma nova RM é recebida simultaneamente pelo processo de reutilização e de manutenção, que interagem entre si para o tratamento da RM. No processo de reutilização, é realizada a busca de componentes que possam atender à RM, e, possivelmente, a adaptação destes para que possam ser reutilizados. No processo de manutenção, é feita a análise, a implementação e a verificação da modificação. Estes processos interagem para a troca de informações sobre a reutilização de componentes, e podem interagir com a administração da biblioteca, pedindo a alteração ou aquisição de novos componentes, ou mesmo fornecendo novos componentes produzidos. O processo de gerência de configuração apóia todos os demais processos.

Porém, no processo de manutenção, nota-se uma inversão de atividades e uma necessidade de maior integração com o processo de reutilização. Para avaliar uma RM, o CCC[4] recebe um relatório técnico para tomada de decisão sobre a RM, podendo ser

---

[4] O CCC (Comitê de Controle da Configuração), ou, em inglês, CCB (*Configuration Control Board*), representa um conjunto de pessoas responsáveis pela avaliação e aprovação de uma requisição de modificação, e pela garantia da implementação das modificações aprovadas (IEEE, 1990).

aprovada, rejeitada ou postergada. Porém, não consta neste relatório informações importantes, como estimativas de custo e esforço, pois a atividade de análise de impacto somente é realizada depois que a RM foi aprovada. Além disso, como a reutilização de um componente terá impacto no custo e esforço da modificação, é fundamental que estas informações sejam contempladas na elaboração do relatório técnico que será utilizado pelo CCC, o que não acontece no processo proposto.

Para dar suporte à abordagem MwR, foi desenvolvido o protótipo TERRA. Ele possui as seguintes funcionalidades: busca e registro de componentes, registro de requisições de modificação, registro de histórico de reutilização e registro de aprovação de modificações. Existem formulários próprios para cada uma destas funcionalidades, onde informações são coletadas e armazenadas no sistema. Ao requisitar uma modificação, é possível incluir informações como o identificador e a versão do componente relacionado.

Uma característica muito interessante do sistema é a possibilidade de registrar informações sobre a reutilização de cada versão de componente, mantendo assim o registro da experiência de reutilização dos componentes. Algumas informações que podem ser armazenadas são: o tipo de reutilização, que pode ser caixa-branca ou caixa-preta, ou seja, com alteração ou não do componente, o resumo das modificações sobre o componente, o domínio no qual ele foi reutilizado, os produtos e componentes relacionados, etc. Estas informações podem ser úteis para os usuários que pretendem reutilizar um componente.

O processo da abordagem MwR está implementado no código fonte do sistema TERRA, o que dificulta sua manutenção no caso de necessidade de evolução do processo. Por exemplo, no formulário de registro dos componentes, não há campos para a identificação da versão do componente, e alterações nos formulários demandam alteração do código-fonte do protótipo.

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, e que envie notificações a respeito de manutenções nestes componentes.

O sistema também poderia prover maior assistência ao usuário para o preenchimento automático de certos campos dos formulários, o que reduziria o número de erros e agilizaria seu trabalho. Por exemplo, a data de registro de uma RM e o seu solicitante, caso já registrado e autenticado, poderiam ser preenchidos pelo próprio sistema.

2.4.2 - KobrA

O método KobrA (ATKINSON *et al.*, 2001) é uma abordagem para engenharia de linha de produtos que envolve todo o ciclo de vida do software. Uma das áreas abrangidas pelo método é a gerência de configuração, sendo definidos um metamodelo e um processo para o controle de modificações. Os processos definidos no KobrA seguem uma notação própria, mas foi cogitado o uso da especificação SPEM (OMG, 2005), por ser padronizada pela OMG, que foi descartada por ainda ser uma proposta e não ter sido votada na época da elaboração do método.

No metamodelo, cada modificação está associada a uma RM, que é a fonte das informações sobre as razões da modificação. Cada requisição, por sua vez, está associada a uma justificativa, que define os argumentos e uma avaliação de custo e benefício da implementação da modificação. Cada modificação também está associada a uma descrição, que fornece detalhes sobre a natureza da modificação. A análise desta descrição pode ajudar na estimativa de custo de modificações futuras e pode ser utilizada para comparação do esforço estimado com o real para a modificação, para fins de planejamento.

O modelo estabelece ainda o conceito de conjunto de modificações (*change set*), que encapsula um conjunto de modificações realizadas no desenvolvimento de um software. Ele provê informações essenciais para a transferência destas modificações entre os contextos de desenvolvimento de componentes e de desenvolvimento com 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 são úteis para a análise e localização de modificações.

O processo de controle de modificações do KobrA preocupa-se tanto com as modificações no contexto de desenvolvimento de componentes quanto no contexto de desenvolvimento com componentes. Além disso, preocupa-se com o impacto das modificações entre estes dois contextos. Entretanto, ele não contempla o envio de notificações sobre modificações às pessoas envolvidas no processo. Além disso, a questão da cadeia de responsabilidades de manutenção dos componentes não é considerada e não existe implementação de apoio a esta abordagem.

2.4.3 - Select Perspective

Select Perspective (APPERLY *et al.*, 2003) é uma abordagem de apoio ao desenvolvimento de sistemas no paradigma DBC baseada no modelo de *fornecimento*, *gerenciamento* e *consumo* de componentes, definida pela Select Business Solutions (2006).

Para cada perspectiva deste modelo, é definido um processo próprio. O processo de fornecimento de componentes preocupa-se principalmente com a construção, entrega e manutenção de componentes. O processo de gerenciamento está relacionado à aquisição, certificação, publicação e busca de componentes. Já o processo de consumo de componentes preocupa-se principalmente com o desenvolvimento e manutenção de sistemas que reutilizam componentes, sendo que estes sistemas podem ser, na verdade, outros componentes.

O processo de controle de modificações, apesar de não ser definido formalmente, está presente nas três perspectivas do modelo definido. Uma questão importante considerada pela abordagem diz respeito à responsabilidade da manutenção. Segundo a abordagem, as modificações corretivas de componentes do tipo caixa-preta e do tipo caixa-branca, em geral, são de responsabilidade do fornecedor. Porém, os componentes do tipo caixa-branca podem ser modificados pelos consumidores, e, neste caso, os fornecedores podem recusar dar manutenção nestes componentes já modificados. Para auxiliar a identificação do responsável pelas modificações, a abordagem propõe a definição de Acordos de Nível de Serviço (*Service Level Agreements - SLA*) para os componentes, onde sejam especificados os direitos e deveres de cada parte.

Algumas melhorias podem ser identificadas nesta abordagem. É importante que os SLAs sejam definidos não só para os componentes do tipo caixa-branca, mas também para os caixa-preta, para considerar, por exemplo, o período de tempo válido para a manutenção corretiva. Além disso, o SLA também deve ser usado para definir questões relacionadas à evolução e adaptação do software, definindo a responsabilidade das partes sobre modificações evolutivas e adaptativas.

A Select Business Solutions desenvolveu uma solução contendo alguns sistemas que dão apoio a esta abordagem, auxiliando a construção, publicação e busca de componentes, e a definição de processos de desenvolvimento de software, entre outras funcionalidades. Dentre estes sistemas fornecidos, está uma biblioteca de componentes.

Entretanto, a solução não contém um sistema específico de controle de modificações, sendo que algumas funcionalidades são fornecidas através da biblioteca de componentes, como a disseminação de informações. A biblioteca mantém os componentes disponíveis para consumo, sendo que cada componente pode estar associado a um SLA. Os consumidores podem se inscrever como interessados em cada componente. Assim, quando uma nova versão do componente é publicada na biblioteca, é enviada uma mensagem eletrônica a cada usuário interessado.

Porém, a abordagem considera que todos os consumidores utilizarão os componentes sob as mesmas regras, já que o SLA é único por componente. Seria interessante que o SLA pudesse ser especializado por consumidor, garantindo direitos e deveres personalizados, como a vigência da garantia de manutenção corretiva, os direitos sobre evolução e adaptação, entre outros, o que certamente influenciaria na determinação dos preços dos componentes comercializados. Além disso, os fornecedores não têm acesso às informações de consumo dos seus componentes publicados, como a identificação dos consumidores, seus dados de contato, etc. Estas informações são importantes no processo de controle de modificações do fornecedor, sendo utilizadas, por exemplo, na definição da importância e prioridade de modificações.

2.5 - Comparação dos Sistemas

Um conjunto de contribuições e limitações foi identificado na análise dos sistemas e abordagens apresentados neste capítulo. Estas características são descritas a seguir, e na Tabela 1, os sistemas são caracterizados segundo a presença ou ausência delas.

1. **Modelagem gráfica do processo de controle de modificações**

   Indica se a abordagem possui recursos para a modelagem gráfica de processos de controle de modificações. Os sistemas podem não oferecer recurso algum, sendo necessária a alteração do código fonte do sistema para a modelagem do processo, ou podem oferecer recursos visuais, possibilitando a modelagem através do próprio sistema. Estes recursos visuais podem também ser categorizados em textuais, quando a modelagem é realizada através de uma tabela, por exemplo, ou gráficos, quando a modelagem é realizada através de desenhos e funcionalidades como arrastar, soltar e ligar elementos. Assim, o ícone ▬ indica que o sistema não oferece nenhum recurso de modelagem, o

ícone ± indica a existência de recursos textuais e o ícone + indica a existência de recursos gráficos para modelagem de processos.

2. Configuração da coleta de informações

A maior parte dos sistemas define formulários para a coleta de informações, sendo que em alguns é possível customizá-los, inserindo, alterando e removendo campos para a coleta das informações. Alguns sistemas oferecem recursos para facilitar a customização, e em outros é necessária a alteração do código fonte. Em alguns sistemas é possível customizar apenas o estilo do formulário, como alterar as cores e as imagens exibidas pelo formulário. Assim, os sistemas que oferecem recursos para a configuração dos formulários em termos de coleta de informações estão marcados com o ícone +, e os outros estão marcados com o ícone –.

3. Linguagem de processos padronizada

Este item indica se a abordagem utiliza uma linguagem de modelagem de processos padronizada para a modelagem de processos de controle de modificações. Ela é marcada com o ícone + em caso positivo e com o ícone – em caso negativo.

A maior parte dos sistemas de controle de modificações utiliza a notação de diagramas de transição de estados para representar o processo de controle de modificações. Na verdade, estes diagramas são adequados para a representação de estados de um determinado produto, sendo que, neste caso, o produto é a RM. Porém, para a representação de um processo, com suas atividades, fluxo de dados e controle, produtos consumidos e produzidos, papéis responsáveis pelas atividades e produtos, a utilização de uma linguagem de modelagem de processos é mais adequada.

Além disso, idealmente esta linguagem deve ser padronizada, para que os processos modelados sejam compreendidos por um número maior de pessoas e sistemas.

4. Processo de controle de modificações configurado por projeto

Este item indica se o sistema ou abordagem permite a modelagem e a configuração de um processo de controle de modificações para cada projeto de software. Algumas abordagens não permitem a configuração dos processos, e outras sim, mas este processo configurado deve ser utilizado por todos os projetos. Esta característica é importante, pois as empresas e, mais

especificamente, os projetos de cada empresa, podem possuir características e necessidades individuais, e por isso requerem processos de controle de modificações adaptados às suas necessidades. A presença desta característica é marcada com o ícone ✚ na Tabela 1, e a ausência é marcada com o ícone ▬.

5. Envio de notificações

Este item indica se a abordagem ou sistema se preocupa com o envio de notificações aos usuários com informações sobre eventos ocorridos no processo de controle de modificações (✚), de acordo como a função de acompanhamento, definida nas principais normas internacionais de GCS (ISO, 1995a; EIA, 1998; IEEE, 2005), ou não (▬). Geralmente, o envio de notificações é configurável com relação ao momento do envio, os destinatários e o conteúdo da mensagem, e ocorre através do envio de mensagens eletrônicas.

6. Pacotes de modificações

Indica se a abordagem ou sistema mantém rastros entre a modificação realizada, no nível lógico, e os artefatos modificados, através da integração com um sistema de controle de versões (✚), ou não (▬). Isso permite contextualizar as modificações realizadas nos artefatos em um nível mais alto de abstração, facilitando a verificação e auditoria dos artefatos. Além disso, permite a comparação entre as modificações lógicas, em termos dos artefatos modificados, e a aplicação das alterações realizadas de um ambiente para outro. Isto seria interessante, por exemplo, para auxiliar a aplicação das modificações realizadas em uma nova versão de componente em componentes consumidos, mais antigos.

7. Software implementado

Indica se a abordagem possui um sistema de controle de modificações implementado (✚), o que demonstra a viabilidade de implementação das idéias propostas. Este item é usado na caracterização das abordagens de controle de modificações para DBC, já que para desenvolvimento convencional de software foram descritos os próprios sistemas de software.

8. Responsabilidade de manutenção

Indica se o sistema ou abordagem oferece mecanismos para auxiliar a identificação dos responsáveis pela manutenção (✚), especialmente para software desenvolvido no paradigma DBC, onde existe o problema da cadeia de responsabilidades de manutenção.

9. Identificação dos consumidores

Indica se a abordagem ou sistema mantém e possui mecanismos que auxiliem a coleta de informações sobre os consumidores (✚), principalmente dos componentes, auxiliando, por exemplo, a identificação do impacto de uma modificação e o contato do produtor com o consumidor.

A Tabela 1 apresenta uma comparação entre os sistemas e abordagens apresentados neste capítulo, segundo os critérios descritos acima. Estes critérios foram selecionados a partir da análise das contribuições e limitações dos sistemas e abordagens descritos neste capítulo, tendo como base as definições relacionadas ao controle de modificações encontradas nas normas internacionais que tratam da GCS (IEEE, 1987; ISO, 1995a; EIA, 1998; IEEE, 2005).

O objetivo desta comparação não é identificar qual abordagem é melhor ou pior, pois elas podem não atender a alguns dos critérios por estes não serem importantes para atingir seus objetivos específicos. Algumas abordagens são muito mais abrangentes, como o GConf, o MwR e Terra, o KobrA e o Select Perspective, sendo que elas foram analisadas somente sob a perspectiva de sistemas de controle de modificações. Porém, esta comparação ajuda a identificar alguns pontos que podem ser desenvolvidos nos sistemas e abordagens atuais.

Foram analisados também outros sistemas de controle de modificações, mas verificou-se que eles eram muito similares a algum dos descritos acima, com relação aos critérios utilizados. Foram analisados os sistemas de código aberto *Roundup* (2006), *phpBugTracker* (2006) e *Scarab* (TIGRIS, 2006), e foi concluído que eles são muito similares ao Bugzilla.

Outros sistemas comerciais analisados são muito similares ao ClearQuest, como o *StarTeam* (BORLAND, 2005), o *TeamTrack* (SERENA, 2004) e o *Razor* (VISIBLE, 2003). O StarTeam e o TeamTrack, porém, permitem que os processos de controle de modificações sejam modelados graficamente, através de ferramentas de desenho. O Razor também permite a modelagem de processos e da coleta de informações, mas toda a configuração é realizada através de arquivos textuais.

Percebe-se também, através da Tabela 1, que o ClearQuest e JIRA são muito similares. Na verdade, as principais diferenças entre eles são: O módulo administrativo do JIRA, que permite a customização do processo de controle de modificações e da coleta de informações, possui interface para Internet, enquanto no ClearQuest este

módulo é um aplicativo próprio para ser executado em ambiente MS Windows. Além disso, a forma de manter os rastros entre uma RM e as novas versões dos arquivos alterados é diferente. No ClearQuest, o usuário seleciona a atividade na qual ele está trabalhando antes de fazer uma operação de *check-in*. Isso é possível devido à integração do ClearQuest com o ClearCase, o sistema de controle de versões da IBM Rational. No JIRA, o desenvolvedor deve ter acesso ao identificador da RM, e inseri-lo em conjunto com uma palavra especial no comentário do *check-in* realizado, o que torna a tarefa do usuário mais lenta, já que ele deverá procurar o número da RM relacionada.

Tabela 1 – Características dos sistemas e abordagens descritos neste capítulo

Critérios	Bugzilla, GConf, phpBugTracker, Scarab, Roundup	Mantis	ClearQuest, JIRA, Razor	StarTeam, TeamTrack	MwR-TERRA	KobrA	Select Perspective
Modelagem gráfica do processo de controle de modificações	–	–	±	+	–	–	–
Configuração da coleta de informações	–	+	+	+	–	–	–
Linguagem de processos padronizada	–	–	–	–	–	–	–
Processo de controle de modificações configurado por projeto	–	–	+	+	–	–	–
Envio de notificações	+	+	+	+	+	–	+
Pacotes de modificações	+	+	+	+	–	+	–
Software implementado	+	+	+	+	+	–	–
Responsabilidade de manutenção	–	–	–	–	–	–	+
Identificação dos consumidores	–	–	–	–	–	–	–

Além destes sistemas descritos, existem ainda vários outros, como os listados em (CMCROSSROADS, 2006).

Os critérios apresentados neste capítulo não formam um conjunto completo para análise de sistemas e abordagens de controle de modificações. Entretanto, eles ajudam a detectar pontos de melhoria nos sistemas atuais, podendo ser utilizados como guia na elaboração de uma proposta mais abrangente, trazendo contribuições para a área, como é o caso da proposta descrita no Capítulo 3.

2.6 - Linguagens de Processo

Na Tabela 1, nota-se que o critério referente ao uso de uma linguagem de processos padronizada não é atendido por nenhuma das abordagens ou sistemas. Na maioria deles, o processo é modelado através de um diagrama de transição de estados aplicado às RMs. Um diagrama de transição de estados estabelece os possíveis estados de um objeto, as ações permitidas em cada estado, e qual evento dispara a mudança de um estado para outro (ESTUBLIER *et al.*, 2005). Assim, é possível modelar a seqüência de estados das requisições de modificação inseridas nos sistemas. Algumas abordagens permitem inclusive a modelagem gráfica destes diagramas (SERENA, 2004; BORLAND, 2005).

Porém, estes diagramas não fornecem toda a semântica necessária para a definição de um processo. Um processo, segundo PLFEEGER (2004), é um conjunto de tarefas ordenadas que envolvem atividades, restrições e recursos para se obter uma saída desejada. Algumas características de processos são: contém atividades que devem ser realizadas, consumindo e gerando produtos, podem conter subprocessos, definidos em uma hierarquia de processos, utilizam recursos, possuem procedimentos e restrições e possuem responsáveis pelas atividades e pelos produtos destas.

Através dos diagramas de transição de estados, não é possível modelar alguns dos conceitos envolvidos em um processo, como os produtos consumidos e gerados pelas atividades e os responsáveis pelas atividades e produtos, entre outros. Assim, os sistemas devem prover outros recursos para que estas informações possam ser inseridas.

Uma alternativa aos diagramas de transição de estados é a modelagem dos processos de controle de modificações através de linguagens de processos. Neste caso, o foco da modelagem é transferido para as atividades que devem ser realizadas para o controle de modificações, e a RM passa a ser um produto do processo. Dessa forma, a visão do processo torna-se mais global, e é possível modelar características como os produtos gerados e consumidos pelas atividades e as responsabilidades envolvidas no processo, dependendo da linguagem de modelagem de processos utilizada.

Além disso, é interessante que seja utilizada uma linguagem de processos padronizada, pois isso permite que vários sistemas a utilizem, facilita o intercâmbio de processos entre os sistemas, e facilita também a compreensão dos processos. Entretanto, não existe uma única linguagem de processos padronizada que seja universalmente utilizada. Existem várias iniciativas, como a *IDEF3* da IDEF (MAYER *et al.*, 1995), a

Process Definition Interchange da WfMC (WFMC, 1999), a *PSL* da NIST e ISO (SCHLENOFF *et al.*, 2000), a *BPEL4WS* da Microsoft, IBM e outros (ANDREWS *et al.*, 2003) e a *SPEM* da OMG (OMG, 2005).

Todas estas abordagens definem formalismos para a modelagem de processos, sendo que a SPEM é focada em processos de software. Por ser baseada na UML, uma linguagem de modelagem de sistemas muito utilizada, espera-se que os profissionais de software tenham facilidade em utilizar a notação SPEM, tanto para modelar quanto para compreender os processos modelados. Por este motivo, será dada uma maior ênfase à SPEM, linguagem adotada por este trabalho.

2.6.1 - A especificação SPEM

A especificação SPEM (*Software Process Engineering Metamodel*) foi definida pela OMG (2005), em conjunto com várias empresas, como IBM Rational e Computer Associates, entre outras. A OMG, uma organização internacional fundada em 1989 e mantida por centenas de membros, produz diversas especificações na área da Informática, com foco na interoperabilidade de aplicações. Entre as especificações da OMG, estão a UML e a MDA, por exemplo. Todas as especificações estão disponíveis na página da OMG na Internet (OMG, 2006).

A SPEM estabelece um metamodelo para a definição de processos de software. É interessante ressaltar que ela não se preocupa com a execução dos processos, apenas com a especificação destes. Este metamodelo é baseado no MOF (OMG, 2002b), um metamodelo para definição de metamodelos, ou seja, um metametamodelo. Na hierarquia de metamodelos do MOF, apresentada na Fig. 2-2, o próprio MOF está no nível M3, o metamodelo SPEM, que é uma instância do MOF, se situa no nível M2, os processos modelados segundo o SPEM se situam no nível M1, e os processos em execução no nível M0.

Além do metamodelo, a especificação SPEM define um *UML Profile* para SPEM. Através de um *UML Profile*, é possível fazer uma adaptação da UML para determinado domínio. Isso é feito utilizando-se o mecanismo de extensão da UML, através do uso de estereótipos (*stereotypes*), definições de marcas (*tag definitions*) e restrições (*constraints*). Assim, é possível utilizar ferramentas CASE UML que permitam a utilização de *UML Profiles* (OBJECTEERING, 2006; SPARXSYSTEMS, 2006) para modelar processos através da SPEM.

A OMG está interessada na definição de uma linguagem de transformação entre modelos baseados no MOF (OMG, 2002a), o que permitiria a transformação de modelos baseados em *UML Profile* para SPEM em modelos baseados no metamodelo da SPEM. Isso permitiria a interoperabilidade entre as ferramentas baseadas em *UML Profile* e metamodelo da SPEM.

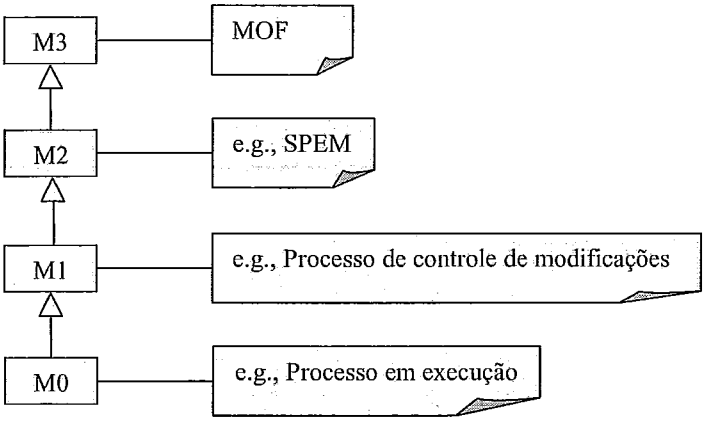


Fig. 2-2 - Níveis de modelagem do MOF, adaptado da especificação SPEM (OMG, 2005)

O metamodelo SPEM possui dois pacotes principais, o *SPEM_Foundation*, que é uma extensão de um subconjunto do metamodelo da UML 1.4 (OMG, 2001), e o *SPEM_Extensions*, que contém novos elementos necessários à modelagem de processos. O principal subpacote deste pacote de extensão é o *Process Structure*, e suas principais classes são exibidas na Fig. 2-3.

A classe *Macroatividade* (*WorkDefinition*) representa uma macroatividade do processo, que pode ser composta por outras macroatividades ou atividades. Suas entradas e suas saídas são representadas explicitamente. A classe *Atividade* (*Activity*) representa uma atividade do processo, e, apesar de possuir herança de *WorkDefinition*, normalmente não é decomposta em outras atividades, apesar disso não ser proibido pela especificação.

Toda *Macroatividade* possui um *Executor* (*ProcessPerformer*), que é o responsável geral por todos os seus subelementos. Já uma *Atividade* específica possui como responsável um *Papel* (*ProcessRole*), e pode ter outros *Papéis* como assistentes para sua realização. Um *Papel* também pode assumir a responsabilidade sobre determinados produtos. Um *Produto* (*WorkProduct*) representa qualquer artefato que seja produzido, consumido ou modificado por uma atividade do processo.

A especificação SPEM também possui elementos para a organização dos processos, como o *Processo* (*Process*), que representa a descrição de um processo

completo, contendo atividades, produtos, etc. Ele é uma especialização da classe *Pacote* (*Package*), definida como na UML 1.4.

Outro tipo de elemento foi introduzido na especificação para auxiliar na condução e entendimento do trabalho, chamado *Guia* (*Guidance*). Ele fornece mais informações sobre um determinado elemento modelado, podendo representar técnicas, métricas e exemplos, entre outros.

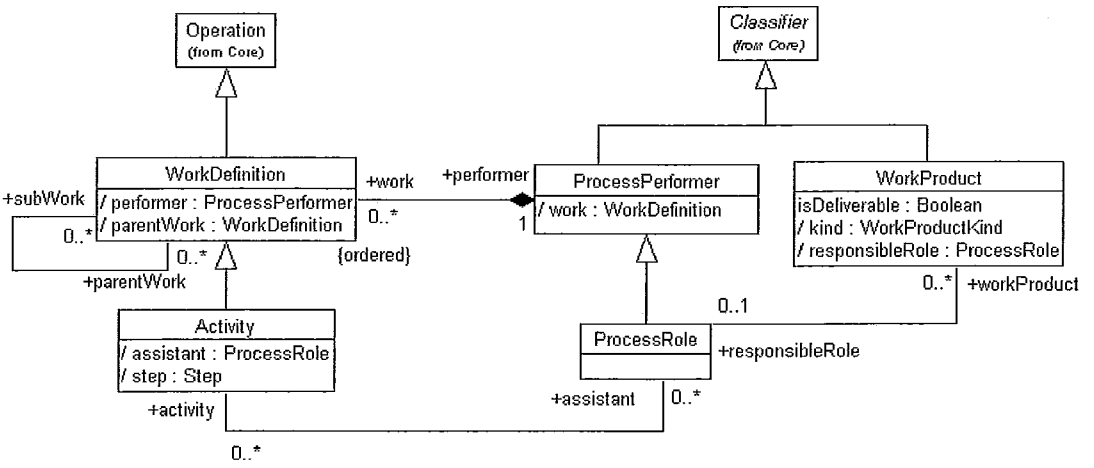


Fig. 2-3 - Principais classes do pacote *Process Structure* da especificação SPEM (OMG, 2005)

Além dos elementos específicos do pacote *SPEM_Extensions*, a SPEM possibilita a utilização de diagramas da UML adaptados para a modelagem de processos. O Diagrama de Atividades é utilizado para a modelagem da seqüência das atividades e dos produtos consumidos e produzidos no processo. Através dele, é possível definir o início e o fim do processo, as transições entre as atividades, a entrada e saída de produtos nas atividades, as decisões e os sincronismos existentes.

O Diagrama de Casos de Uso é utilizado para a modelagem da responsabilidade sobre as atividades. Os Executores e os Papéis responsáveis pelas atividades são modelados como atores, e as atividades e macroatividades como casos de uso. As associações podem indicar que os atores são assistentes ou executores das atividades.

O Diagrama de Classes é utilizado para a modelagem dos produtos das atividades e seu relacionamento com os atores do processo. Através dele, pode-se modelar o relacionamento entre os produtos, como as associações, composições e dependência entre eles, e quem são os seus responsáveis.

O Diagrama de Seqüência é utilizado para ilustrar os padrões de interação entre as instâncias de elementos do modelo SPEM, o Diagrama de Estados, para a

modelagem do comportamento de elementos do modelo, e o Diagrama de Pacotes, para representação da composição de processos.

A especificação SPEM também sugere ícones a serem utilizados para representar seus elementos nos referidos diagramas. Os principais ícones são exibidos na Tabela 2.

Tabela 2 – Ícones dos elementos principais da SPEM

Macroatividade *WorkDefinition*	Atividade *Activity*	Executor *ProcessPerformer*	Papel *ProcessRole*	Produto *WorkProduct*	Processo *Process*	Guia *Guidance*
						

Algumas ferramentas comerciais estão preparadas para a modelagem de processos através da especificação SPEM, como a *Enterprise Architect* (SPARXSYSTEMS, 2006), a *Objecteering* (OBJECTEERING, 2006) e a *IRIS Suite* (OSELLUS, 2006), sendo que esta última permite também a execução dos processos modelados.

A SPEM encontra-se atualmente na versão 1.1, sendo que a versão 2.0 está sendo elaborada pela OMG e seus parceiros. Esta nova versão deverá incorporar alterações da versão 2.0 da UML e melhorias da especificação atual da SPEM.

2.7 - Considerações finais

Neste capítulo, foram analisados diversos trabalhos relacionados ao controle de modificações, comparados segundo os critérios descritos na Seção 2.5. Esta comparação é apresentada pela Tabela 1.

Analisando-a, percebe-se que nenhum dos trabalhos satisfaz todos os critérios. Os trabalhos que mais se aproximam disso são os sistemas de controle de modificações comerciais, como o ClearQuest, JIRA, StarTeam e TeamTrack. Eles apenas não satisfazem o critério de modelagem gráfica de processos de controle de modificações, o critério de utilização de uma linguagem de processos padronizada e alguns critérios relacionados ao DBC, mais especificamente o auxílio à resolução do problema da cadeia de responsabilidades de manutenção e a coleta e armazenamento de informações sobre os consumidores dos componentes.

As abordagens relacionadas ao DBC mostram-se bastante inflexíveis, tanto com relação à definição do processo de controle de modificações, quanto com relação à

definição das informações que devem ser coletadas ao longo do processo. Estas questões são interessantes, pois permitem que os processos sejam moldados especificamente para os diferentes projetos, contribuindo para a eficiência do trabalho de controle de modificações.

Das abordagens relacionadas ao DBC, somente a Select Perspective leva em consideração a questão da responsabilidade sobre a manutenção dos componentes. Porém, ela não possui um sistema de controle de modificações implementado, mas possui um sistema de gerenciamento de componentes que mantém algumas informações sobre a reutilização destes, como o produtor e o acordo de nível de serviço para cada componente.

Nenhuma das abordagens e sistemas mantém informações relacionadas aos consumidores de componentes. Deixa-se de utilizar informações importantes na atividade de análise de impacto e avaliação de uma RM. Informações sobre o consumo dos componentes podem alterar a definição da prioridade de atendimento das RMs, e são fundamentais para a notificação dos consumidores sobre as modificações realizadas nos componentes por eles reutilizados.

Outra constatação obtida pela análise realizada é que os sistemas e abordagens atuais não utilizam uma linguagem de processos padronizada. Na verdade, a maior parte é baseada em diagramas de transição de estados, que não permitem expressar toda a semântica envolvida em um processo. Porém, como foi apresentado na Seção 2.6, existem algumas linguagens de processos padronizadas, como por exemplo a SPEM, que podem ser utilizadas para a definição dos processos de controle de modificações.

Pode-se concluir que nenhuma das abordagens ou sistemas analisados satisfazem todos os critérios considerados anteriormente, o que motiva a definição de uma nova abordagem para o controle de modificações de software com ênfase no contexto do DBC, que será apresentada no Capítulo 3.

Capítulo 3 - Abordagem Odyssey-CCS

3.1 - Introdução

Os processos de controle de modificações em software desenvolvidos através do DBC são diferentes dos convencionais (KWON *et al.*, 1999; ATKINSON *et al.*, 2000; DANTAS *et al.*, 2003). Algumas atividades existentes são adaptadas e novas atividades e decisões são inseridas. Além disso, as novas atividades requerem e produzem novas informações, que devem ser armazenadas e devidamente disseminadas aos atores envolvidos na execução do processo, como sugere a função Acompanhamento da Configuração (*Status Accounting*), descrita nas principais normas internacionais de GCS (ISO, 1995a; EIA, 1998; IEEE, 1998).

No Capítulo 1, foi discutido o problema chamado cadeia de responsabilidades de manutenção, que se refere à determinação dos responsáveis pela manutenção de cada componente reutilizado, dos consumidores destes componentes e dos direitos e deveres de cada parte. Não foram encontrados na literatura sistemas de controle de modificações que auxiliem a resolução deste problema.

Considerando os problemas referenciados anteriormente e as deficiências e virtudes das abordagens e sistemas analisados no Capítulo 2, foi definida uma abordagem para auxiliar o controle de modificações de software no contexto do DBC, chamada de Odyssey-CCS (*Change Control System*).

Esta abordagem é composta por atividades, organizadas em duas fases, chamadas Fase de Preparação e Fase de Execução, e pela definição de módulos de software responsáveis por auxiliar a execução destas atividades. A Fig. 3-1 exibe uma visão geral da abordagem. As fases são representadas por retângulos, as atividades por elipses, os módulos por cubos e os sistemas externos que se relacionam com a abordagem por engrenagens.

A primeira fase, Fase de Preparação, é de responsabilidade do Gerente de Configuração, e seu objetivo é definir e organizar um ambiente que possa ser utilizado para o controle de modificações de sistemas de software desenvolvidos através do DBC. Esta é a fase inicial, que vigora antes de um sistema ser colocado sob controle de modificações, sendo que as definições aqui realizadas deverão constar no Plano de Gerência de Configuração de Software. É esperado que já tenha sido definido pela

organização um processo padrão de controle de modificações, que poderá ser adaptado para cada projeto. As seguintes atividades estão presentes nesta fase: Adaptar Processo, Modelar Formulário e Configurar Execução, que serão discutidas no decorrer deste capítulo.

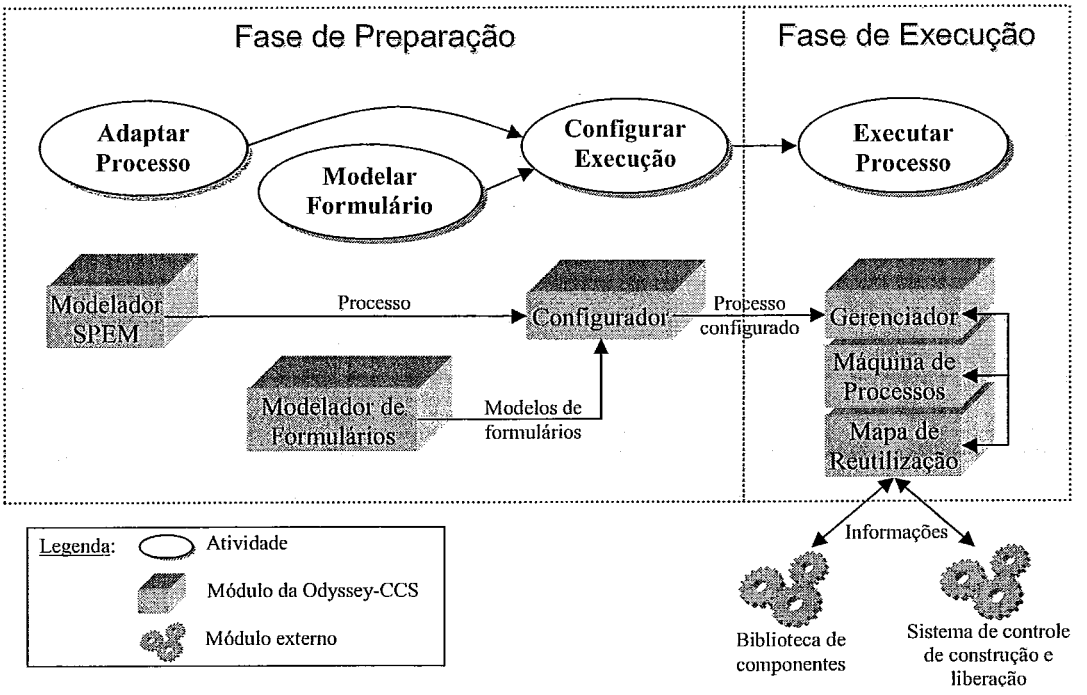


Fig. 3-1 - Abordagem Odyssey-CCS

Ao final desta fase, estando o ambiente configurado, entra em vigor a Fase de Execução, cujo objetivo principal é auxiliar a execução dos processos de controle de modificações. Esta fase vigora durante todo o tempo em que um sistema está sob controle de modificações e nela atuam todas as pessoas envolvidas com o controle das modificações, como os Clientes, o Gerente de Configuração, Analistas de Sistemas, Implementadores, Testadores, Gerente da Qualidade, etc.

A seguir, são apresentadas as atividades definidas na abordagem, em conjunto com os módulos de software definidos para auxiliarem a sua realização. Na Seção 3.2, são apresentadas questões relacionadas à adaptação de processos de controle de modificações. Na Seção 3.3, é apresentada a atividade de modelagem de formulários para coleta de informações. Na Seção 3.4, é introduzido o Mapa de Reutilização, que contém as informações sobre reutilização de componentes. Na Seção 3.5, são discutidas questões relacionadas à atividade de configuração da execução do controle de modificações. Na Seção 3.6, é apresentada a atividade de execução de processos de controle de modificações, e, na Seção 3.7, são apresentadas as considerações finais.

3.2 - Adaptação de processos

A atividade **Adaptar Processo** faz parte da Fase de Preparação da abordagem e é executada pelo Gerente de Configuração. A principal motivação para definição desta atividade é o fato de haver diferenças entre os processos de controle de modificações convencionais e para DBC (KWON *et al.*, 1999; ATKINSON *et al.*, 2000; DANTAS *et al.*, 2003). Isso sugere a necessidade de adaptação dos processos de controle de modificações.

É válido ressaltar que esta necessidade não é exclusiva para processos no contexto do DBC. No desenvolvimento convencional de software, os processos de GCS devem poder ser adaptados às necessidades específicas das empresas e dos projetos (ISO, 1995a; EIA, 1998; IEEE, 1998), e isso inclui também os processos de controle de modificações.

Porém, esta necessidade é ainda mais evidente quando se trata de DBC, já que as principais normas de GCS (ISO, 1995a; EIA, 1998; IEEE, 1998) não possuem diretrizes para a definição de processos de GCS adequados ao DBC, e os poucos trabalhos que tratam do assunto encontrados na literatura (KWON *et al.*, 1999; ATKINSON *et al.*, 2000) definem processos de controle de modificações distintos dos convencionais.

Normalmente, uma organização planeja e define um processo padrão de controle de modificações, que retrata de maneira geral as atividades que devem ser executadas para o controle de modificações na organização. Este processo padrão deve ser adaptado para cada projeto específico, de forma a considerar as características individuais de cada projeto.

A principal função desta atividade é adaptar o processo padrão de controle de modificações definido pela organização para ser utilizado em determinado projeto, modelando-o graficamente através de uma notação padronizada de modelagem de processos. O uso de uma notação padronizada permite a compreensão dos processos modelados por um número maior de pessoas, considerando que sua especificação está disponível e é de livre acesso, facilita a reutilização dos processos definidos, e permite a utilização de qualquer ferramenta de modelagem de processos que seja compatível com a notação utilizada.

Há várias especificações para a modelagem de processos (MAYER *et al.*, 1995; WFMC, 1999; SCHLENOFF *et al.*, 2000; ANDREWS *et al.*, 2003; OMG, 2005). Na

abordagem Odyssey-CCS, é utilizada a SPEM (OMG, 2005), uma especificação para modelagem de processos de software, detalhada no Capítulo 2.

Para auxiliar a modelagem e adaptação dos processos, foi definido um módulo de software que permite ao Gerente de Configuração modelar e visualizar os processos graficamente, segundo o metamodelo definido pela especificação SPEM. Não foram encontradas ferramentas livres com estas funções, somente ferramentas comerciais, como a *Objecteering* (OBJECTEERING, 2006) e a *Enterprise Architect* (SPARXSYSTEMS, 2006), que utilizam o *UML Profile* para SPEM, e a *IRIS Suite* (OSELLUS, 2006).

Através deste módulo, é possível modelar o processo padrão de controle de modificações definido pela organização e também criar um novo processo adaptado deste processo padrão para ser utilizado em um projeto específico.

Os processos modelados através deste módulo de software também devem poder ser externalizados segundo a especificação SPEM, para que possam ser utilizados por outros sistemas e pessoas. Mais detalhes são fornecidos na Seção 3.2.2.

A seguir, na Seção 3.2.1, é apresentado um exemplo de processo de controle de modificações de software para o contexto do DBC, que será utilizado ao longo deste trabalho para auxiliar a explicação da abordagem proposta.

3.2.1 - O Processo Exemplo

Para auxiliar a explicação da abordagem proposta neste trabalho, será utilizado um exemplo de processo de controle de modificações para DBC adaptado daquele definido por DANTAS *et al.* (2003). Este *Processo Exemplo* foi modelado através da especificação SPEM.

Neste processo, são definidas as atividades que devem ser realizadas desde a submissão de uma Requisição de Modificação (RM) até a sua finalização, no contexto de sistemas de software desenvolvidos através do DBC. São identificados também os responsáveis pelas atividades e os produtos consumidos e produzidos por elas.

O processo é iniciado pela submissão de uma RM de software, que pode ser feita por usuários de sistemas, por desenvolvedores, por analistas de qualidade ou por revisores do software. Todas estas pessoas podem exercer o papel Cliente da modificação, que é o papel responsável por esta atividade, chamada de **Requisitar Modificação**. Ao final da execução desta atividade, é gerado um Relatório de Requisição de Modificação, que fornecerá detalhes sobre a modificação requerida.

Uma vez solicitada, a RM deve ser encaminhada para **classificação**. Por exemplo, uma RM enviada para correção de um problema que indisponibiliza o sistema deve ser considerada mais importante e prioritária do que uma RM para uma melhoria cosmética do software. No contexto do DBC, quando a RM for dirigida a um componente, a equipe produtora pode utilizar como um dos critérios de classificação o número de consumidores deste componente, o que ajuda a medir o impacto da modificação. Os critérios de classificação utilizados devem estar especificados no Plano de Gerência de Configuração de Software (LEON, 2000). Esta atividade é de responsabilidade do Analista de Sistemas. Os responsáveis pela classificação devem ser **notificados** a cada nova RM.

Em seguida, a RM deve ser **analisada** quanto ao impacto da modificação, considerando o esforço necessário à realização da modificação e o impacto nos custos e cronograma. Para os produtores de componentes, faz parte da análise a identificação e sugestão de confecção de novos componentes, e quem seriam os interessados em sua reutilização. Para os consumidores de componentes, é necessário identificar quais componentes serão afetados pela modificação e quem são os responsáveis por sua manutenção.

No caso do responsável ser a equipe produtora, a RM lhe é enviada e ela deve fornecer o tempo previsto para a modificação. No caso do responsável ser a própria equipe consumidora, a análise pode sugerir entre a implementação da modificação internamente, caso o componente seja do tipo caixa branca, ou através do uso de adaptadores (GAMMA *et al.*, 1995), caso ele seja caixa preta, e a substituição do componente por um outro. Cada uma destas opções passa por um subprocesso diferente. O papel responsável por esta atividade é o de Analista de Sistemas, auxiliado pelo papel Desenvolvedor. Ao final da atividade, é gerado um Relatório Técnico com as informações da análise.

Este relatório, em conjunto com o Relatório de Requisição de Modificação, é utilizado pelo CCC para **avaliação** da RM. Nesta atividade, são tomadas as decisões sobre os rumos da RM, se ela será aceita, rejeitada ou postergada. Para os consumidores de componentes, aceitar a RM implica em tomar outras decisões, conforme a responsabilidade sobre a modificação. No caso da responsabilidade ser dos próprios consumidores, deve-se decidir entre a implementação interna e a substituição do componente. O Relatório Técnico possui informações importantes para esta tomada de decisão, com recomendações justificadas a respeito de quais ações tomar. No caso da

responsabilidade ser do produtor do componente, a RM deve ser repassada a ele e é iniciado o subprocesso **Requisitar Produtor**.

Neste caso, ao final deste subprocesso, a equipe consumidora recebe o componente modificado pela equipe produtora, verifica-o através de testes de regressão e integração (GAO *et al.*, 2003), e o integra ao sistema.

Caso o CCC opte pela **implementação** da modificação na própria organização, a modificação é implementada, verificada e liberada para o Cliente. Caso opte pela substituição do componente, é iniciado o subprocesso **Adquirir Novo**, responsável por obter o componente, analisar sua licença, verificá-lo, e integrá-lo ao sistema.

O CCC pode também considerar que o Relatório de Análise não contém todas as informações necessárias para a tomada de decisões, e decidir por pedir uma re-análise.

A parte principal deste processo modelada através de um diagrama de atividades da SPEM é exibida na Fig. 3-2. Os ícones que indicam o início, final, sincronismo e decisão são os mesmos definidos no diagrama de atividades da UML. As setas tracejadas indicam fluxo de controle e de dados, e as setas contínuas indicam fluxo de controle. Os outros elementos são descritos na Fig. 3-3.

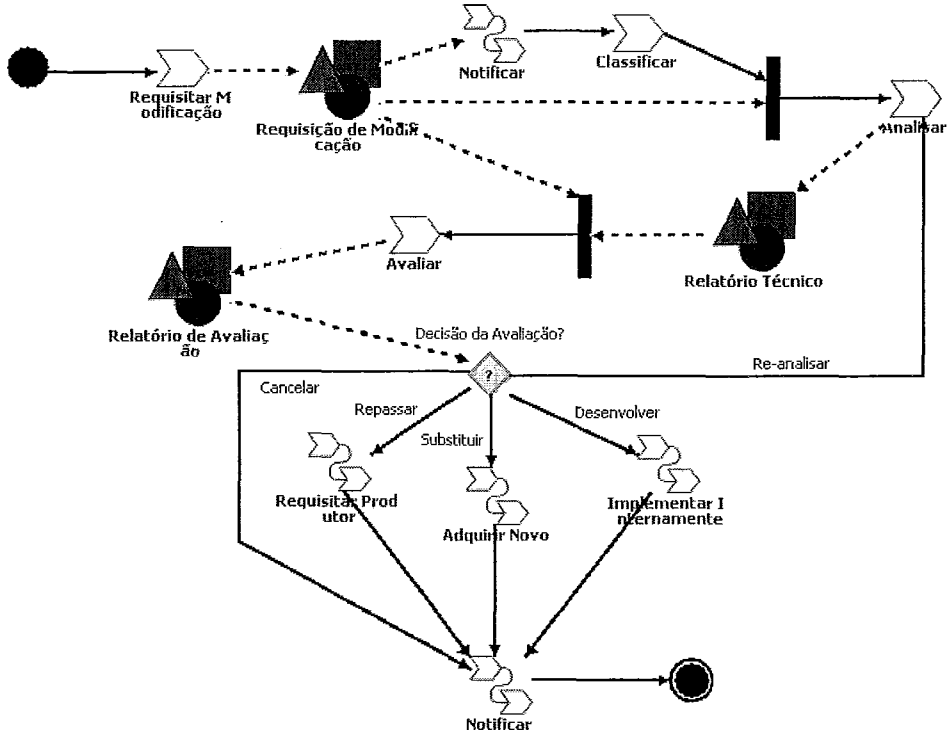


Fig. 3-2 - Diagrama de Atividades do exemplo de processo de controle de modificações para DBC, modelado através da SPEM

Macroatividades como Analisar, Notificar, Adquirir Novo, Implementar Internamente e Requisitar Produtor são detalhadas em outros diagramas de atividades, e não são exibidos na Fig. 3-2.

Os responsáveis pelo processo são modelados através de diagramas de caso de uso, como explicado anteriormente. Na Fig. 3-3, é apresentado o diagrama de caso de uso para o processo exemplo definido nesta seção.

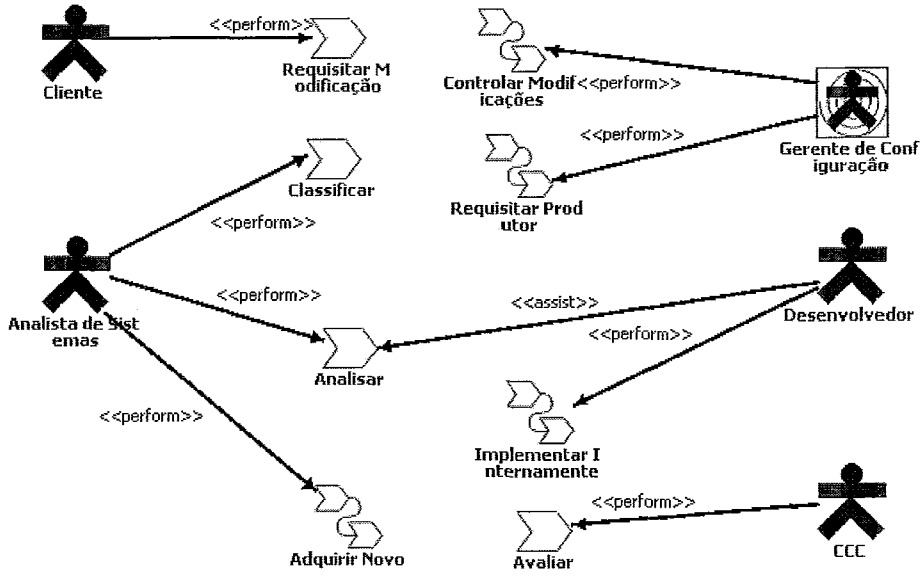


Fig. 3-3 - Responsáveis pelas atividades do processo exemplo

3.2.2 - Modelador SPEM

A abordagem Odyssey-CCS define um módulo de software para auxiliar a modelagem de processos, chamado de Modelador SPEM. Ele possui os seguintes requisitos:

- Compatibilidade com a SPEM: Deve disponibilizar um conjunto mínimo de elementos SPEM que permite a modelagem de processos de controle de modificações em software. São eles: Macroatividade (*WorkDefinition*), Atividade (*Activity*), Produto (*WorkProduct*), Executor (*ProcessPerformer*), Papel (*ProcessRole*), Processo (*Process*), Guia (*Guidance*), Diagrama de Atividades, Diagrama de Casos de Uso e Diagrama de Classes, sendo que cada diagrama possui também seus subelementos. O Diagrama de Atividades possui os elementos Início e Fim do processo, Sincronismo, Fluxo de Controle e Fluxo de Controle e Dados. O Diagrama de Casos de Uso possui as ligações do tipo Executa e Assiste. O Diagrama de Classes possui ligações de Associação, Dependência, Herança e Composição.

- **Modelagem gráfica**: Deve fornecer recursos para a modelagem de processos graficamente, e não textualmente, dispensando a alteração de arquivos de configuração e atividades de implementação. Os processos devem poder ser modelados através de diagramas, como os exibidos nas figuras Fig. 3-2 e Fig. 3-3, utilizando-se os ícones sugeridos pela especificação SPEM.

- **Persistência dos dados**: Deve fornecer recursos para o armazenamento e a recuperação das informações dos processos modelados, permitindo que a modelagem seja realizada em mais de uma iteração e que o processo seja compartilhado com outros usuários da abordagem.

- **Interoperabilidade**: Deve fornecer recursos para a exportação dos processos modelados através do metamodelo SPEM, possibilitando que outros sistemas de software compatíveis com a especificação SPEM reutilizem os processos modelados.

3.3 - Modelagem de formulários

Outra atividade da Fase de Preparação da abordagem é a **Modelar Formulário**, também de responsabilidade do Gerente de Configuração. A principal motivação para a definição desta atividade é a necessidade de coletar informações durante a execução das atividades do processo de controle de modificações. Além disso, como os processos devem ser adaptados por empresa e por projeto (ISO, 1995a; EIA, 1998; IEEE, 1998), a definição das informações a serem coletadas durante a sua execução também deve ser configurável.

Durante a execução do processo de controle de modificações, diversas informações devem ser coletadas, pois serão necessárias à execução de uma atividade futura no processo e poderão ser utilizadas para posterior análise da modificação realizada (ISO, 1995a; EIA, 1998; IEEE, 1998).

No processo modelado na Fig. 3-2, por exemplo, na atividade Requisitar Modificação é produzido um documento, Requisição de Modificação, que contém informações importantes para a execução das atividades Classificar e Analisar. Exemplo destas informações são o Cliente que está fazendo a requisição, o sistema alvo de modificação, a descrição da RM e a prioridade com que o Cliente gostaria que a RM fosse tratada.

Na atividade Analisar, é produzido o documento Relatório Técnico, que possui informações importantes para a tomada de decisões pelo CCC sobre a condução da modificação. Exemplos de informações presentes neste relatório são a referência para a

RM sendo analisada, referência para os autores do relatório, as alternativas de implementação, os itens afetados, o impacto em custo e prazo e a recomendação de ação fornecida pela equipe técnica (LEON, 2000).

No contexto de DBC, este relatório precisa de outras informações, no caso em que um componente reutilizado é afetado. Por exemplo, devem estar presentes as alternativas de condução da modificação, que podem ser: requisição de modificação ao produtor do componente, modificação interna ou substituição do componente.

No caso em que a responsabilidade da modificação é do produtor, este deve informar a sua estimativa de prazo, que será agregada à estimativa da equipe consumidora, que inclui a verificação da modificação e integração do componente modificado ao sistema.

Quando a responsabilidade é do próprio consumidor, o relatório deverá ter informações sobre: orçamento em termos de custo e prazo para a modificação pelo produtor, os componentes disponíveis para substituição do componente alvo da RM, incluindo custo, análise de compatibilidade com o componente atual, dados do produtor, dos consumidores deste componente e do contrato de aquisição, e alternativas para a implementação da modificação pelo próprio consumidor.

Na perspectiva do produtor do componente, outra informação importante são os consumidores do componente sendo modificado, que é um tipo de indicador do impacto da modificação.

Outras atividades, como Requisitar Produtor e Adquirir Novo Componente, ausentes no processo convencional de controle de modificações, precisam ter suas informações coletadas e armazenadas. Para isso, novos formulários deverão ser modelados.

Um formulário poderá ser associado a um determinado Produto de uma Atividade, permitindo a coleta de informações durante a execução dos processos. Detalhes são encontrados na Seção 3.5.

A seguir, na Seção 3.3.1, é apresentado o modelo conceitual dos formulários, e na Seção 3.3.2, é discutido o módulo de software proposto para auxiliar a execução desta atividade.

3.3.1 - Modelo conceitual

Na Fig. 3-4 são apresentados os conceitos relacionados à coleta de informações na abordagem Odyssey-CCS. O Formulário, disposto no centro do diagrama, é a

entidade que representa a interface de coleta de informações. Ele é composto por campos, utilizados pelos usuários para a inserção de unidades de informação. O conjunto de informações coletadas em um mesmo Formulário constitui um Documento. Dessa forma, uma Informação é dependente de um Campo, e um Documento é dependente de um Formulário. Os documentos, e conseqüentemente suas informações, são agrupados por Modificação, que é a entidade que representa uma execução completa do processo de controle de modificações, discutida na Seção 3.6.

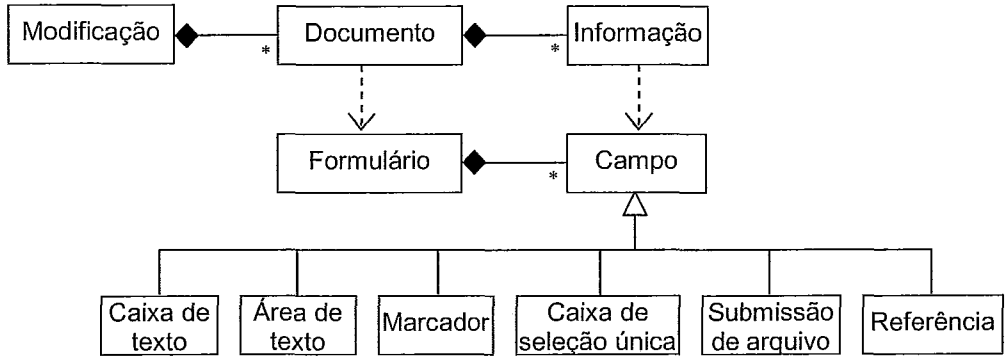


Fig. 3-4 - Modelo Conceitual dos Formulários e Informações

São considerados na abordagem os seguintes tipos de campos:

- **Caixa de texto**: É o tipo de campo utilizado para coletar informações textuais pequenas, que não ultrapassam uma linha. São utilizados para a coleta de informações como nome, telefone, endereço eletrônico, etc.
- **Área de texto**: É o tipo de campo utilizado para coletar informações textuais maiores, que podem ter mais de uma linha. Exemplos de informações coletadas por estes tipos de campos são observações, descrições, mensagens, etc.
- **Marcador**: É o tipo de campo utilizado para a marcação de opções. Possui dois estados, o marcado, com semântica de verdadeiro, e o desmarcado, significando falso. Pode ser usado também para enviar respostas a perguntas binárias. Por exemplo, para responder se determinado relatório foi aprovado.
- **Caixa de seleção única**: É o tipo de campo utilizado para a seleção de uma única opção dentre uma lista. Pode ser utilizado, por exemplo, para a seleção do tipo da RM, dentre Corretiva, Adaptativa, Evolutiva e Preventiva (PRESSMAN, 2005).
- **Submissão de arquivo**: Tipo de campo utilizado para a submissão de um arquivo (*upload*). Pode ser utilizado, por exemplo, para envio de um arquivo de *log* na RM.

- **Referência**: Este tipo de campo é utilizado para a navegação entre formulários. A Referência possui o endereço de um formulário e permite a navegação do formulário que a contém para o formulário referenciado.

3.3.2 - Modelador de Formulários

A abordagem Odyssey-CCS define um módulo de software para auxiliar a modelagem de formulários para a coleta de informações, chamado de Modelador de Formulários. Ele possui os seguintes requisitos:

- **Modelagem gráfica**: Devem ser fornecidos recursos para a modelagem de formulários graficamente, dispensando a alteração de arquivos de configuração e atividades de programação. Os tipos de campos descritos na Seção 3.3.1 podem ser configurados e inseridos no formulário.

- **Instanciação de formulários**: Os formulários modelados contêm apenas informações estruturais, como os campos que o compõem e suas posições. Para ser utilizado, um formulário deve ser instanciado, ou seja, deve ser gerado a partir de seu modelo. Assim, um módulo externo que deseje utilizar determinado formulário pode solicitar uma instância dele ao Modelador de Formulários.

- **Persistência dos dados**: Os formulários modelados devem ser armazenados, e os documentos e informações coletados por cada formulário também devem ser armazenados, agregados a uma Modificação, a pedido do cliente do formulário.

- **Consulta às informações**: Deve ser fornecida uma API (*Application Programming Interface*) para consulta às informações persistidas pelo Modelador de Formulários. Esta API pode ser utilizada por outros módulos para realizar os seguintes tipos de consultas:

1. Quais são os formulários cadastrados?
2. Quais são os campos de um determinado formulário?
3. Quais são os documentos gerados por determinado formulário no contexto de uma determinada modificação?
4. Qual informação foi preenchida em um determinado campo, no contexto de um determinado documento?

Como exemplo de módulo que pode utilizar esta API, podemos citar o *Gerenciador de Execução*, definido na Seção 3.6.

3.4 - Informações sobre reutilização de componentes

Como já discutido no Capítulo 1, no contexto de manutenção de software desenvolvido através do DBC, existe o problema chamado cadeia de responsabilidades de manutenção. Este problema refere-se à dificuldade de se encontrar quem é o responsável pela manutenção de determinado componente e quem são os consumidores dos componentes.

No exemplo da Seção 3.2.1, na atividade Analisar, os Analistas de Sistemas deverão identificar se algum componente reutilizado precisa de modificação. Em caso afirmativo, deverá ser identificada de quem é a responsabilidade da modificação, se do produtor ou do consumidor do componente. Para determinar o responsável, devem ser analisados os termos do contrato de aquisição do componente.

A responsabilidade sobre a manutenção pode variar segundo diversos critérios. Por exemplo, o contrato pode vincular a garantia de manutenção ao tipo da modificação, isto é, se corretiva, evolutiva ou adaptativa. Pode ser que seja garantida somente a manutenção corretiva do software sem ônus, e ainda assim por um determinado período. É importante que o componente adquirido inclua sua especificação de requisitos, para que não haja dúvidas quanto ao tipo da modificação solicitada. Também pode ser que o componente seja um software de código aberto (OSI, 2006) e sua licença não garanta a manutenção sobre o componente.

Voltando ao exemplo da Seção 3.2.1, nas atividades Analisar e Avaliar, quando o responsável pela modificação é o produtor do componente, é importante saber quem são os seus consumidores. Os Analistas podem entrar em contato com os consumidores para identificar seu interesse pela modificação e o CCC pode utilizar estes dados na avaliação, por exemplo, para identificar clientes estratégicos entre os consumidores do componente e assim definir prioridades. Na liberação do componente modificado, é fundamental saber quem são os consumidores e quais os seus direitos sobre a nova versão, para que sejam notificados e ações de atualização do componente sejam tomadas.

Na Seção 3.4.1, apresentamos uma proposta que abrange as informações sobre reutilização de componentes identificadas como necessárias durante o processo de controle de modificações.

3.4.1 - Mapa de Reutilização

Para auxiliar a resolução do problema da cadeia de responsabilidades de manutenção, a abordagem Odyssey-CCS mantém um conjunto de informações sobre a reutilização dos componentes, em formato eletrônico, que podem ser utilizadas para a identificação das responsabilidades de manutenção e dos consumidores de componentes. Este conjunto de informações constitui o Mapa de Reutilização, apresentado na Fig. 3-5.

Neste modelo, a classe *Componente* representa os componentes de software reutilizáveis cadastrados, e inclui informações sobre suas versões, dependências, produtores e consumidores. Os produtores e consumidores são representados pela classe *Equipe*. Cada componente também deve estar associado a pelo menos um tipo de licença, representado pela classe *Licença*, que possui um conjunto genérico de cláusulas que regem a utilização e manutenção do componente. Um *Componente* pode estar associado a vários tipos de *Licenças*, o que permite a especificação de direitos e deveres para diferentes contextos. Por exemplo, um componente pode possuir diferentes licenças para uso acadêmico e comercial, pode ser gratuito para empresas sem fins lucrativos e projetos de código aberto (*Open Source*) e pago para uso comercial, pode ter licenças que incluam ou não manutenção, com preços diferenciados, entre outras possibilidades.

Quando o *Componente* é adquirido pela equipe consumidora, é firmado um *Contrato* entre o produtor e o consumidor para o determinado componente. Este *Contrato* segue uma determinada *Licença*, selecionada dentre aquelas associadas ao *Componente*, e inclui informações específicas à aquisição corrente, como a data de início da vigência do *Contrato*.

Este modelo permite, portanto, saber quem é o produtor de um componente, quem são seus consumidores, e quais os contratos firmados entre um produtor e um consumidor na aquisição de cada componente.

O Mapa de Reutilização pode ser utilizado tanto por equipes produtoras quanto por equipes consumidoras de componentes. No caso das equipes produtoras, elas têm disponíveis as informações sobre o consumo de cada componente produzido. No caso de equipes consumidoras, elas têm acesso às informações sobre a aquisição de cada componente consumido. No caso de equipes híbridas, elas têm acesso aos dois tipos de informações.

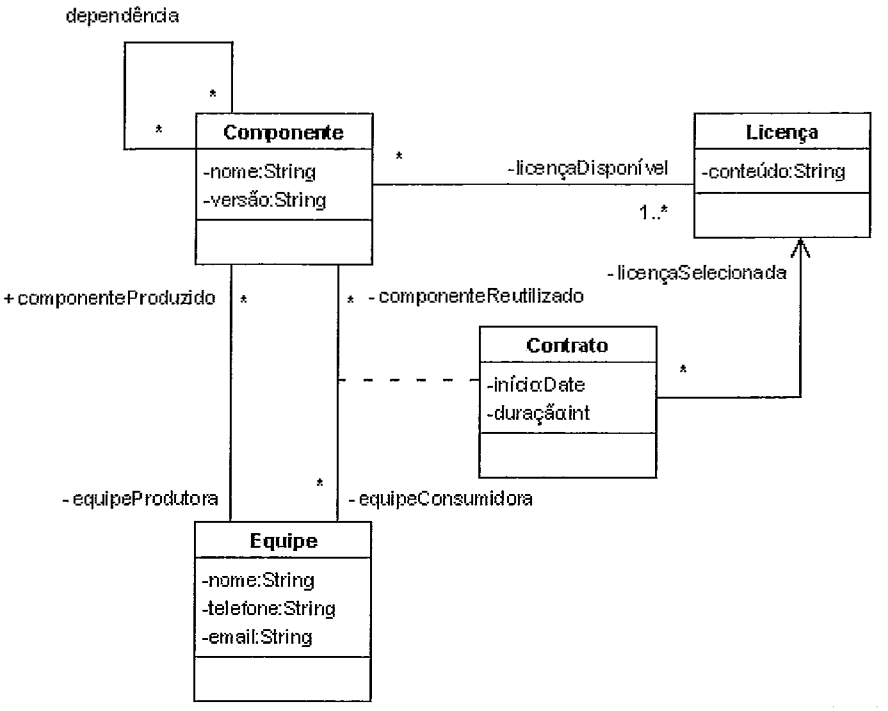


Fig. 3-5 - Modelo Conceitual do Mapa de Reutilização

Estas equipes podem representar empresas distintas ou mesmo equipes internas de uma determinada organização, já que em ambos os casos existe o problema da cadeia de responsabilidades de manutenção. Em quaisquer destes casos, é importante ter acesso às informações mantidas pelo Mapa de Reutilização, para identificação dos consumidores e produtores e dos responsáveis pela manutenção.

A seguir, na Seção 3.4.2, são discutidas questões relacionadas ao preenchimento do Mapa de Reutilização.

3.4.2 - Preenchimento do Mapa de Reutilização

Para ser útil, o Mapa de Reutilização deve estar preenchido com as informações sobre reutilização de componentes pertinentes às organizações, que são: Quais são os componentes produzidos e reutilizados, quem são seus produtores e consumidores, quais foram os contratos estabelecidos, quais as licenças disponíveis para cada componente, se os componentes são compostos por outros componentes, e quais são eles.

Como não há uma padronização com relação à manutenção destas informações, é possível que cada organização as mantenha de uma forma diferente, em diferentes formatos. Ou elas estarão anotadas em uma planilha, ou em um documento eletrônico de texto, ou em papel, ou até mesmo não estarão cadastradas em nenhum local.

Essas informações devem ser identificadas, reunidas e transferidas para o Mapa de Reutilização. Elas podem ser cadastradas manualmente, através de uma interface gráfica para cadastramento. Porém, esta atividade pode ser bastante trabalhosa, dependendo do número de componentes utilizados. Pode-se também construir uma solução de software que consiga coletar todas as informações e transferi-las para o Mapa de Reutilização.

Além de preenchido, o Mapa de Reutilização deverá ser atualizado no surgimento de novas informações. Mais uma vez, pode ser empregado um método manual de atualização, que é muito suscetível a erros, tanto por esquecimento de cadastrar as novas informações quanto por erro no cadastramento destas.

Uma das maneiras de automatizar a obtenção e atualização destas informações é integrar o Mapa de Reutilização a uma Biblioteca de Componentes, que é um local preparado para o armazenamento, seleção e recuperação de componentes de software (SAMETINGER, 1997).

Algumas informações estão presentes tanto no Mapa de Reutilização quanto na Biblioteca de Componentes, como os componentes e suas versões cadastradas e os seus produtores. Estas informações são inseridas quando um componente é publicado na Biblioteca, que pode ser adaptada e monitorada para obtenção das demais informações.

A adaptação consiste em acrescentar as licenças disponíveis para cada componente, caso essa informação ainda não exista na Biblioteca de Componentes. Ao solicitar a aquisição de um componente, o consumidor deverá se identificar, selecionar a licença de uso desejada, e informar os demais dados requeridos para a definição do contrato. Assim, são capturadas e inseridas no Mapa de Reutilização as informações do consumidor e do contrato para o componente adquirido.

Se o Mapa de Reutilização integrado à Biblioteca de Componentes for utilizado desde o início do desenvolvimento dos sistemas, as informações de reutilização serão coletadas automaticamente e o Mapa estará completo. Caso contrário, ou seja, a empresa já reutiliza ou produz componentes anteriormente à utilização do Mapa integrado à Biblioteca, estas informações deverão ser incluídas de outra forma, por exemplo, manualmente ou com o auxílio de um outro software específico para isso.

Para obter informações sobre os componentes produzidos sob encomenda, a equipe produtora pode integrar o Mapa de Reutilização ao Sistema de Controle de Construção e Liberação. Quando uma nova liberação for criada, é solicitado ao Gerente de Configuração que informe quem é o consumidor do componente produzido. Isto é

interessante quando uma Biblioteca de Componente não é utilizada ou quando o componente não for nela publicado, por exemplo, por questões contratuais.

O caso mais complicado é aquele no qual a empresa não possui as informações sobre a reutilização dos componentes armazenadas, ou suas informações não são completas. Neste caso, pode-se monitorar o próprio Sistema de Controle de Modificações. A cada RM, é verificado se o requerente está cadastrado no Mapa de Reutilização como consumidor do componente alvo da modificação. Caso contrário, o Gerente de Configuração é alertado para solucionar o problema. Ele deve buscar as informações sobre reutilização do componente, possivelmente tendo que contatar o próprio consumidor, e atualizar as informações no Mapa de Reutilização.

A empresa ComponentSource (COMPONENTSOURCE, 2006) faz algo semelhante ao que é feito pela Biblioteca de Componentes integrada ao Mapa de Reutilização para aquisição das informações dos consumidores dos seus componentes. Quando um consumidor compra um componente, ele se identifica e preenche um cadastro, que é armazenado. Uma diferença é que os componentes possuem apenas um tipo de licença associada a eles. Porém, não se sabe como as informações são armazenadas, e se são integradas ao sistema de controle de modificações da empresa.

3.4.3 - Configuração do acesso ao Mapa de Reutilização

No contexto do DBC, é importante que as pessoas envolvidas no processo de controle de modificações tenham acesso às informações dos componentes reutilizados pelo software que está sendo controlado, sendo que ele próprio pode ser um componente. Estas informações são mantidas pelo Mapa de Reutilização.

O acesso ao Mapa de Reutilização é introduzido através da configuração de um campo do tipo Referência em um formulário, com o caminho para o Mapa de Reutilização, como exemplificado pela Fig. 3-6. Assim, quando este formulário for apresentado a um usuário, ele poderá acessar o Mapa. Isto é feito para facilitar o trabalho dos usuários que participam do controle de modificações, pois estas referências são colocadas nos locais onde as informações de reutilização serão necessárias, como na atividade de Análise de Impacto. O Mapa também pode ser acessado diretamente, mesmo quando o Odyssey-CCS não for utilizado.

Esta Referência configurada dá acesso à consulta de componentes provida pelo Mapa de Reutilização e às informações relacionadas a cada um destes.

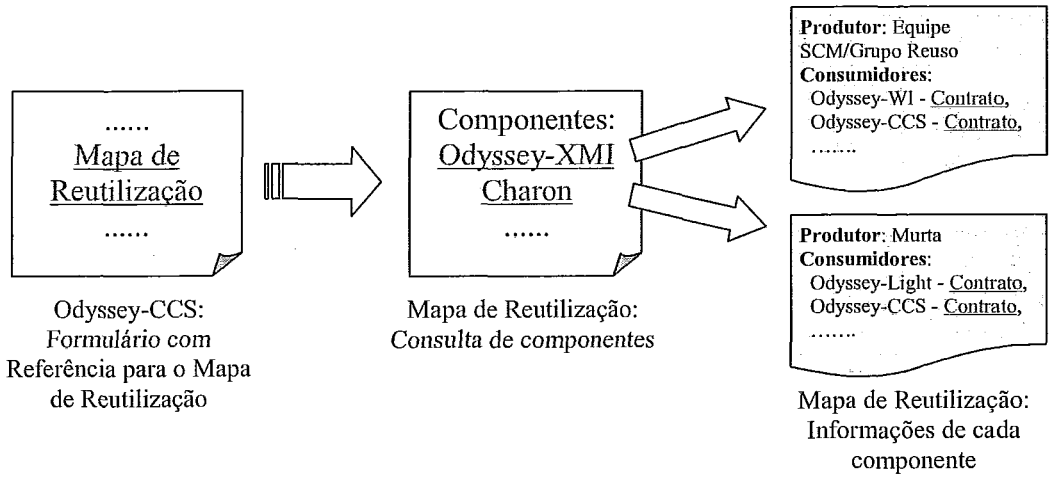


Fig. 3-6 - Exemplo de formulário do Odyssey-CCS configurado para dar acesso ao Mapa de Reutilização

3.5 - Configuração do controle de modificações

A atividade **Configurar Execução** também faz parte da Fase de Preparação da abordagem e é executada pelo Gerente de Configuração para cada projeto. Neste contexto, projeto refere-se a uma configuração para controle de modificações específica para um determinado software. As características que devem ser configuradas são detalhadas ao longo desta seção. A principal motivação para a definição desta atividade é a necessidade de configuração da execução de um processo de controle de modificações para determinado software.

Nesta atividade, o Gerente de Configuração é responsável pela criação de um projeto, associação de um processo previamente adaptado ao projeto criado, atribuição das pessoas responsáveis pelas atividades do processo, configuração da coleta de informações durante a execução de instâncias do processo e configuração do envio de notificações. Estas subatividades são descritas nas seções a seguir.

3.5.1 - Criação de um Projeto

Para cada sistema de software em desenvolvimento e manutenção, deve ser criado um projeto específico para seu controle de modificações. Este projeto possui como atributos um nome e uma descrição, e uma associação para o software cujas modificações serão controladas.

Também deve ser associado a ele um processo de controle de modificações modelado em conformidade com a especificação SPEM, conforme definido na Seção 3.2. Outras informações específicas ao projeto deverão ser associadas ao processo, para

que ele possa ser executado. A execução do processo é detalhada na Seção 3.6, e o restante da configuração é detalhada a seguir.

3.5.2 - Atribuição de responsabilidades

No processo de controle de modificações selecionado para o projeto, estão definidos os papéis que participam de cada atividade, sejam como executores ou como assistentes. É necessário identificar quais são as pessoas envolvidas com o controle de modificações no projeto criado, e quais papéis cada uma assumirá. É possível que uma mesma pessoa assuma mais de um papel no processo, principalmente quando as equipes de desenvolvimento e manutenção são pequenas.

Este mapeamento possibilita o controle de acesso às atividades e a notificação sobre as atividades pendentes de execução a cada pessoa participante do processo de controle de modificações.

A abordagem também define um usuário especial chamado *iniciador* (*_owner*), que representa o usuário que inicia uma instância de execução de um processo. A principal motivação para a definição deste usuário especial é a necessidade de associar o instanciador de um processo a um determinado papel, sendo que não é possível saber de antemão qual usuário instanciará o processo.

Por exemplo, no final do processo modelado na Seção 3.2.1, existe a macroatividade *Notificar* definida, que notifica o usuário que instanciou o processo de controle de modificações sobre o seu resultado. Assim, pode-se definir o usuário especial *iniciador* como o destinatário desta notificação, em tempo de modelagem, e em tempo de execução, fazer a identificação do usuário real que instanciou o processo como o destinatário da notificação.

Mais detalhes sobre a execução dos processos são fornecidos na Seção 3.6.

3.5.3 - Configuração da coleta de informações

Conforme definido na função de Acompanhamento da GCS (ISO, 1995a; EIA, 1998; IEEE, 1998), as informações referentes às modificações realizadas devem ser coletadas e armazenadas para registro do histórico das modificações e posterior análise.

Para a coleta das informações, a abordagem utiliza os formulários modelados na atividade descrita na Seção 3.3. Eles podem ser reutilizados, caso já tenham sido utilizados em outros projetos, ou podem ter sido modelados especificamente para uso em um determinado projeto.

Além da definição de quais informações devem ser coletadas, deve ser definido em que momento isso deve ocorrer (ISO, 1995a; EIA, 1998; IEEE, 1998). Na abordagem Odyssey-CCS, esta definição é feita através da associação de um formulário a cada produto. Assim, as informações são coletadas ao término de cada atividade, através do preenchimento dos formulários associados aos produtos gerados.

No exemplo da Seção 3.2.1, a atividade *Requisitar Modificação* produz uma *Requisição de Modificação*. A este produto será associado um formulário composto pelos campos necessários para a coleta das informações relevantes neste momento do processo, como: nome do sistema alvo de modificação, descrição e prioridade da modificação. Quando a atividade *Requisitar Modificação* for finalizada, o formulário associado ao produto *Requisição de Modificação* será exibido ao executor da atividade para o preenchimento das informações, que serão armazenadas.

3.5.4 - Configuração do envio de notificações

Conforme orienta a função de Acompanhamento da GCS (ISO, 1995a; EIA, 1998; IEEE, 1998), determinadas pessoas devem receber certas informações durante o processo de controle de modificações. Para isso, devem ser configurados quais são estas informações, quem são os destinatários e quando as informações devem ser enviadas.

Na abordagem Odyssey-CCS, estas configurações são realizadas pelo Gerente de Configuração antes que instâncias do processo possam ser executadas. As informações a serem enviadas são definidas em um formulário próprio para notificação, chamado de *Formulário de Notificação*, que contém campos para a configuração dos destinatários, do assunto da notificação e da mensagem a ser enviada.

Os destinatários são definidos através da seleção de papéis presentes no processo modelado. Todas as pessoas que exercem algum papel selecionado receberão as notificações.

O momento do envio da notificação deve ser modelado explicitamente no próprio processo de controle de modificações. Isto é, deve ser modelada uma atividade que produz a notificação, que deverá ser associada ao formulário de notificação padrão.

No exemplo da Seção 3.2.1, há dois pontos de notificação, determinados pela macroatividade *Notificar*, detalhada na Fig. 3-7. Neste caso, o produto *E-mail* deve ser associado ao formulário de notificação, que é preenchido pelo Gerente de Configuração com os destinatários, o assunto e as informações a serem enviadas pela notificação.

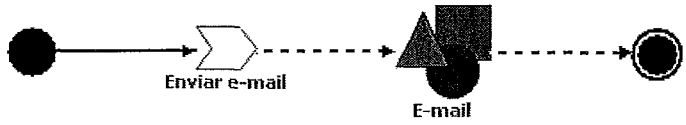


Fig. 3-7 - Detalhe da atividade Notificar, do exemplo apresentado na Fig. 3-2

3.6 - Execução de processos de controle de modificações

A atividade **Executar Processo** faz parte da Fase de Execução da abordagem (Fig. 3-1). Ela atua sobre um processo especificado através da realização das atividades da Fase de Preparação. O principal objetivo desta atividade é auxiliar a execução dos processos de controle de modificações, controlando a dinâmica de cada processo, fornecendo informações para as pessoas neles envolvidas e coletando e armazenando as informações geradas durante cada execução.

A execução do processo de controle de modificações é iniciada com a criação de uma instância do mesmo, que é a entidade que representa concretamente um fluxo de execução do processo. Ela possui uma identidade única e agrupa todas as informações coletadas durante sua execução. Várias instâncias podem estar em execução simultaneamente, cada uma representando o controle de uma determinada modificação, como exemplificado na Fig. 3-8.

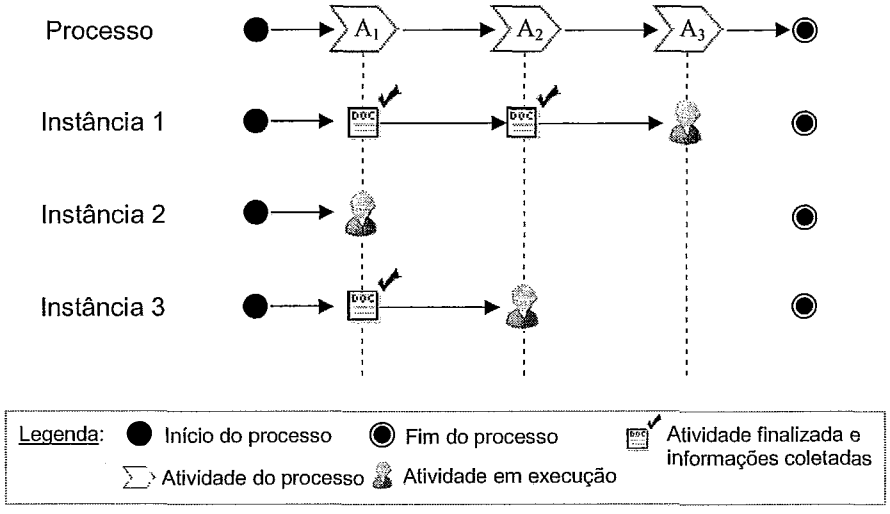


Fig. 3-8 - Instâncias de um processo em execução

Nesta figura, é apresentado um processo como exemplo, constituído por três atividades, A_1, A_2 e A_3, e três instâncias deste processo em execução, cada uma representando o controle de uma modificação em curso. Na primeira instância, foram

executadas as atividades A_1 e A_2, e informações relacionadas a cada uma delas foram coletadas. Já a atividade A_3 está em execução. Simultaneamente, as instâncias 2 e 3 também estão em execução, sendo que na instância 2 a atividade A_1 está em execução, e na instância 3 a atividade A_2 está sendo executada.

Além das informações coletadas, cada instância de processo, que representa uma modificação, está associada aos itens de configuração afetados pela modificação. Estes conceitos são exibidos na Fig. 3-9. Toda *Transação* de *check-in* é realizada no contexto de uma *Modificação*, e gera uma nova *Versão* de cada *Item de Configuração (IC)* que foi modificado. Assim, é possível saber quais *ICs* foram afetados por uma determinada *Modificação*, ou qual foi a *Modificação* que gerou uma determinada *Versão* de *IC*.

Para relacionar uma *Modificação* aos *ICs* modificados, é necessária a integração do controle de modificações com o controle de versões, responsável por manter e controlar a evolução dos *ICs*. A abordagem Odyssey-CCS está integrada ao sistema de controle de versões Odyssey-VCS (OLIVEIRA *et al.*, 2005).

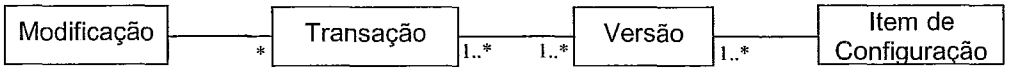


Fig. 3-9 - Modelo conceitual que exibe o relacionamento entre Modificação e Item de Configuração

Conforme detalhado na Seção 3.2, os processos são compostos por atividades, que devem ser realizadas por pessoas que exerçam os papéis responsáveis por elas. Estas atividades devem ser realizadas em uma seqüência predefinida, de acordo com o processo modelado. As pessoas envolvidas no processo de controle de modificações devem estar cientes das atividades pendentes de execução, que são de sua responsabilidade, devem realizá-las e finalizá-las, fornecendo as informações requisitadas pelos formulários de coleta de informações associados aos produtos destas atividades, conforme configuração realizada previamente, descrita na Seção 3.5.

Além de atividades, podem ser atribuídas decisões aos envolvidos no processo de controle de modificações. Da mesma forma, eles devem estar cientes das decisões pendentes e devem tomar as decisões nos momentos definidos pelo processo.

Ao iniciar a realização de uma atividade, é aconselhável que o executor marque a atividade sendo executada, para que outras pessoas que também exerçam um papel associado à atividade tenham ciência que outra pessoa já está envolvida em sua execução. Elas podem contatar o responsável para obter informações sobre o andamento

da execução, e podem participar da realização da atividade, quando necessário, marcando também a atividade com sua participação.

Para realizar uma atividade, os responsáveis devem ter acesso às informações dos produtos consumidos por ela. A abordagem permite que estas pessoas visualizem as informações coletadas pelos formulários associados aos respectivos produtos, exibidas nos próprios formulários. No exemplo da Seção 3.2.1, a realização da atividade Analisar requer as informações do produto Requisição de Modificação, que devem ficar disponíveis para os Analistas de Sistemas e Desenvolvedores. As informações coletadas também devem poder ser consultadas após a execução dos processos, para análise das modificações realizadas.

Quando uma atividade que produz uma notificação, ou seja, cujo produto esteja associado ao formulário padrão de notificação, como descrito na Seção 3.5, ficar pendente, a notificação deve ser automaticamente enviada aos destinatários, conforme configuração previamente definida pelo Gerente de Configuração, sem necessidade de intervenção humana.

Para auxiliar esta atividade, a abordagem faz uso de três módulos principais de software, o Gerenciador, a Máquina de Processos e o Mapa de Reutilização.

O **Gerenciador** é o módulo responsável pela execução dos processos de controle de modificações. Ele mantém os relacionamentos entre as informações do processo adaptado e as informações que foram configuradas na atividade Configurar Execução, descrita na Seção 3.5.

O processo adaptado, especificado com a SPEM, é independente da abordagem e pode ser reutilizado por sistemas de execução de processos compatíveis com esta especificação. Quando o processo é configurado para execução, ele passa a ser específico da abordagem Odyssey-CCS, ou melhor, específico de um determinado projeto de controle de modificações gerenciado através desta abordagem.

O módulo Gerenciador informa aos responsáveis as atividades e decisões pendentes de execução, permite que eles indiquem quais atividades e decisões estão executando, finalizem as atividades e tomem as decisões. Também exibe os formulários para coleta de dados quando uma atividade é finalizada, identifica o momento de enviar notificações e as envia com auxílio de um notificador.

O Gerenciador depende de uma **Máquina de Processos** para controlar a dinâmica de execução das instâncias dos processos. A Máquina de Processos controla o fluxo de execução de um processo adaptado, especificado em SPEM. Ela fornece as

atividades e decisões pendentes por papel, permite a finalização destas, armazenando para cada uma os executores e o tempo de execução, e infere as próximas atividades a serem realizadas. Em resumo, ela permite que seja dado andamento às atividades dos processos.

Algumas características importantes da Máquina de Processos a ser utilizada pela Odyssey-CCS são:

- Compatibilidade com a especificação SPEM, para que ela possa interpretar os processos modelados conforme esta especificação.
- Mapeamento automático da definição gráfica do processo em uma representação executável, sem a necessidade de intervenção humana. Caso este procedimento não seja automatizado, pode haver a introdução de erros e aumento do esforço para a execução dos processos. É importante ressaltar que a especificação SPEM não se preocupa com a execução do processo, apenas com a definição do mesmo. É de responsabilidade da Máquina de Processos a forma de executar os processos definidos através da SPEM.
- Possibilidade de execução de processos contendo ciclos e recursão em sua representação. Um ciclo é caracterizado quando o fluxo de execução volta a uma atividade anterior do processo. No exemplo da Fig. 3-2, há um ciclo modelado, que se inicia na atividade Analisar. Se a resposta da decisão "Decisão da Avaliação?" for "Re-analisar", o fluxo de execução voltará à atividade Analisar. A recursão ocorre quando uma atividade possui referência para uma macroatividade pai. Nos dois casos, as atividades serão re-executadas.
- Possibilidade de extensão de suas funcionalidades sem a necessidade de alteração da implementação da Máquina de Processos em si, mas através da adição de funcionalidades, por exemplo, para a coleta de novas métricas.
- Verificação do processo modelado utilizando-se a especificação SPEM, se ele possui um fluxo que ligue o início ao fim do processo, e se possui ciclos ou recursão infinita. É recomendado que o processo seja verificado antes de ser liberado para execução.

Durante a execução de um processo de controle de modificações, o Gerenciador utiliza o **Mapa de Reutilização** para fornecer as informações sobre reutilização dos componentes envolvidos no projeto.

Isto é feito através da configuração de um campo do tipo Referência em um formulário, como descrito na Seção 3.4.3, para dar acesso à consulta de componentes do

Mapa de Reutilização. Assim, podem ser consultados os componentes envolvidos no projeto e as informações relacionadas a eles, como o produtor, os consumidores, as licenças disponíveis e os contratos firmados.

No exemplo descrito na Seção 3.2.1, foi identificado como importante o acesso às informações sobre reutilização dos componentes envolvidos em um projeto nas atividades Analisar e Avaliar. Assim, os formulários associados aos produtos Requisição de Modificação e Relatório Técnico devem ter um campo Referência configurado para acessar o Mapa de Reutilização.

3.7 - Considerações finais

Os processos de controle de modificações para software desenvolvido no paradigma DBC são diferentes dos processos para software desenvolvido de maneira convencional (KWON *et al.*, 1999; ATKINSON *et al.*, 2000; DANTAS *et al.*, 2003). Novas atividades, decisões e informações devem estar envolvidas no contexto do DBC. Aliado a este fato, há a necessidade de adequação dos processos às organizações e projetos específicos (ISO, 1995a; EIA, 1998; IEEE, 1998).

Existem diversos sistemas de controle de modificações (CMCROSSROADS, 2006), sendo que muitos permitem a configuração dos processos e das informações a serem coletadas sem a necessidade de alteração de código-fonte, como o IBM Rational ClearQuest (WHITE, 2000) e JIRA (ATLASSIAN, 2006a), entre outros. Porém, não foi encontrado um que utilizasse uma notação padronizada de modelagem de processos. Além disso, a grande maioria utiliza unicamente diagramas de estados como notação, que não explicitam os produtos consumidos e produzidos pelas atividades, nem os papéis responsáveis por elas.

Estes sistemas também não oferecem apoio para a solução do problema da cadeia de responsabilidades de manutenção, discutido no Capítulo 2.

Neste capítulo, foi apresentada uma abordagem para controle de modificações em software desenvolvido através do DBC. Ela define algumas atividades e alguns módulos de software para apoiá-las. As principais características da abordagem são a utilização de uma notação padronizada de modelagem de processos, a SPEM (OMG, 2005), a flexibilidade e o apoio para a definição de processos e das informações que devem ser coletadas em cada etapa, e a manutenção de informações a respeito da reutilização de componentes, reunidas no Mapa de Reutilização, para auxiliar a resolução do problema da cadeia de responsabilidades de manutenção.

No Capítulo 4, será apresentado com detalhes o protótipo funcional Odyssey-CCS, homônimo da abordagem, que implementa os módulos de software propostos para apoiar as atividades definidas na abordagem, juntamente com um exemplo de utilização.

Capítulo 4 - Protótipo Implementado

4.1 - Introdução

No Capítulo 3, foi apresentada a abordagem Odyssey-CCS, cujo propósito é auxiliar o controle de modificações em software desenvolvido no contexto do DBC. Esta abordagem define um conjunto de atividades que devem ser realizadas para a implantação e utilização de um ambiente de apoio ao controle de modificações em software.

A falta de apoio computacional prejudica a utilização desta abordagem. Dentre os vários motivos, pode-se citar que a manutenção e análise das informações serão mais difíceis e a disseminação das informações e das atividades pendentes de execução serão mais lentas, principalmente quando estiverem envolvidas várias pessoas, distribuídas geograficamente. Dessa forma, a própria abordagem define módulos de software para viabilizar a sua utilização.

Neste capítulo, é apresentado um protótipo que implementa as idéias introduzidas na abordagem, chamado Odyssey-CCS. A maior parte deste software foi desenvolvida para utilização através da Internet, permitindo que seja acessado por clientes que utilizem diferentes plataformas e que estejam distribuídos geograficamente.

Sua arquitetura é descrita na Seção 4.2 e seus módulos são detalhados individualmente em seções específicas (seções 4.3 a 4.7), conforme o mapeamento exibido na Fig. 4-1. Para auxiliar a explicação destes módulos, é utilizado o *Processo Exemplo* definido no Capítulo 3. Na Seção 4.8, são apresentados dois casos de utilização do protótipo, e, na Seção 4.9, são fornecidas as considerações finais.

4.2 - Visão geral do Odyssey-CCS

Esta seção apresenta uma visão geral da arquitetura do Odyssey-CCS e detalhes da sua implementação. Na Fig. 4-1 são exibidos os principais módulos de software que compõem o protótipo, suas dependências, representadas por setas tracejadas, e os principais produtos gerados e consumidos. Os módulos exibidos como cubos na figura foram inteiramente implementados no âmbito deste trabalho, e os exibidos como engrenagens foram reutilizados. Estes módulos reutilizados foram adaptados para

atenderem a todos os requisitos da abordagem, com exceção do Odyssey (2006), apresentado na Seção 4.3.1, que já estava preparado para ser reutilizado.

O módulo *Modelador SPEM* dá apoio à modelagem de processos através da especificação SPEM. Ele foi implementado como um *plugin* do *Odyssey* e utiliza o módulo *Odyssey-XMI* para a exportação de processos. Estes módulos são detalhados na Seção 4.3.

Outro módulo do Odyssey-CCS é o *Modelador de Formulários*, responsável por dar apoio à modelagem dos formulários para a coleta de informações. Ele é descrito com detalhes na Seção 4.4.

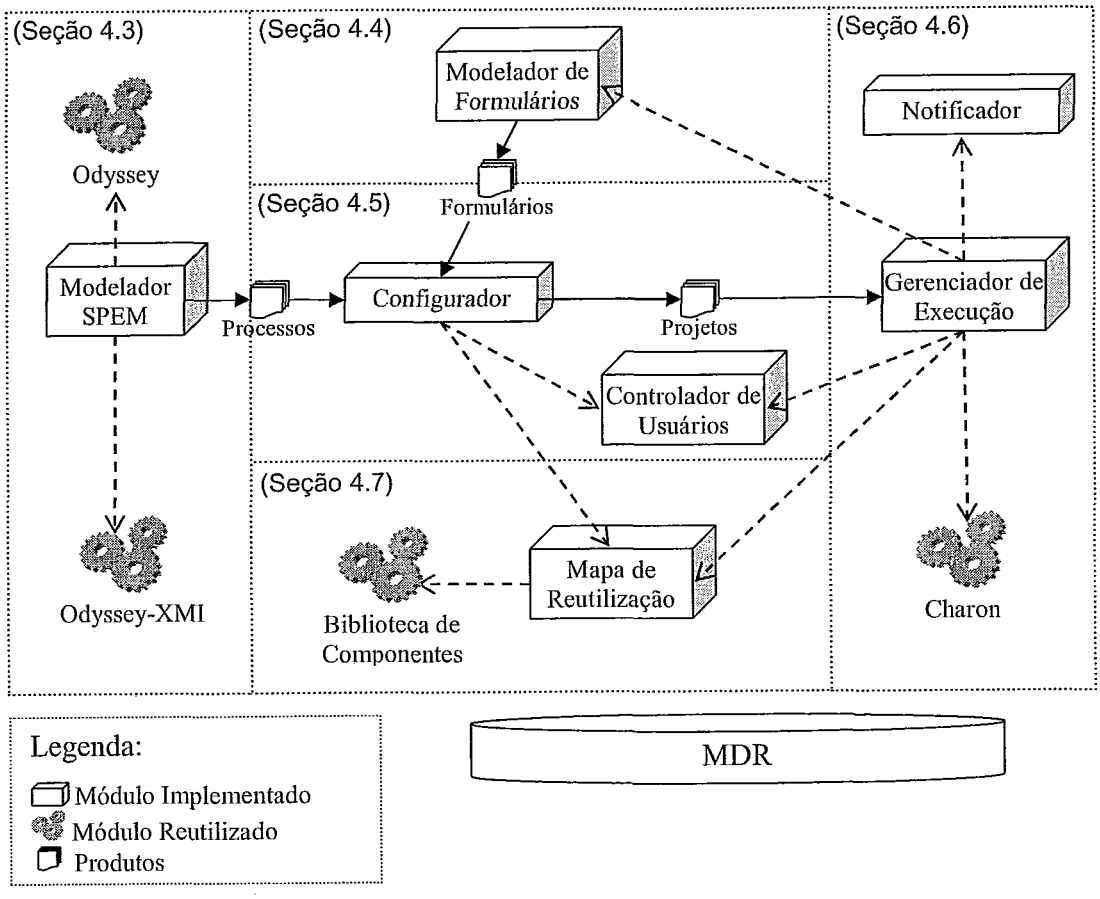


Fig. 4-1 - Principais módulos do Odyssey-CCS

O módulo *Configurador* é o responsável por dar apoio à implantação dos projetos para o controle de modificações. Ele depende de informações que são geradas por outros módulos, como o processo de controle de modificações a ser utilizado, os formulários de coleta de informações e os usuários e componentes candidatos a participarem do projeto. Os usuários do Odyssey-CCS são mantidos pelo componente *Controlador de Usuários*. Ambos os componentes são descritos na Seção 4.5.

O *Gerenciador de Execução* é o módulo responsável por dar apoio à execução dos processos. Ele utiliza a *Charon* (MURTA, 2002) como máquina de processos, e o módulo *Notificador* para enviar notificações durante a execução dos processos. Estes módulos são descritos na Seção 4.6.

Tanto o Configurador quanto o Gerenciador de Execução necessitam de informações relacionadas a componentes, que são mantidas pelo módulo chamado *Mapa de Reutilização*, implementado sobre uma *Biblioteca de Componentes*. Estes módulos são detalhados na Seção 4.7.

Todos os módulos implementados utilizam o repositório MDR (MATULA, 2006) para armazenamento dos dados, com exceção do Modelador SPEM, que utiliza o mecanismo de persistência do Odyssey.

4.3 - Modelador SPEM

O protótipo Odyssey-CCS dá apoio à atividade de modelagem e adaptação de processos de controle de modificações, descrita no Capítulo 3, através do módulo Modelador SPEM. Ele possui as características definidas na Seção 3.2.3, como recursos para a modelagem gráfica de processos através da especificação SPEM, mecanismos de persistência e a exportação do processo para utilização por outros sistemas e pessoas.

Este é o único módulo desenvolvido no escopo do protótipo que não foi projetado para utilização pela Internet. Ele foi implementado como um *plugin* do ambiente Odyssey, que é uma aplicação desenvolvida em Java (SUN, 2006a).

A especificação SPEM define um *UML Profile* e um metamodelo, como descrito anteriormente, na Seção 3.3.2, sendo que este último foi adotado na implementação deste módulo. A utilização do *UML Profile* para SPEM é interessante para a modelagem de processos, pois permite que eles sejam modelados através das ferramentas atuais de modelagem em UML, que são numerosas e muito utilizadas, desde que possuam o *UML Profile* para SPEM. Porém, para a execução dos processos, a melhor opção é a utilização do próprio metamodelo da SPEM, pois assim é possível manipular diretamente os elementos SPEM, e não os elementos da UML estereotipados. Neste último caso, seria necessário identificar qual elemento da SPEM cada elemento da UML está representando, através da identificação do estereótipo utilizado, o que é mais trabalhoso e torna o código-fonte que manipula os processos mais difícil de entender.

Por exemplo, são definidos os elementos Executor (*ProcessPerformer*) e Papel (*ProcessRole*) na especificação SPEM. Através do metamodelo SPEM, é possível

manipular objetos destes tipos diretamente. A Fig. 4-2-a exibe o Executor (*ProcessPerformer*) como elemento do metamodelo SPEM. Já através do *UML Profile*, os objetos manipulados nestes casos são do tipo Ator (*Actor*), definido na UML, marcados com os estereótipos *ProcessPerformer* e *ProcessRole*, respectivamente. A Fig. 4-2-b exibe o Executor (*ProcessPerformer*) definido como estereótipo de Ator (*Actor*), através da notação utilizada para declaração de estereótipos, definida na especificação da UML 1.4 (Seção 3.35.2) e utilizada na especificação SPEM.

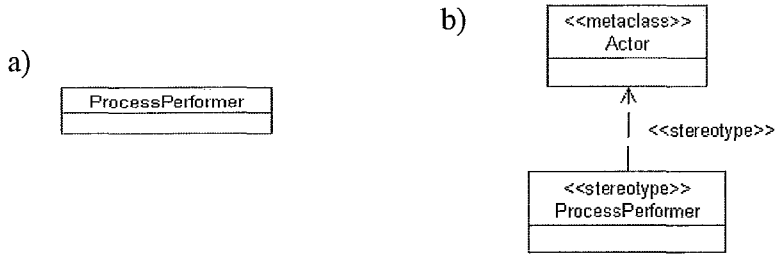


Fig. 4-2 - Representação do elemento Executor (*ProcessPerformer*) , a) como elemento do metamodelo SPEM, b) através do uso de estereótipo em um *UML Profile*

A seguir, são fornecidos mais detalhes sobre o Odyssey e seu mecanismo de extensão através de *plugins*, sobre a implementação do Modelador SPEM e sobre a biblioteca Odyssey-XMI, utilizada para a exportação dos modelos de processo SPEM.

4.3.1 - O Odyssey e seu mecanismo de extensão

O Odyssey é um ambiente de desenvolvimento de software com ênfase em reutilização (ODYSSEY, 2006), que visa dar apoio às atividades de Engenharia de Domínio (BRAGA, 2000) e de Engenharia de Aplicação (MILER, 2000). Este ambiente vem sendo desenvolvido pelo grupo de reutilização da COPPE/UFRJ desde 1998. Graças ao mecanismo de extensão e carga dinâmica oferecida pelo ambiente, introduzido com a reformulação do ambiente Odyssey (MURTA *et al.*, 2004), é possível acrescentar funcionalidades através de *plugins*, sem a necessidade de alteração do próprio ambiente. Ele já possui várias ferramentas desenvolvidas como *plugins*, que dão apoio a diversas atividades, como a documentação de componentes (MURTA *et al.*, 2001), suporte a padrões no projeto de software (DANTAS *et al.*, 2002) e engenharia reversa (VERONESE *et al.*, 2002), entre outras.

Os *plugins* são armazenados em um servidor, e podem ser instalados ou desinstalados a qualquer momento em cada instância do Odyssey. É mantido um arquivo de configuração que contém as informações de todos os *plugins* disponíveis, inclusive das suas dependências. Através de uma interface gráfica fornecida pelo

Odyssey, os *plugins* disponíveis são exibidos e podem ser selecionados e instalados. A instalação engloba o *download* do *plugin* selecionado e das bibliotecas das quais ele depende, caso elas ainda não estejam presentes na instância do Odyssey utilizado.

Para que o Odyssey possa se comunicar com um *plugin*, este deve implementar a interface *Tool* provida pelo ambiente. Esta interface, exibida na Fig. 4-3, possui métodos para acessar os menus específicos do *plugin* (métodos *get...Menu()*) e para indicar qual elemento do ambiente está selecionado (*setSelection (element:ModelElement) : void*).

Além disso, cada *plugin* deve ser empacotado como um arquivo JAR (*Java Archive*) e ter alguns atributos configurados no seu arquivo de manifesto, necessários ao mecanismo, como o nome da classe que implementa a interface *Tool*. Esta classe será acessada pelo Odyssey, através da API de reflexão do Java. O arquivo de configuração de *plugins* também deve ser atualizado com o nome, descrição, localização e dependências de cada *plugin*.

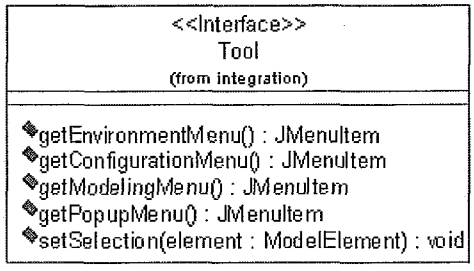


Fig. 4-3 - Interface *Tool* do Odyssey

Outra característica do ambiente que motivou a implementação do Modelador SPEM como um *plugin* do Odyssey é a existência de uma infra-estrutura para modelagem de diagramas, como diagrama de classes, de atividades e casos de uso, que pôde ser utilizada para permitir a modelagem dos diagramas presentes na especificação SPEM.

4.3.2 - Detalhes da implementação do Modelador SPEM

Para que seja possível a compreensão dos detalhes da implementação do Modelador SPEM, serão apresentados os elementos principais da referida infra-estrutura, exibidos na Fig. 4-4.

As informações utilizadas na infra-estrutura são mantidas hierarquicamente em uma árvore semântica de objetos. Todo objeto desta árvore é um *ModeloAbstrato*, classe

que representa um elemento de modelagem. A partir de qualquer um destes elementos é possível percorrer a árvore semântica, a fim de obter as informações necessárias.

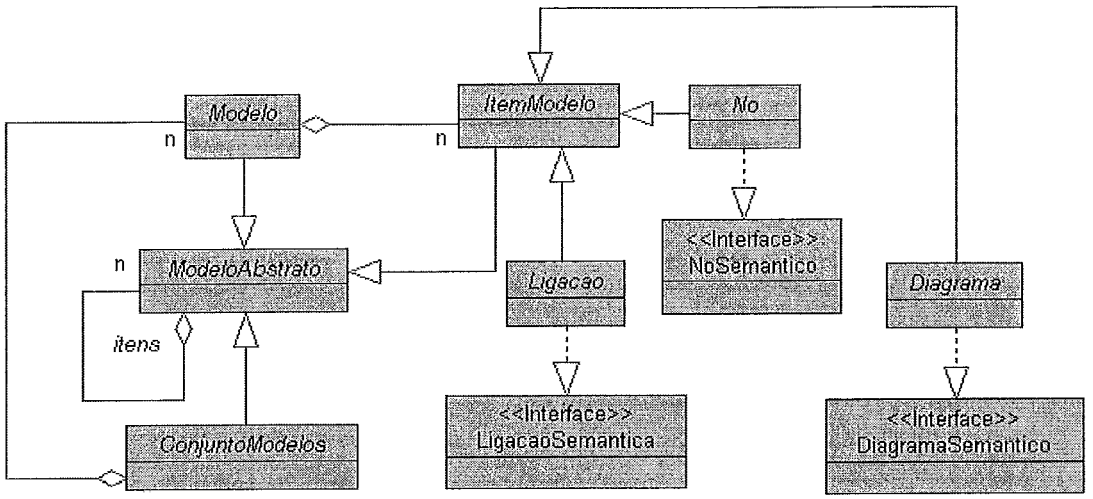


Fig. 4-4 - Principais classes do núcleo do Odyssey

A árvore semântica de objetos é organizada através de categorias de modelos, representadas pela classe *Modelo*. Exemplos destas categorias de modelo no Odyssey são a de casos de uso e a de classes, elementos definidos pela UML. Cada *Modelo* pode possuir diversos itens, representados pela classe *ItemModelo*. Exemplos destes itens são nós, ligações e diagramas, representados respectivamente pelas classes *No*, *Ligacao* e *Diagrama*. Todos os elementos específicos dos modelos possuem herança de *No*, como classes, estados, casos de uso, etc. Assim como todas as ligações herdam de *Ligacao*, como heranças, associações, agregações, etc., e todos os diagramas herdam de *Diagrama*, como diagrama de classes, de seqüência, de colaboração, etc. A raiz da árvore semântica é representada pela classe *ConjuntoModelos*.

Além dos elementos semânticos, o Odyssey possui elementos léxicos para a modelagem de diagramas, sendo que os principais são apresentados na Fig. 4-5. As classes *NoPadrao*, *ArestaPadrao* e *DiagramaPadrao* são representações léxicas dos elementos semânticos *No*, *Aresta* e *Diagrama*, respectivamente. São estes itens léxicos que formam os desenhos dos diagramas.

A classe *PainelDiagramador* é a responsável por apresentar o desenho de um diagrama léxico, ou seja, o desenho com todos os nós e arestas que o compõe. É interessante ressaltar que cada elemento léxico é responsável pelo seu próprio desenho. Um *PainelDiagramador* também possui objetos do tipo *AgenteDiagramacao*, responsáveis pela inserção, remoção e edição de itens nos diagramas.

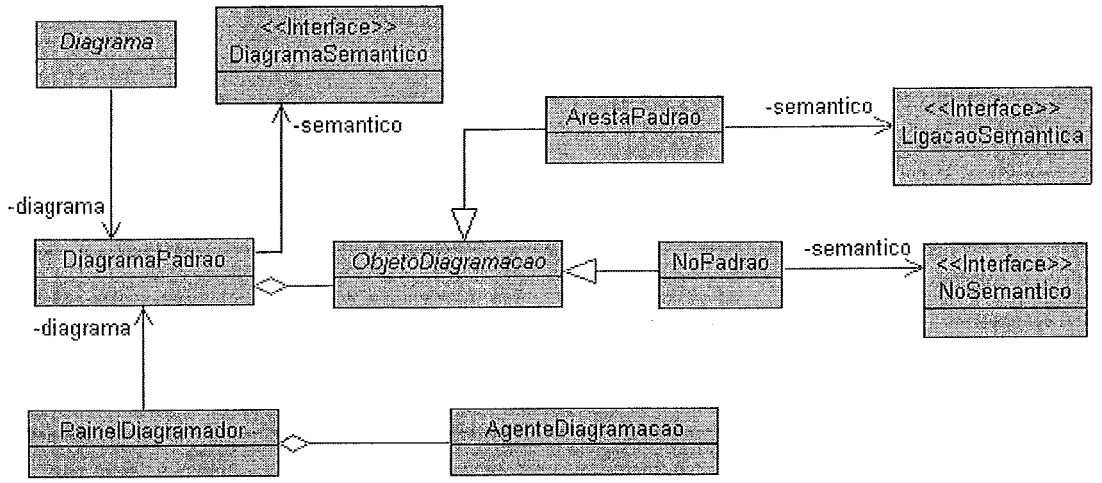


Fig. 4-5 - Principais classes de diagramação do Odyssey

A fim de oferecer recursos para a modelagem de processos, o Modelador SPEM introduz elementos da especificação SPEM através da extensão dos elementos semânticos e léxicos do Odyssey. A Fig. 4-6 exibe os principais elementos semânticos, e a Fig. 4-7 exibe os principais elementos léxicos. As classes da infra-estrutura do Odyssey que são estendidas possuem o fundo destacado, e as classes introduzidas que representam os elementos SPEM possuem o fundo branco. Apenas os elementos da SPEM considerados essenciais à modelagem de processos de controle de modificações foram implementados.

Os diagramas adaptados para a SPEM são o de atividades, o de casos de uso e o de classes, representados na Fig. 4-6 pelas classes *DiagramaAtividadesSpem*, *DiagramaCasosUsoSpem* e *DiagramaClassesSpem*, respectivamente. Cada um destes diagramas possui o seu próprio diagramador, exibidos na Fig. 4-7, estendendo *PainelDiagramador*, que são as classes que apresentam o desenho dos diagramas.

Os elementos semânticos implementados estendem as classes *No* e *Ligacao*. Estes elementos são: *Macroatividade*, *Atividade*, *Produto*, *Executor*, *Papel*, *Guia* e *Processo*, entidades descritas na Seção 3.2.1, e as ligações que representam fluxo de controle e de dados, e execução e assistência às atividades, respectivamente dos diagramas de atividades e de casos de uso.

Cada um destes elementos possui também sua representação léxica, responsável pelo desenho do elemento em um diagrama. Estes elementos são apresentados na Fig. 4-7. As entidades e ligações estendem, respectivamente, as classes *NoPadrao* e *ArestaPadrao*.

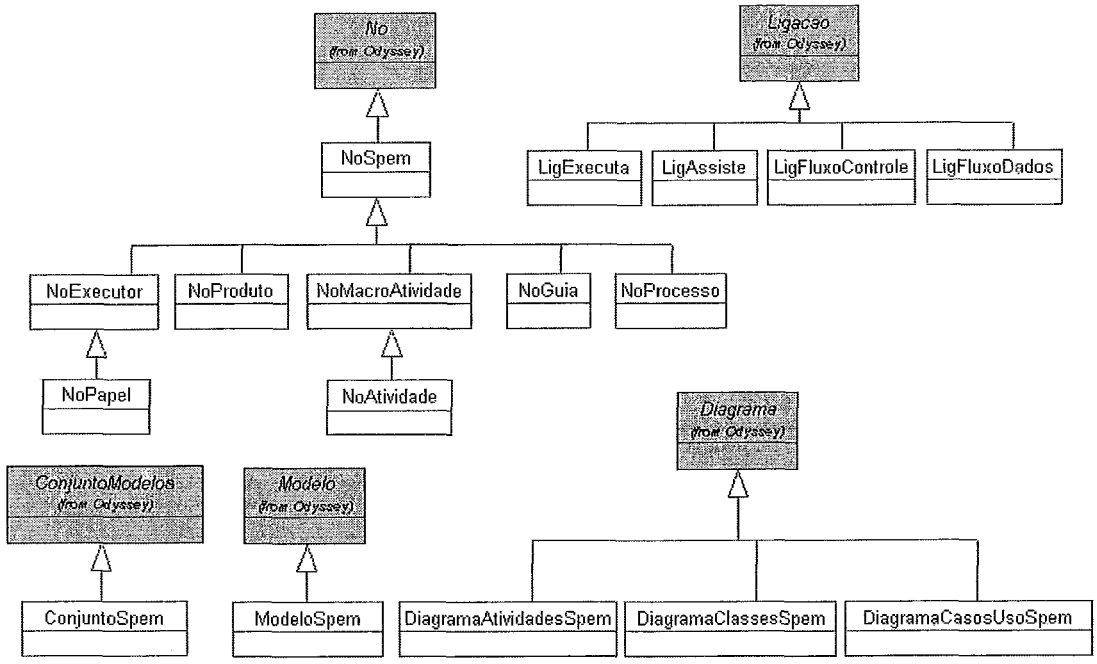


Fig. 4-6 - Principais classes da representação semântica do Modelador SPEM

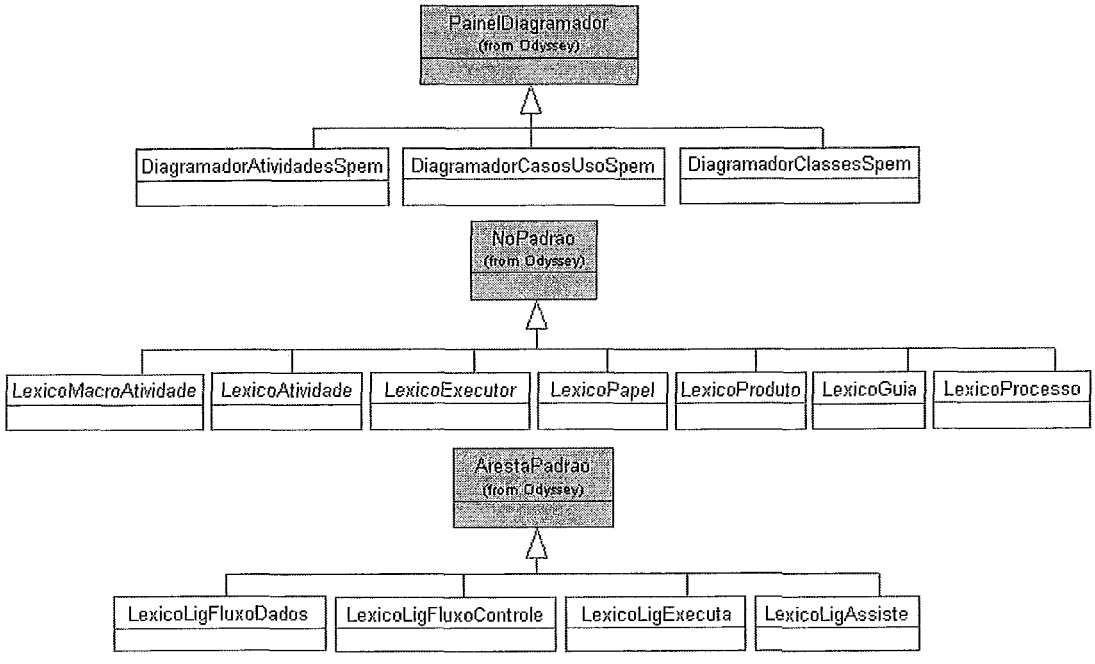


Fig. 4-7 - Principais classes da representação léxica do Modelador SPEM

A Fig. 4-8 exibe a tela principal do Modelador SPEM. Ela segue a mesma estrutura de telas do Odyssey. Na parte superior está a Barra de Ferramentas, onde ficam disponíveis os tipos de elementos que podem ser utilizados na modelagem. Eles são disponibilizados dinamicamente de acordo com o item selecionado no Painel de

Objetos, que contém a árvore de elementos semânticos instanciados, localizado à esquerda da tela. Do lado direito está o Painel de Diagramação, onde são desenhados os diagramas. No painel presente nesta figura é exibido um diagrama de atividades.

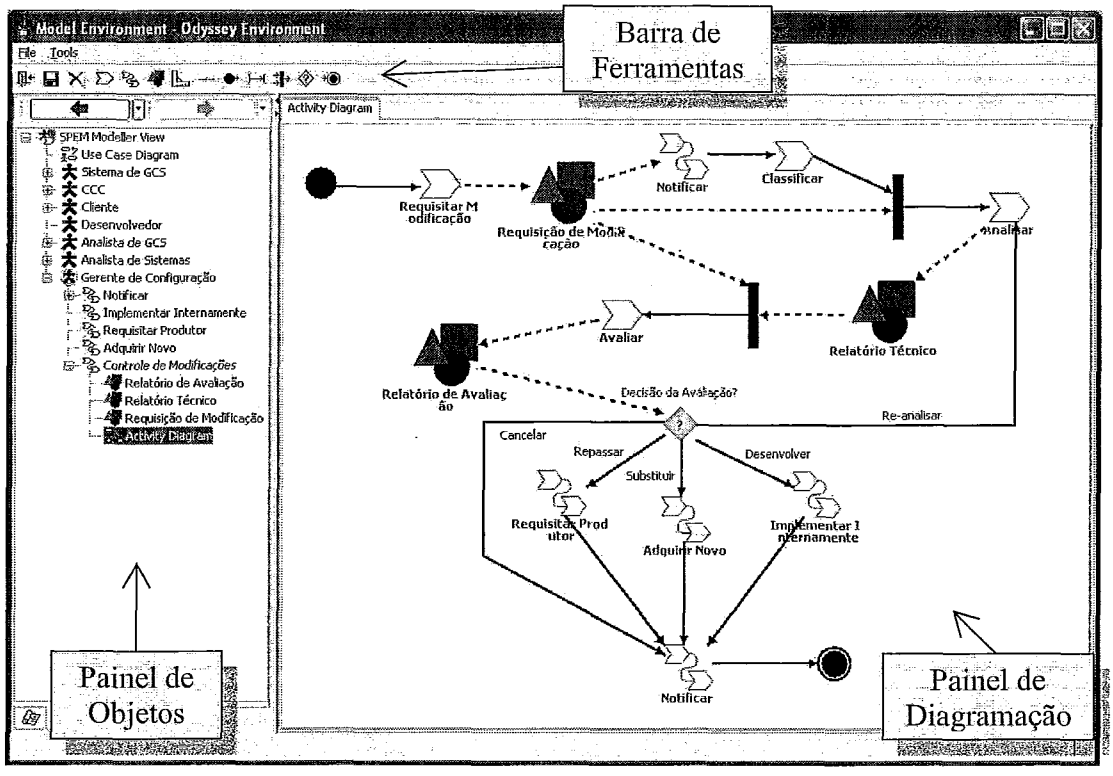


Fig. 4-8 - Tela do *plugin* Odyssey-CCS

Através da Barra de Ferramentas, o usuário seleciona os tipos de elementos que deseja inserir no processo que está sendo modelado. Cada elemento selecionado é instanciado e disponibilizado no Painel de Objetos. Os elementos são organizados hierarquicamente, de acordo com a estrutura de composição definida na especificação SPEM. Para cada tipo de objeto selecionado no Painel de Objetos, um Painel de Diagramação é exibido do lado direito da tela. Se o objeto selecionado for do tipo *Diagrama*, o Painel de Diagramação oferecerá os recursos para desenho. Se for do tipo *No*, o painel exibirá as informações deste objeto para edição. Os elementos do tipo *Ligacao* não são exibidos no Painel de Objetos.

Na Fig. 4-8, o Painel de Diagramação exibe um Diagrama de Atividades adaptado para a SPEM, utilizado para modelar a seqüência de atividades e os produtos gerados e consumidos em um processo. Neste caso específico, está modelado o *Processo Exemplo* definido no Capítulo 3, no contexto da Macroatividade *Controle de Modificações*. Neste tipo de diagrama, podem ser utilizados os seguintes tipos de

elementos para modelagem: Macroatividade, Atividade, Produto, Guia, Início de Processo, Sincronismo, Decisão, Fim de Processo e Fluxo de Controle e de Controle e Dados.

É importante destacar que toda Macroatividade possui um Diagrama de Atividades para a modelagem da seqüência e dos produtos de suas próprias Atividades e Macroatividades. No diagrama apresentado na Fig. 4-8, por exemplo, são exibidas quatro Macroatividades diferentes (*Notificar*, *Requisitar Produtor*, *Adquirir Novo* e *Implementar Internamente*). Cada uma delas possui um Diagrama de Atividades próprio, onde são detalhadas.

Já o Diagrama de Casos de Uso, utilizado para a modelagem das responsabilidades sobre as Atividades e Macroatividades, aceita os seguintes tipos de elementos: Executor, Papel, Atividade, Macroatividade, Guia e ligações do tipo Executa e Assiste.

No caso do Diagrama de Classes, utilizado para a definição de responsabilidades sobre os Produtos e para a representação da estrutura, decomposição e dependências entre os Produtos, são aceitos os tipos Executor, Papel, Produto, Processo, Guia, Associação e Herança.

4.3.3 - Exportação de modelos de processos

Os processos construídos no Modelador SPEM podem ser exportados segundo o padrão XMI (OMG, 2005), permitindo a reutilização destes por outros sistemas compatíveis com o metamodelo SPEM. A implementação desta funcionalidade foi feita reutilizando-se o módulo Odyssey-XMI, utilizado também em outros projetos (DANTAS, 2005; OLIVEIRA, 2005).

O Odyssey-XMI faz o mapeamento de objetos do *Odyssey* para objetos de um metamodelo baseado no MOF, que são exportados em arquivos XMI com o auxílio do repositório MDR (MATULA, 2006). Este mapeamento é realizado por tratadores específicos dos objetos em questão.

No caso específico do Modelador SPEM, foram implementados os tratadores que manipulam os objetos que representam elementos da SPEM no *Odyssey* (e.g. *NoAtividade*) e os mapeiam nos próprios objetos do metamodelo SPEM (e.g. *Activity*), acessados através de interfaces JMI (*Java Metadata Interface*) (DIRCKZE, 2002), que são interfaces em linguagem Java geradas a partir de metamodelos MOF. Este conjunto

de tratadores é registrado, juntamente com o metamodelo SPEM, no Odyssey-XMI, que se encarrega da exportação do processo modelado no padrão XMI.

4.4 - Modelador de formulários

O protótipo Odyssey-CCS auxilia a atividade de modelagem de formulários para a coleta de informações, definida no Capítulo 3, através do módulo Modelador de Formulários, desenvolvido como uma aplicação *Web*, utilizando-se tecnologias da especificação J2EE 1.4 (SUN, 2006b), como JSP, Servlets e EJB.

Além da geração de formulários, este módulo permite o armazenamento e recuperação das informações coletadas, agrupadas logicamente em documentos e modificações, conforme explicado na Seção 3.3.

As principais classes do Modelador de Formulários são exibidas na Fig. 4-9. A estrutura das entidades conceituais foi mantida, conforme definição feita na Seção 3.3.1, ou seja, formulários são constituídos por campos, que são utilizados para a coleta de informações. Estas são agrupadas em documentos, específicos para cada formulário. Há vários tipos de campos, exibidos no diagrama como herança da classe *Campo*. Estas classes, presentes no pacote *Persistencia*, e marcadas no diagrama como *(from Persistencia)*, foram implementadas como entidades EJB BMP. Este detalhe não foi modelado no diagrama para não deixá-lo muito carregado.

As classes *GerenciadorFormularios* e *GerenciadorDocumentos* são responsáveis pela manipulação dos formulários e dos documentos e modificações, respectivamente. Eles executam ações como listar, buscar e codificar objetos em determinada linguagem (e.g. HTML), entre outras, preparando as informações para apresentação ou para persistência. Eles foram implementados como sessões EJB. A classe *FachadaModeladorFormularios* é o ponto de acesso ao módulo, que pode ser acessada por outros módulos, e pela camada de apresentação. Ela foi implementada segundo o padrão de projeto *Facade* (GAMMA *et al.*, 1995), para ser o único ponto de acesso ao Modelador de Formulários.

O pacote Sintaxe possui as classes responsáveis pela codificação de objetos utilizando determinada sintaxe. Neste caso, a sintaxe utilizada é a da linguagem HTML. Estas classes são exibidas na Fig. 4-10.

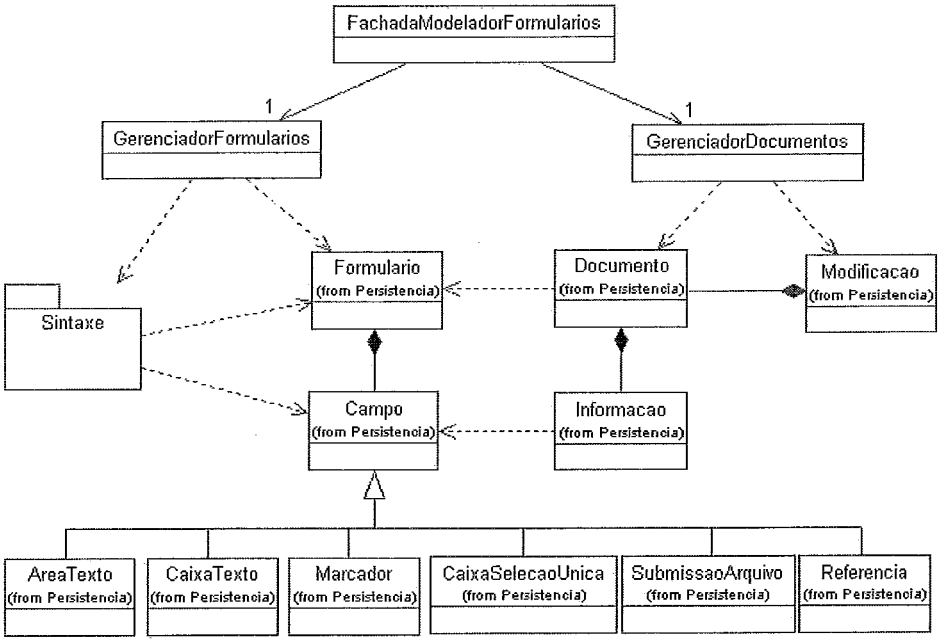


Fig. 4-9 - Principais classes do Modelador de Formulários

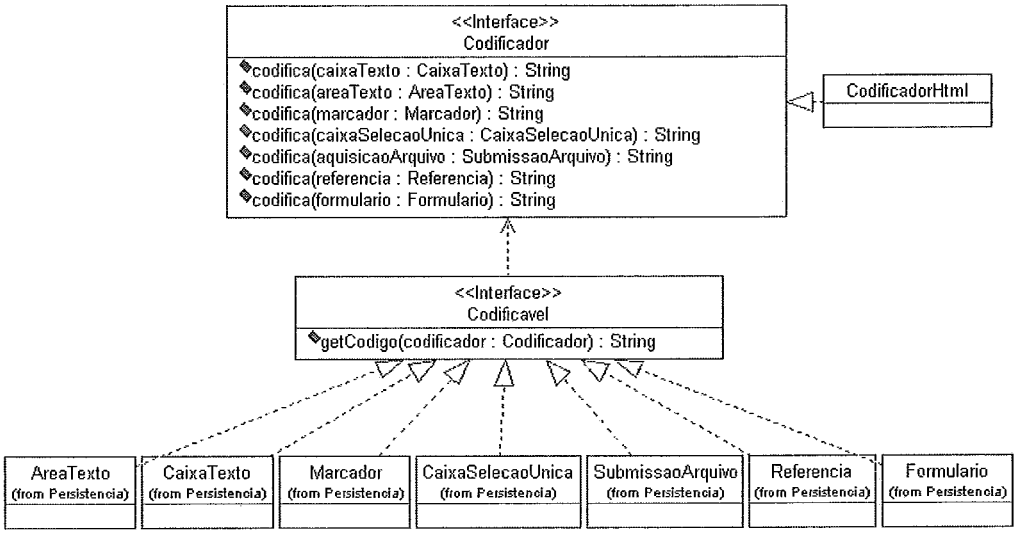


Fig. 4-10 - Principais classes do pacote Sintaxe

Esta implementação segue o padrão de projeto *Visitor* (GAMMA *et al.*, 1995), que permite a definição de operações auxiliares para um conjunto de classes sem que elas sejam alteradas, evitando a poluição destas classes. As operações de codificação são definidas na interface *Codificador* e implementadas por classes específicas, como a *CodificadorHtml*. Para a codificação das informações dos objetos em outras linguagens, pode-se criar novas classes específicas que estendam *Codificador*.

Para codificar um objeto, o *GerenciadorFormularios* o acessa através da interface *Codificavel*, utilizando o método *getCodigo(codificador: Codificador): String*,

que invoca o método específico de codificação do objeto, implementado pelo *codificador* passado como parâmetro.

A Fig. 4-11 exibe a tela para a criação de campos em um formulário. Neste caso específico, está sendo modelado um formulário para a coleta de informações da atividade de Requisição de Modificação, presente no processo modelado no exemplo da Seção 3.2.2.

O Gerente de Configuração seleciona o tipo de campo a ser inserido (Fig. 4-11-a) e o configura, através de sua respectiva tela de configuração (Fig. 4-12). Uma vez inserido, o campo é exibido na área de Prévia do Formulário (Fig. 4-11-c), que permite ao Gerente visualizar o formato final do formulário. Esta tela fornece ainda recursos para a exclusão de campos e reposicionamento no eixo vertical (Fig. 4-11-b).

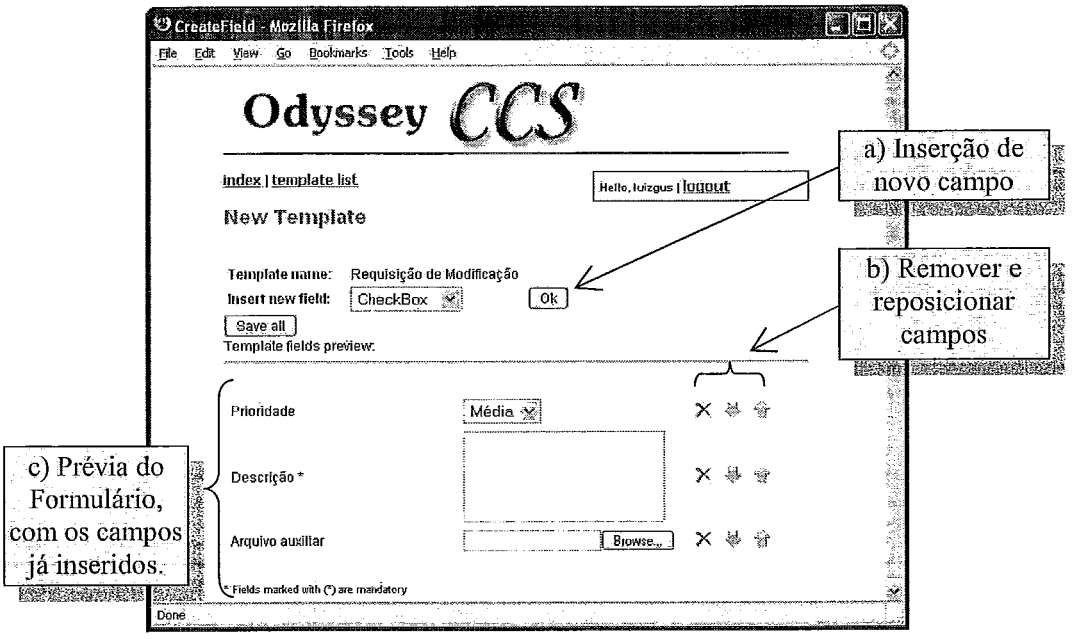


Fig. 4-11 - Tela para criação de um formulário

Na Fig. 4-12 são exibidas as telas de configuração para cada tipo de campo, cada uma marcada com uma letra. Cada tela é explicada a seguir:

a) Tela para a configuração do tipo de campo Marcador. Na linguagem HTML, ele é representado pelo elemento *Input* do tipo *Checkbox*. Deve-se configurar o rótulo e o valor inicial, que pode ser marcado ou desmarcado.

b) Tela para a configuração de uma Referência (*Link*). Na linguagem HTML, este tipo de campo é representado pelo elemento *Anchor* (*A HREF*). Deve ser atribuído a ele o rótulo e o endereço do elemento referenciado (URI).

c) Tela para a configuração de Caixa de Seleção Única. Na linguagem HTML, este tipo de campo é representado pelo elemento *Select.* Deve-se configurar o rótulo, a opção inicial e as opções existentes, separadas por vírgulas.

d) Tela para a configuração de Área de Texto, tipo de campo representado pelo elemento *Textarea* na linguagem HTML. Deve-se configurar o rótulo, o valor inicial, o tamanho, e se o campo é de preenchimento obrigatório ou não. Se for, seu rótulo será exibido com um asterisco.

e) Tela para a configuração de campo do tipo Caixa de Texto, representado pelo elemento *Input* do tipo *Text* na linguagem HTML. Deve-se configurar o nome, o valor inicial, o tipo de dados aceito, o tamanho e a obrigatoriedade de preenchimento.

f) Tela para a configuração de Submissão de Arquivo. Na linguagem HTML, este tipo de campo é representado pelo elemento *Input* do tipo *File.* Deve-se configurar o rótulo do campo. Além do rótulo, este campo é exibido com uma Caixa de Texto para inserção do caminho do arquivo, e com um botão, que chama uma tela de seleção de arquivos.

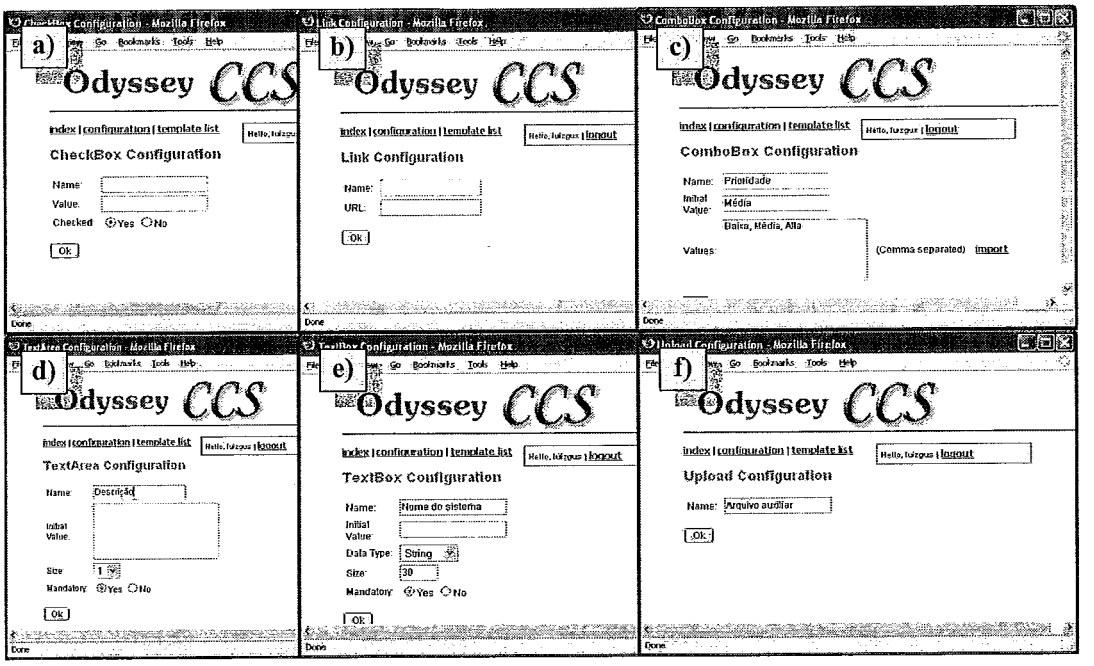


Fig. 4-12 - Telas para criação de campos

Os formulários modelados são listados na tela exibida na Fig. 4-13. Nela, o usuário pode optar pela edição, remoção e inserção de formulários.

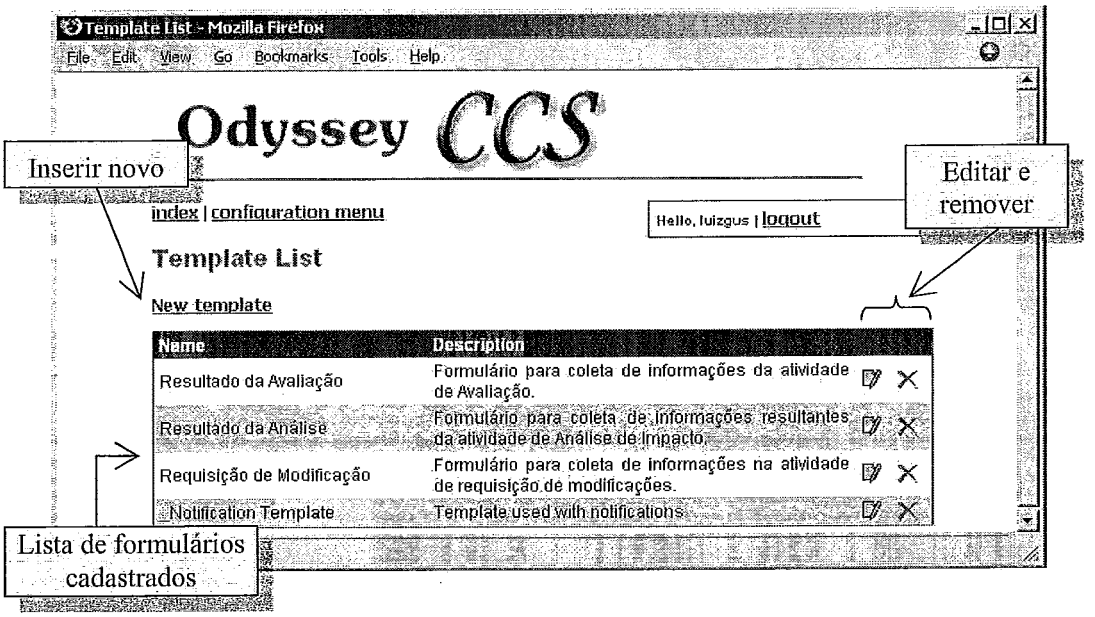


Fig. 4-13 - Tela de listagem de formulários

4.5 - Configurador de projetos

O módulo *Configurador* auxilia o Gerente de Configuração na criação de um projeto para controle de modificações em um determinado sistema de software. Cada projeto mantém um processo de controle de modificações configurado, ou seja, com usuários assumindo as responsabilidades definidas no processo e com formulários de coleta de dados associados aos produtos das atividades. Além disso, ele mantém informações sobre os componentes utilizados pelo sistema de software e sobre como as notificações devem ser gerenciadas.

O Configurador é composto pela classe *GerenteConfiguracao*, implementada como sessão EJB, que gerencia toda a configuração do projeto, por uma camada de apresentação, que é composta por JSP's e classes Servlets, e pela camada de persistência. Esta última é composta por classes implementadas como entidades EJB BMP associadas a classes implementadas segundo o padrão de projeto DAO (ALUR *et al.*, 2001), responsáveis por armazenar as configurações realizadas no repositório MDR.

A configuração de um projeto é realizada em quatro etapas. Na primeira, deve-se informar o nome e a descrição do Projeto e o processo de controle de configurações a ser utilizado. Este processo deve ser compatível com a especificação SPEM e ter sido escrito em arquivo em conformidade com a especificação XMI. Pode-se utilizar o Modelador SPEM para a modelagem do processo, ou outra ferramenta que preencha estes requisitos. O processo fornecido é interpretado e simulado pela máquina de

processos Charon, descrita na Seção 4.6.1. Se algum erro no processo for detectado pela simulação, este será informado ao usuário para que ele corrija o processo modelado. A Fig. 4-14 exibe a tela para execução desta primeira etapa.

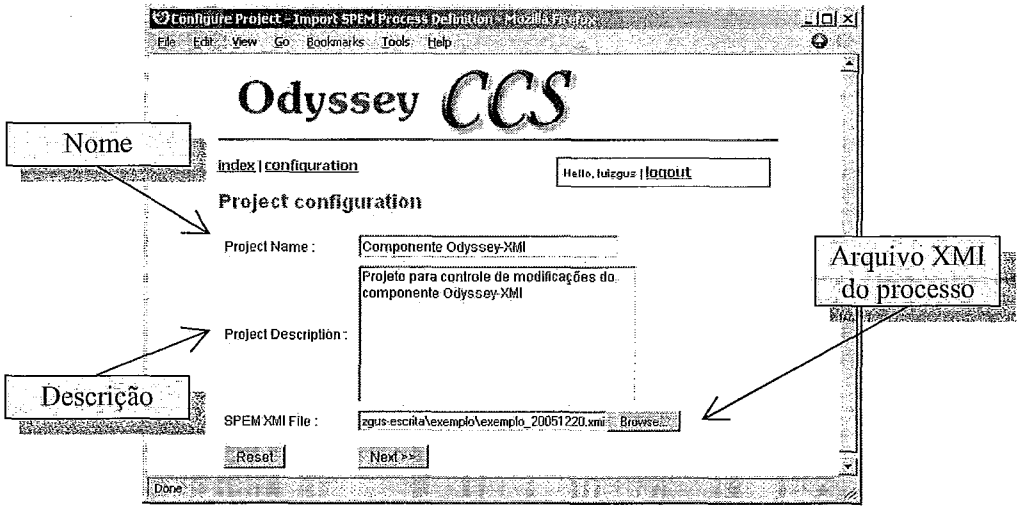


Fig. 4-14 - Tela para criação de projeto - Etapa 1 da configuração de projeto

Na segunda etapa, deve-se atribuir responsabilidades do processo às pessoas que participarão da GCS no respectivo projeto. A Fig. 4-15 exibe uma tabela com duas colunas. Na primeira, são exibidos os papéis presentes no processo a ser utilizado, obtidos do arquivo XMI importado na etapa anterior da configuração do projeto. Na segunda coluna, são exibidos os usuários cadastrados no Odyssey-CCS, e o usuário especial *iniciador* (*_owner*), descrito na Seção 3.5.2. Para cada papel presente na primeira coluna, deve-se selecionar uma ou mais pessoas da segunda coluna, lembrando-se que uma pessoa pode assumir mais de um papel. Esta configuração possibilita a disponibilização e envio de informações relativas à execução do processo às pessoas envolvidas e o controle de permissão de execução das atividades e decisões, características discutidas na Seção 4.6.

Para acessar as informações sobre as pessoas cadastradas no Odyssey-CCS, o Configurador utiliza o Controlador de Usuários, que é o módulo responsável pelo cadastro de usuários. Cada usuário possui as seguintes informações: Nome, identificador, endereço eletrônico e senha. A Fig. 4-16-a exibe a tela para listagem dos usuários cadastrados, que também permite incluir e remover usuários. A Fig. 4-16-b exibe a tela para o cadastro de usuário.

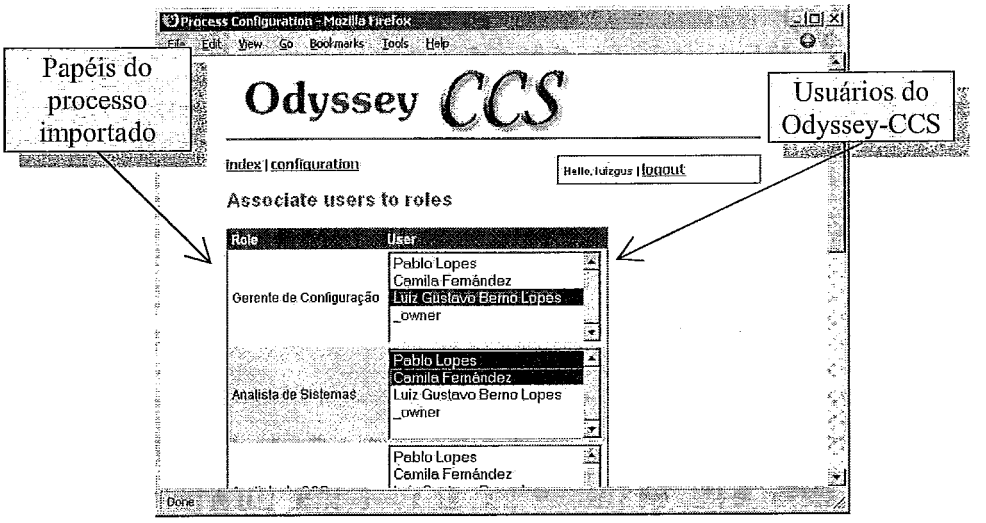


Fig. 4-15 - Tela para configuração de responsabilidades - Etapa 2 da configuração de projeto

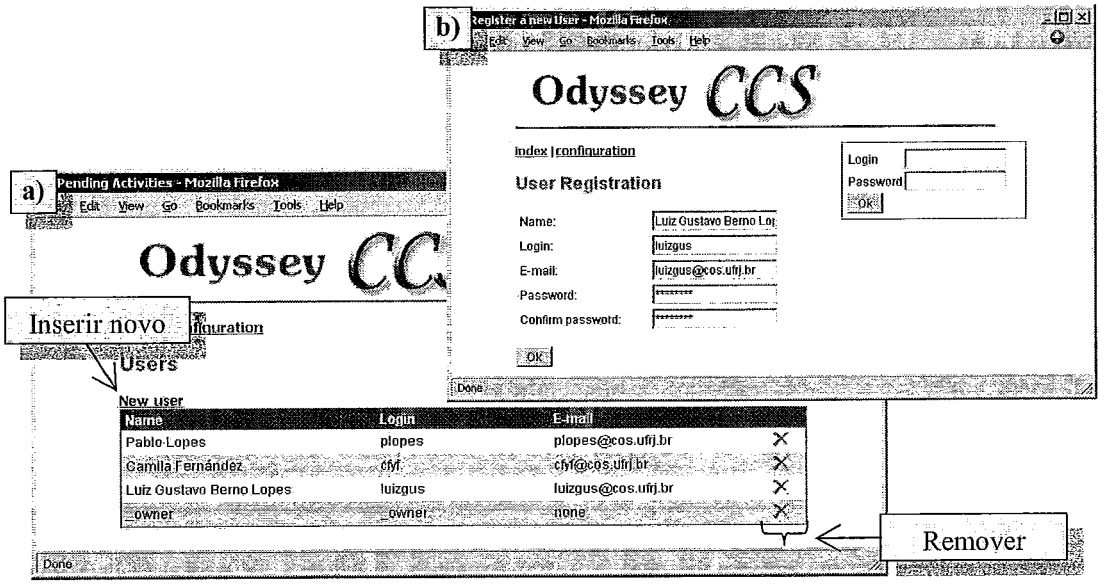


Fig. 4-16 - a) Tela de listagem de usuários, b) Tela de cadastro de usuário

Na terceira etapa da configuração, pode-se atribuir um formulário a cada produto do processo. A Fig. 4-17 exibe a tela onde as associações são feitas. Na coluna da esquerda da tabela, são exibidos os produtos existentes no processo a ser utilizado, e na coluna da direita são exibidos os formulários cadastrados através do módulo Modelador de Formulários. Para cada produto, pode-se selecionar um formulário, que será exibido ao final da execução da atividade na qual o produto é criado.

Na quarta e última etapa, é feita a configuração das notificações a serem enviadas durante a execução do processo. A cada produto do processo que representa uma notificação, deve-se associar o formulário especial *_NotificationTemplate*,

fornecido pelo Odyssey-CCS. Por exemplo, na Fig. 4-17, o produto *E-mail* é associado a este formulário especial. O Configurador detecta os produtos que representam notificações e habilita a configuração para cada um deles. A Fig. 4-18 exibe a tela para a configuração de uma notificação.

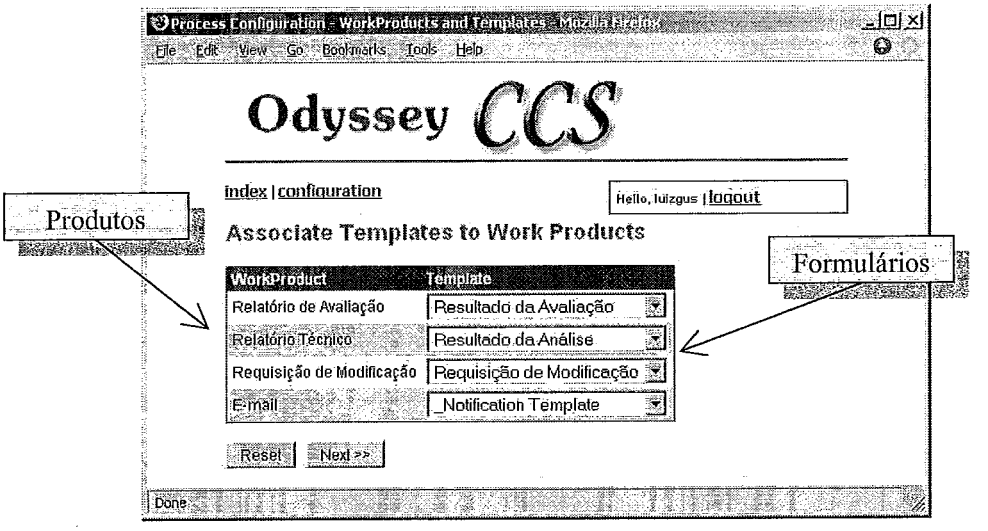


Fig. 4-17 - Tela para associação de formulários a produtos - Etapa 3 da configuração de projeto

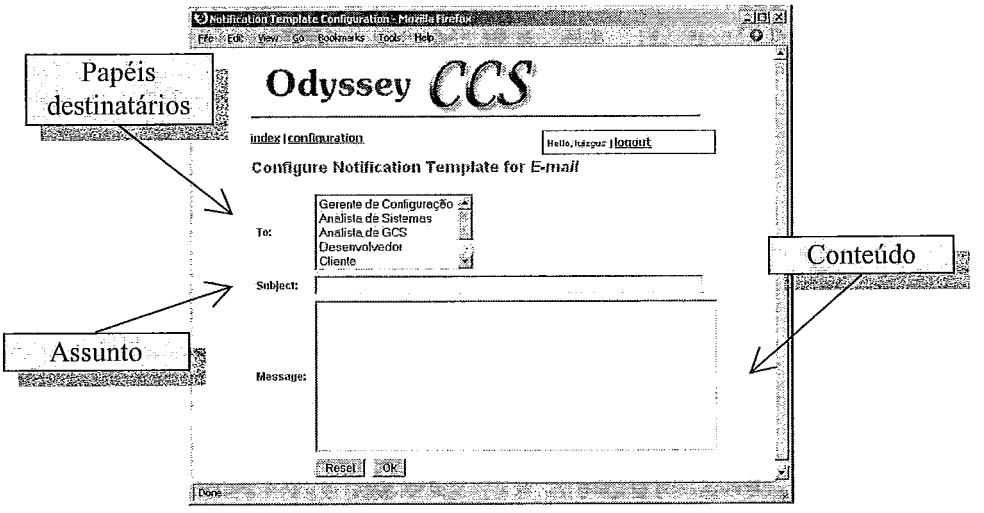


Fig. 4-18 - Tela do formulário especial para configuração de notificação - Etapa 4 da configuração de projeto

Esta configuração engloba o preenchimento das seguintes informações: Os destinatários, o assunto e o conteúdo da mensagem. O destinatário é informado por meio dos papéis do processo. Dessa forma, todas as pessoas que desempenham os papéis selecionados como destinatários recebem a notificação quando a atividade que gera o produto associado ao formulário de notificação é finalizada. O assunto e o

conteúdo da mensagem são informações do tipo texto. Mais detalhes sobre o envio de notificações são discutidos na Seção 4.6.

4.6 - Gerenciador de execução

Após a configuração de um projeto, pode-se dar início à execução de instâncias de seu processo, que é realizada principalmente com o apoio do módulo Gerenciador de Execução, também desenvolvido como uma aplicação *Web*, utilizando-se tecnologias da especificação J2EE 1.4 (SUN, 2006b), como JSP, Servlets e EJB.

Este módulo é responsável pelo controle do ciclo de vida das instâncias de execução de processos, auxiliado pela máquina de processos *Charon* (MURTA, 2002), descrita na Seção 4.6.1. Ele também exibe as atividades e decisões pendentes para cada usuário e os formulários de coleta de informações, armazenando as informações preenchidas com auxílio do módulo Modelador de Formulários. Além disso, ele permite a visualização das informações coletadas e identifica os momentos de envio de notificações, sendo o envio destas de responsabilidade do módulo *Notificador*.

A principal classe do Gerenciador de Execução é a *GerenciadorExecucao*, implementada como uma sessão EJB, que gerencia a execução de processos. Ela é exibida com seus principais métodos na Fig. 4-19, sem o estereótipo de sessão EJB. Este módulo também é composto por uma camada de apresentação, composta por JSP's e classes Servlets, e por uma camada de persistência, composta por classes responsáveis por armazenar as informações sobre a execução das informações no repositório MDR.

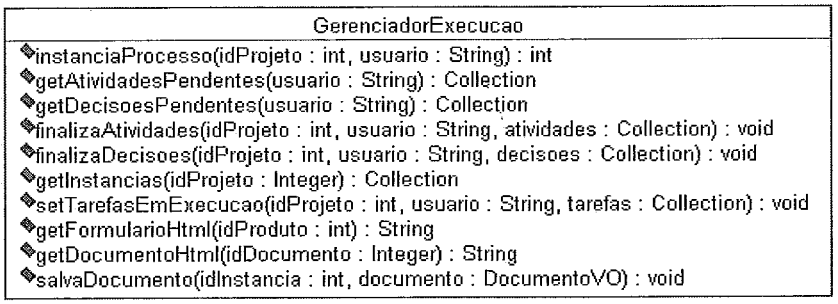


Fig. 4-19 - Classe *GerenciadorExecucao*

A classe *GerenciadorExecucao* é responsável pela instanciação de processos para execução, implementada através do método *instanciaProcesso(idProjeto: int, usuario: String): int*. Cada instância de processo é executada dentro do contexto de um projeto, identificado pelo parâmetro *idProjeto*, e é armazenado também quem solicita a instanciação. Este usuário passa a ser tratado como um usuário especial para esta instância de processo, chamado *iniciador* (ou *_owner*), que é descrito na Seção 3.5.2.

Esta classe também retorna as atividades e decisões pendentes para cada usuário, ações implementadas respectivamente pelos métodos *getAtividadesPendentes(usuario: String): Collection* e *getDecisoesPendentes(usuario: String): Collection*, e as finaliza. Todos estes métodos utilizam a máquina de processos *Charon*, descrita na Seção 4.6.1.

Outra responsabilidade da classe *GerenciadorExecucao* é marcar tarefas, sejam atividades ou decisões, que estejam sendo executadas por determinado usuário, para que outros usuários fiquem cientes deste fato. Isto é realizado através do método *setTarefasEmExecucao(idProjeto: int, usuario: String, tarefas: Collection): void.*

Outras operações executadas por esta classe estão relacionadas à coleta de informações após a finalização das atividades e posterior visualização das mesmas. Ela fornece o formulário associado a um determinado produto presente em um processo, na linguagem HTML, salva as informações coletadas através de formulários, agrupadas em documentos, e recupera um documento com suas informações, também formatado na linguagem HTML. Estas operações são executadas com o auxílio do módulo Modelador de Formulários, descrito na Seção 4.4.

A seguir, são fornecidos detalhes sobre a máquina de processos utilizada, a *Charon*, sobre o módulo responsável pelas notificações, o *Notificador*, e sobre as interfaces gráficas que auxiliam os usuários durante a execução dos processos.

4.6.1 - Máquina de processos Charon

A máquina de processos utilizada pelo Odyssey-CCS é a Charon (MURTA, 2002), desenvolvida no contexto de um trabalho de pesquisa de mestrado realizado na COPPE/UFRJ. Ela foi escolhida por satisfazer os requisitos definidos pela abordagem descrita no Capítulo 3, e por ser bem conhecida pelos integrantes do Grupo de Reuso da COPPE/UFRJ, ambiente no qual tanto a Charon quanto este trabalho foram desenvolvidos.

A Charon possibilita a instanciação, simulação, execução, monitoramento e evolução de processos de software reutilizáveis. Ela utiliza a tecnologia de agentes (JENNINGS *et al.*, 1998) para a simulação, a execução e o acompanhamento dos processos. Além disso, provê uma arquitetura que permite a criação de novos agentes, permitindo a sua extensão.

A versão da máquina de processos Charon utilizada neste trabalho é uma implementação adaptada daquela detalhada por MURTA (2002). Ela foi reestruturada e transformada em uma biblioteca, já que anteriormente fazia parte do núcleo do ambiente

Odyssey (2006). Além disso, ela foi adaptada para compreender a notação SPEM, pois anteriormente a notação utilizada por ela era baseada no Diagrama de Atividades da UML.

A Fig. 4-20 exibe a classe *FachadaCharon*, único ponto de acesso à Charon, que implementa o padrão de projeto *Facade* (GAMMA *et al.*, 1995). São exibidos também os seus principais métodos.

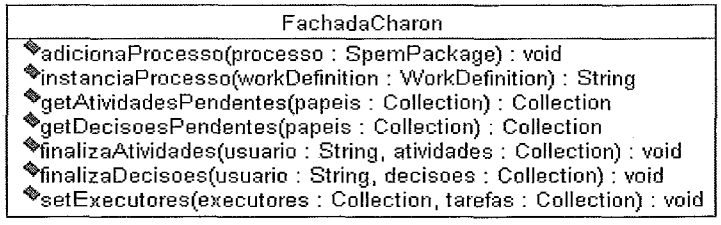


Fig. 4-20 - Classe de acesso à máquina de processos Charon

Para adição de um novo processo à Charon, é utilizado o método *adicionaProcesso(processo: SpemPackage): void*, sendo que o processo é representado por um conjunto de interfaces JMI agrupadas em um pacote do tipo *SpemPackage* (OMG, 2005). Este pacote é gerado a partir do arquivo XMI com a definição do processo, que é importado pelo módulo Configurador, descrito na Seção 4.5. Este método também é responsável pela simulação do processo sendo adicionado. Caso o processo possua algum erro, este é relatado para que o usuário possa corrigi-lo. Este método é utilizado na atividade de Configuração do processo (Seção 4.5).

Uma vez adicionados, os processos presentes na Charon podem ser instanciados. Isto é feito através do método *instanciaProcesso(workDefinition: WorkDefinition): String*.

Após a instanciação, os processos entram em execução e a Charon pode informar quais atividades e decisões estão pendentes para os papéis indicados, respectivamente através dos métodos *getAtividadesPendentes(papeis: Collection): Collection* e *getDecisoesPendentes(papeis: Collection): Collection*. Ao final da execução, as atividades e decisões podem ser finalizadas através dos métodos *finalizaAtividades(usuario: String, atividades: Collection): void* e *finalizaDecisoes(usuario: String, decisoes: Collection): void*. É informado o usuário que solicitou a finalização, e a Charon também observa o tempo decorrido ao longo da execução.

Além disso, a Charon pode manter informações sobre os usuários que estão executando determinada atividade ou decisão. Isso é solicitado através do método *setExecutores(executores: String, tarefas: Collection): void.*

Mais detalhes sobre a máquina de processos Charon podem ser encontrados em (MURTA, 2002).

4.6.2 - Notificação automática

O Odyssey-CCS oferece recursos de notificação às pessoas autorizadas e interessadas no andamento dos processos de controle de modificações. As notificações são realizadas através do envio de mensagens eletrônicas.

Para que as notificações sejam enviadas automaticamente durante a execução dos processos, faz-se necessária a configuração prévia do momento e das informações a serem enviadas. Isto é feito durante a atividade de Configuração através do módulo Configurador, detalhado na Seção 4.5.

Durante a execução dos processos, o módulo Gerenciador de Execução acessa o módulo *Notificador*, que é responsável pelo armazenamento das informações de configuração de notificação e pelo envio das notificações. Este módulo possui uma camada de persistência e uma classe responsável pelo envio das notificações, que possui um fluxo de execução (*thread*) próprio.

4.6.3 - Visualização da execução de processos

O Gerenciador de Execução possui algumas interfaces gráficas para a interação com os usuários do sistema. Os projetos criados na atividade de configuração são exibidos na tela presente na Fig. 4-21. A partir dela, é possível criar uma nova instância do processo do projeto para execução e visualizar os documentos com as informações coletadas durante cada instância do processo.

As instâncias dos processos em execução podem ser visualizadas a partir da tela de visualização de atividades e decisões pendentes, exibida na Fig. 4-22. Nesta tela, são exibidas as tarefas pendentes para um determinado usuário. Neste caso específico, o usuário possui o identificador *luizgus*. Há duas tabelas, sendo que cada linha representa uma instância de processo em execução, que possui uma tarefa pendente para o usuário autenticado.

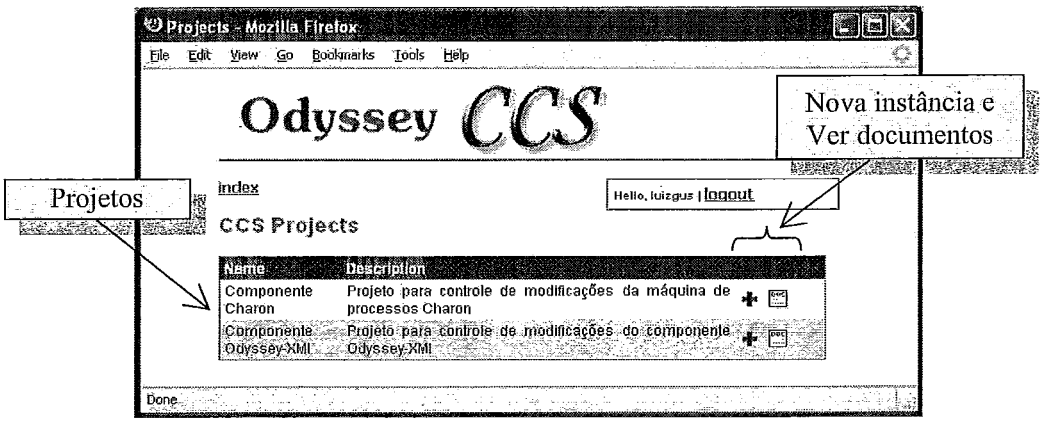


Fig. 4-21 - Projetos configurados no Odyssey-CCS

A primeira tabela exibe as atividades pendentes, e a segunda as decisões que devem ser tomadas. Em cada linha é exibido o número da instância do processo, que corresponde à modificação sendo controlada, o nome da atividade ou decisão, o nome do projeto ao qual a modificação está associada e, no caso de decisão, suas opções.

No caso específico da Fig. 4-22, são exibidas duas instâncias de processo em execução, uma referente ao projeto *Componente Charon* e a outra referente ao projeto *Componente Odyssey-XMI*, com tarefas pendentes para o usuário identificado por *luizgus*, sendo que uma está aguardando a finalização da atividade *Requisitar Modificação*, e a outra está aguardando que a decisão *Decisão da Avaliação?* seja tomada. O processo utilizado pelos dois projetos é aquele descrito no exemplo do Capítulo 3. No caso da decisão, deve-se notar que as opções apresentadas são obtidas do processo, especificamente dos fluxos que saem da decisão modelada, como exibido na Fig. 4-23.

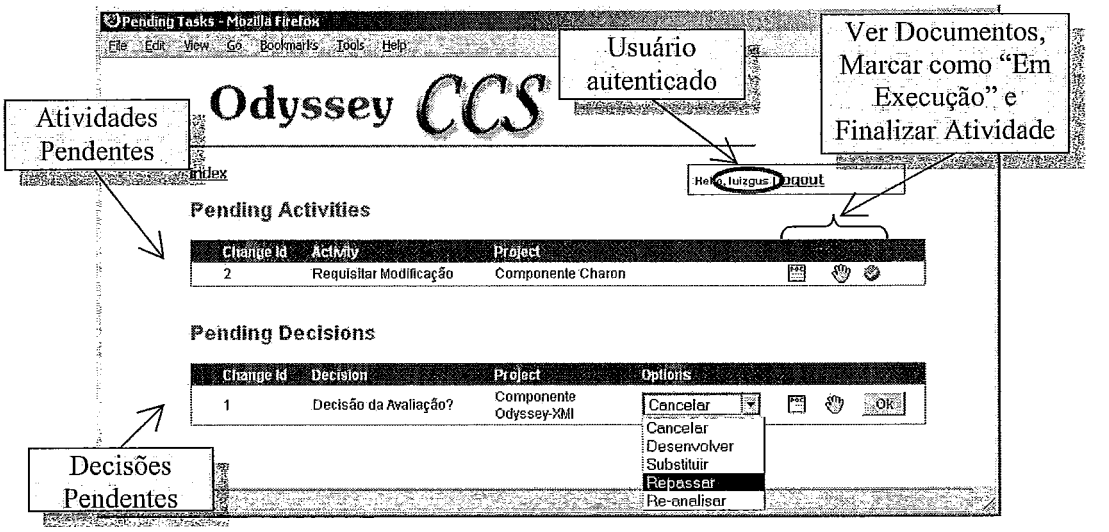


Fig. 4-22 - Tela de exibição das atividades e decisões pendentes para determinado usuário

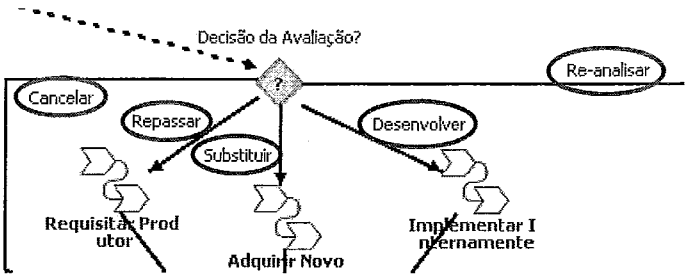


Fig. 4-23 - Detalhe do processo exemplo definido no Capítulo 3, com ênfase no elemento de decisão

Para cada instância de processo em execução, há três ações disponíveis. A primeira é visualizar os documentos associados à instância do processo, pois pode ser necessário analisar as informações já coletadas para a execução da tarefa que está pendente. Por exemplo, para executar a atividade de Análise, é necessário analisar as informações coletadas no formulário Requisição de Modificação.

A segunda ação disponível é marcar uma tarefa, seja atividade ou decisão, como em execução pelo usuário corrente. Ela deve ser acionada quando o usuário vai trabalhar na tarefa e quer que os outros usuários, que também tenham assumido um dos papéis atribuídos à tarefa, tenham conhecimento disso. A Fig. 4-24 exibe a atividade pendente *Classificar* como em execução pelos usuários identificados por *luizgus* e *cfyf*. É exibido um ícone no início da linha indicando que a atividade está em execução, e o nome do usuário que a executa é exibido quando o mouse é colocado sobre o ícone. Dessa forma, quando um usuário se autentica no sistema e lista suas tarefas pendentes, ele fica ciente de quais tarefas já estão sendo executadas e por quem. É importante ressaltar que mais de um usuário pode marcar uma mesma atividade como em execução. O mesmo acontece para as decisões.

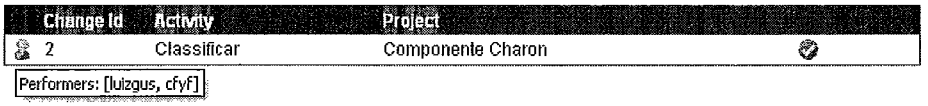


Fig. 4-24 - Atividade marcada como "em execução"

A terceira ação disponível para cada instância de processo em execução é finalizar a tarefa. Quando esta ação é selecionada, o módulo Gerenciador de Execução finaliza a tarefa, armazenando quem solicitou a finalização e o tempo consumido em sua execução. Se a tarefa for uma atividade que gera um produto, e existir um formulário associado a este produto, o formulário é exibido para que o usuário forneça as informações requeridas.

Por exemplo, o formulário modelado na Fig. 4-11 foi associado ao produto *Requisição de Modificação* durante a configuração do projeto, como exibido na Fig.

4-17. Portanto, ao finalizar a atividade pendente *Requisitar Modificação*, exibida na Fig. 4-22, será apresentado o formulário associado ao seu produto, exposto na Fig. 4-25.

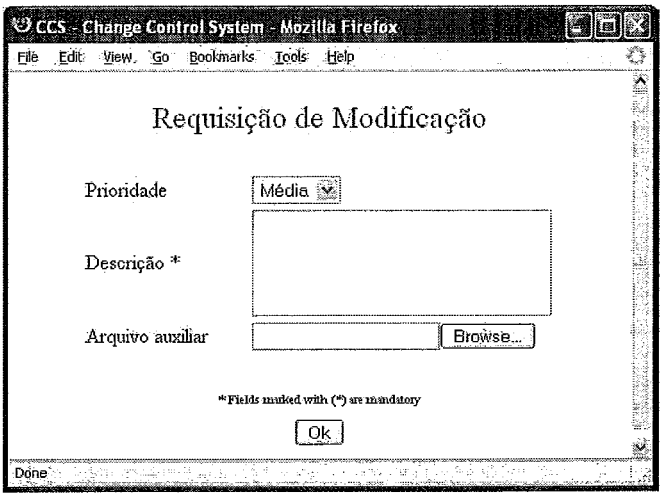


Fig. 4-25 - Formulário *Requisição de Modificação*, associado ao produto homônimo

O formulário é preenchido pelo usuário que finalizou a atividade e suas informações são armazenadas, agregadas a um documento, para posterior consulta, que pode ser realizada a partir da tela que lista os projetos existentes (Fig. 4-21). Na Fig. 4-26 é exibida a tela de consulta de informações do projeto Componente Charon. Há uma tabela onde cada linha se refere a uma instância de processo em execução ou já finalizada. Na primeira coluna, é exibido o número da instância, e na segunda os produtos gerados durante a sua execução. Ao selecionar um produto, o formulário associado é exibido com as informações armazenadas. A Fig. 4-27 exibe como exemplo as informações coletadas para o produto Requisição de Modificação em uma determinada execução de processo. Pode-se notar que o seu formato é similar ao do formulário de coleta de informações exibido na Fig. 4-25.

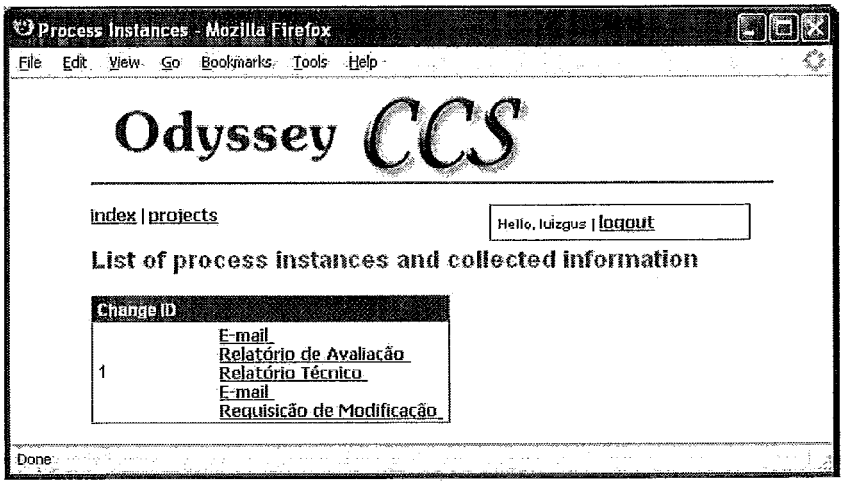


Fig. 4-26 - Tela com a lista de informações coletadas

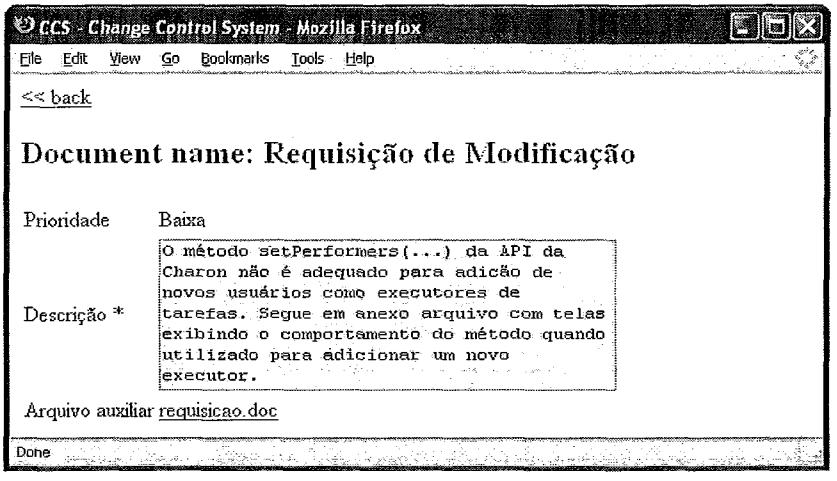


Fig. 4-27 - Documento com as informações coletadas para o produto Requisição de Modificação

4.7 - Mapa de reutilização

O Mapa de Reutilização, como descrito no Capítulo 3, mantém as informações sobre a reutilização de componentes, como seu produtor, seus consumidores, e as licenças firmadas entre os produtores e consumidores, para cada componente. Ele foi desenvolvido integrado a uma biblioteca de componentes já existente, utilizando a sua infra-estrutura e estendendo o seu modelo de dados, que já contemplava algumas informações sobre os componentes e seus produtores.

A biblioteca de componentes é utilizada por produtores e consumidores de componentes. Os produtores publicam seus componentes para serem reutilizados, e os consumidores fazem buscas e *downloads* dos componentes selecionados. Para isso, ambos devem estar autenticados no sistema.

Ao publicar um componente, o produtor insere um conjunto de informações, como o nome, a descrição, se o componente é disponibilizado como serviço para Internet (*WebServices*), o arquivo que contém os artefatos do componente, as categorias das quais faz parte e as licenças associadas a ele. As categorias e as licenças são previamente cadastradas no sistema. Cada categoria está associada a um determinado conjunto de informações próprio, que será requisitado ao produtor no decorrer da publicação, caso a categoria seja selecionada.

Cada componente também pode estar associado a mais de um tipo de licença. Por exemplo, um mesmo componente pode ser utilizado sob uma licença de software de código aberto (*open source*), como GPL ou LGPL (FSF, 2006), ou sob uma licença comercial, dependendo do consumidor e do uso destinado ao componente. Por exemplo, se o componente for utilizado para a construção de um software de código aberto, o

primeiro tipo de licença deve ser selecionado. Se ele for utilizado para a construção de um software comercial, o segundo tipo de licença é o apropriado. Exemplos de software que possuem mais de um tipo de licenças disponíveis são o JIRA (ATLASSIAN, 2006a), MySQL (MYSQL AB, 2006), Perforce (WINGERD, 2005), Berkeley DB (SLEEPYCAT, 2006) e Qt (TROLLTECH, 2006).

A Fig. 4-28-(i) exibe o esquema de publicação de um componente na biblioteca. Neste caso, além de inserir as informações sobre o componente Charon, o produtor o associou a três licenças, que poderão ser selecionadas para uso pelos consumidores.

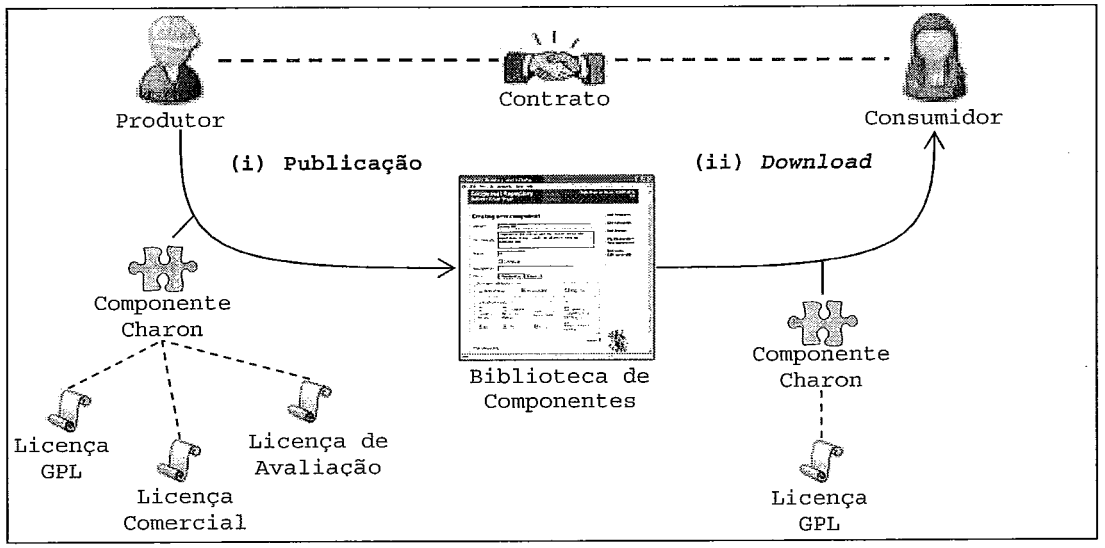


Fig. 4-28 - Publicação e *download* na biblioteca de componentes

É importante que nas licenças esteja claro de quem é a responsabilidade de manutenção de software. Por exemplo, as licenças de avaliação e GPL deixam claro que o produtor não se responsabiliza pela manutenção do software, nem por eventuais falhas ocorridas durante o seu uso. Já algumas licenças comerciais garantem a manutenção corretiva do software por um período de tempo limitado, bem como a sua atualização, caso haja novas liberações do produto neste período.

Ao fazer o *download* de um componente da biblioteca, o consumidor deve selecionar qual das licenças associadas ao componente deve vigorar. Neste momento, são registrados o consumidor do componente, a sua data de aquisição e a licença selecionada, dando início a um contrato entre o produtor e o consumidor para a utilização do referido componente. A Fig. 4-28-(ii) exemplifica esta operação, onde o consumidor faz o *download* do componente Charon selecionando a licença GPL. Neste momento, firma-se um contrato entre o consumidor e o produtor, indicando que o consumidor pode utilizar o componente Charon sob as regras definidas na licença GPL.

À medida que os consumidores fazem novos *downloads* na biblioteca de componentes, novos contratos são concretizados e as informações relativas ao consumo dos componentes são registradas e disponibilizadas para as pessoas autorizadas.

A Fig. 4-29 exibe um exemplo de acesso às informações do Mapa de Reutilização a partir de um formulário modelado no Odyssey-CCS. Neste caso, o componente Charon, o mesmo referenciado na Seção 4.6.3, é alvo de uma RM, e uma instância do processo de controle de modificações está sendo executada pela equipe produtora do componente.

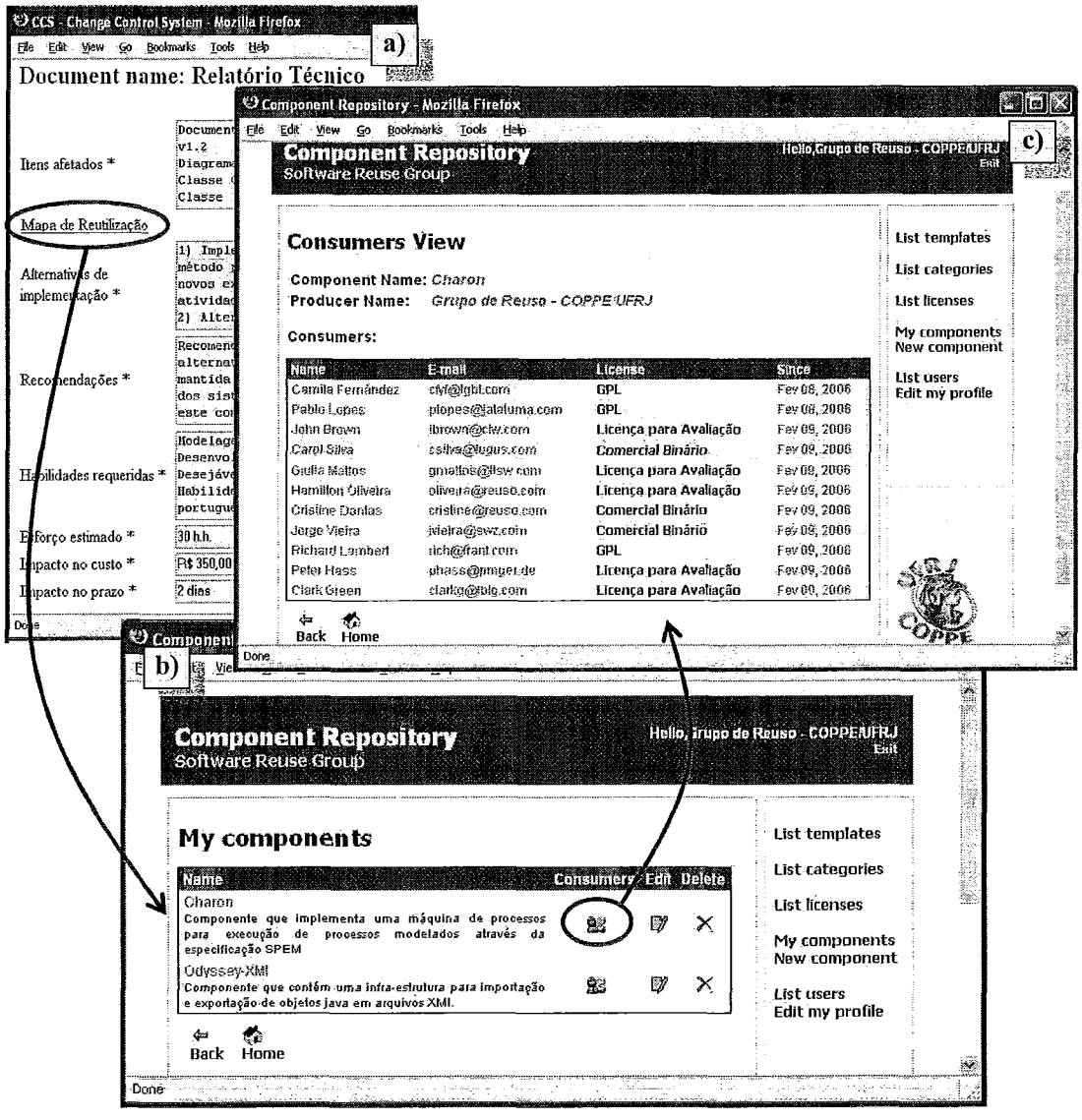


Fig. 4-29 - Exemplo de acesso às informações do Mapa de Reutilização

Após a análise da RM, foi gerado um formulário chamado Relatório Técnico (Fig. 4-29-a), que contém informações necessárias para que o CCC possa tomar decisões sobre as próximas atividades a serem executadas (e.g. cancelar, aprovar,

postergar). Um dos campos do formulário é uma referência (*link*) para o Mapa de Reutilização. Esta referência leva à tela dos componentes produzidos por esta equipe (Fig. 4-29-b), sendo que o componente Charon é um deles. Nesta tela, existe uma referência para os consumidores de cada um destes componentes, registrados a partir dos *downloads* realizados na biblioteca de componentes.

A Fig. 4-29-c exibe algumas informações sobre os consumidores do componente *Charon*, como o nome, o endereço eletrônico, a licença sob a qual utiliza o componente e a data de início da utilização. Dessa forma, a equipe produtora sabe quais são os consumidores que serão afetados pela modificação requisitada.

Supondo que a RM seja do tipo corretiva e foi enviada pelo consumidor *Camila Fernández*, cuja licença utilizada é a GPL, que não lhe garante que a RM será atendida, o CCC pode decidir postergar a RM. Porém, através do Mapa de Reutilização, o CCC percebe que a RM atende a vários consumidores, sendo que alguns deles possuem licenças comerciais, que lhes garante a correção. O CCC pode então decidir antecipar a correção, para que o problema não venha a afetar estes usuários. Caso haja outras RMs com nível de severidade similar, mas que afetem um número menor de consumidores, esta nova RM pode ter sua prioridade elevada em relação às outras.

Há ainda uma outra questão, com relação à importância dos consumidores para a equipe produtora. Este nível de importância pode estar relacionado à quantidade de componentes adquiridos por cada consumidor ou a questões estratégicas da equipe produtora, por exemplo. Sabendo que RMs similares estão em andamento, mas que algumas afetam os consumidores considerados mais importantes, o CCC pode aumentar ou diminuir a prioridade das RMs.

Supondo agora que a RM é do tipo evolutiva, o CCC pode, além de utilizar as informações do relatório técnico produzido na análise, sondar o interesse dos demais consumidores na nova funcionalidade proposta pela RM.

Após a realização da modificação, a equipe produtora deve informar aos consumidores que têm direito às modificações, identificados através das licenças estabelecidas, que uma nova versão do componente está disponível. Além disso, ele pode anunciar aos demais consumidores interessados que uma nova versão do componente foi produzida.

4.8 - Casos de utilização do protótipo

O protótipo Odyssey-CCS possui dois casos de utilização por outros trabalhos. No primeiro deles, foi feita a integração com um sistema de controle de versões, o Odyssey-VCS (OLIVEIRA *et al.*, 2005), possibilitando o agrupamento lógico de artefatos modificados. O Odyssey-CCS fornece uma lista de modificações lógicas em aberto, e os artefatos modificados e armazenados pelo sistema de controle de versões são associados a uma destas modificações lógicas, possibilitando a criação de pacotes de modificações.

No segundo caso, o Odyssey-CCS foi utilizado pelo Odyssey-WI (DANTAS *et al.*, 2005) para auxiliar a detecção de rastros de modificação entre artefatos, através da mineração das informações de suas modificações lógicas, e também como fonte de informações para a fundamentação dos rastros encontrados.

Estes casos de utilização são detalhados a seguir.

4.8.1 - Integração com o sistema de controle de versões Odyssey-VCS

O Odyssey-CCS facilita a coleta de informações sobre as modificações realizadas nos projetos, apresentando formulários de coleta de informações aos usuários ao longo da execução dos processos de controle de modificações, como já discutido anteriormente. Algumas destas informações são coletadas automaticamente pelo protótipo, como a duração e os participantes de uma determinada atividade, por exemplo.

Uma informação importante a ser coletada é a lista de ICs alterados na implementação de cada RM. Para isso, o Gerente de GCS pode criar no formulário que representa o relatório de implementação um campo específico para que os usuários especifiquem os ICs que foram alterados. Porém, a aquisição desta informação pode ser simplificada através da integração do Odyssey-CCS com um sistema de controle de versões.

O Odyssey-CCS está integrado ao Odyssey-VCS (*Version Control System*) (OLIVEIRA *et al.*, 2005), que propõe uma abordagem para o controle de versões de modelos baseados no MOF, como a UML. Ele permite que os elementos internos dos modelos sejam versionados e controlados, ao contrário dos sistemas de controle de versões baseados em arquivos, como o CVS e ClearCase, que versionam o arquivo que contém o modelo, não possibilitando controlar a evolução dos seus elementos internos.

O Odyssey-VCS possui um módulo servidor, que mantém um repositório com os ICs sendo controlados, e um módulo cliente, que mantém um espaço de trabalho individual para o usuário. No seu espaço de trabalho, o usuário pode recuperar uma cópia dos ICs presentes no servidor, através da operação de *check-out*. Após trabalhar nestes ICs, ele devolve ao servidor os artefatos alterados, através da operação de *check-in*.

Os ICs, quando fazem parte de uma linha básica, só podem ser modificados através de procedimentos formais definidos para o projeto do qual fazem parte. Assim, os ICs são alterados dentro de um contexto de execução de um processo de controle de modificações.

O Odyssey-VCS se comunica com o Odyssey-CCS e obtém informações sobre as instâncias de processos de controle de modificações em execução, que representam as modificações lógicas sendo realizadas. Quando um usuário se autentica no módulo cliente do Odyssey-VCS, este exibe a lista de modificações lógicas pendentes do Odyssey-CCS. O usuário seleciona uma delas, a que representa o trabalho que será realizado, e depois trabalha no seu ambiente, executando operações de *check-out*, modificando os ICs necessários e executando operações de *check-in*. Todos os ICs alterados e devolvidos ao módulo servidor do Odyssey-VCS são associados àquela modificação lógica selecionada, formando-se um pacote de modificações (WEBER, 1997).

Mantém-se assim o rastro entre os ICs modificados e o contexto que originou esta alteração, de forma mais automática e menos sujeita a erros do que a descrita anteriormente, onde o usuário deveria informar todos os ICs modificados em cada modificação lógica. Estas informações são úteis para a verificação e auditoria das modificações, comparação entre modificações lógicas e propagação das modificações realizadas em um software.

4.8.2 - Utilização do Odyssey-CCS pela abordagem Odyssey-WI

Ao analisar uma RM, os analistas de sistemas devem estimar qual será o impacto da modificação requisitada. O resultado desta análise engloba a identificação dos artefatos que sofrerão modificação e as estimativas de custo e prazo para a realização da modificação, entre outros.

Uma informação que auxiliaria a detecção dos artefatos afetados é a correlação das alterações dos artefatos realizadas no conjunto de modificações lógicas registradas

para um determinado software. Essa informação é obtida através da análise dos rastros de modificação entre os artefatos.

DANTAS *et al.* (2005) propõem uma abordagem para detecção destes rastros através do uso de técnicas de mineração de dados. Os repositórios dos sistemas de controle de versões e dos sistemas de controle de modificações são ricos em informações sobre as modificações, e os rastros entre os artefatos estão presentes neles de forma implícita. Através do uso de técnicas de mineração sobre estes repositórios, a abordagem de Dantas consegue explicitar este conhecimento. Foi implementado um protótipo com base nesta abordagem, chamado Odyssey-WI (*Workspace Integration*), que minera os repositórios do Odyssey-CCS e do Odyssey-VCS e apresenta os rastros de modificações encontrados. Este protótipo possui um módulo servidor e um módulo cliente, este último implementado como uma aplicação independente e também como um *plugin* do ambiente Odyssey (2006).

O Odyssey-WI permite responder à seguinte questão: "Quando um determinado artefato é modificado, quais outros artefatos são modificados em conjunto?". Além de auxiliar a análise do impacto da modificação, estas informações auxiliam a detectar erros de projeto, pois elas evidenciam situações de alto acoplamento, e também podem ajudar a evitar erros de modificações incompletas.

Ao encontrar os rastros de modificações, o Odyssey-WI exibe uma série de informações associadas a eles, que auxiliam a sua compreensão. Estas informações são: **quem** implementou a modificação, **quando** ela foi implementada, **onde** foi necessário alterar para implementá-la, **como** ela foi implementada, **o que** foi feito e **por que** ela foi necessária. Parte destas informações é obtida no Odyssey-VCS e parte no Odyssey-CCS.

O campo de formulário utilizado para coletar cada uma destas informações depende do projeto, já que o processo e os formulários podem variar conforme a sua necessidade. Como os formulários de coleta de informações do Odyssey-CCS são configuráveis, sempre será possível criar os campos necessários para coleta das informações necessárias ao mecanismo de rastreabilidade do Odyssey-WI. Este mecanismo também é configurável, sendo possível fazer a associação de campos dos formulários com a semântica do rastro, para cada projeto. O Odyssey-WI faz uso da API de consulta às informações do Odyssey-CCS, possibilitando acessar quais são os formulários e campos existentes, bem como as informações coletadas através deles.

A Fig. 4-30 apresenta a tela de configuração do Odyssey-WI, onde é realizada a associação entre os campos dos formulários do Odyssey-CCS, ou seja, a fonte de informações, e a semântica dos rastros. Por exemplo, as informações sobre quem implementou a modificação (campo *who*) serão obtidas através do campo *Responsável pela Implementação* do formulário *Relatório de Implementação e Verificação*. Esta informação também pode ser obtida no repositório do Odyssey-VCS.

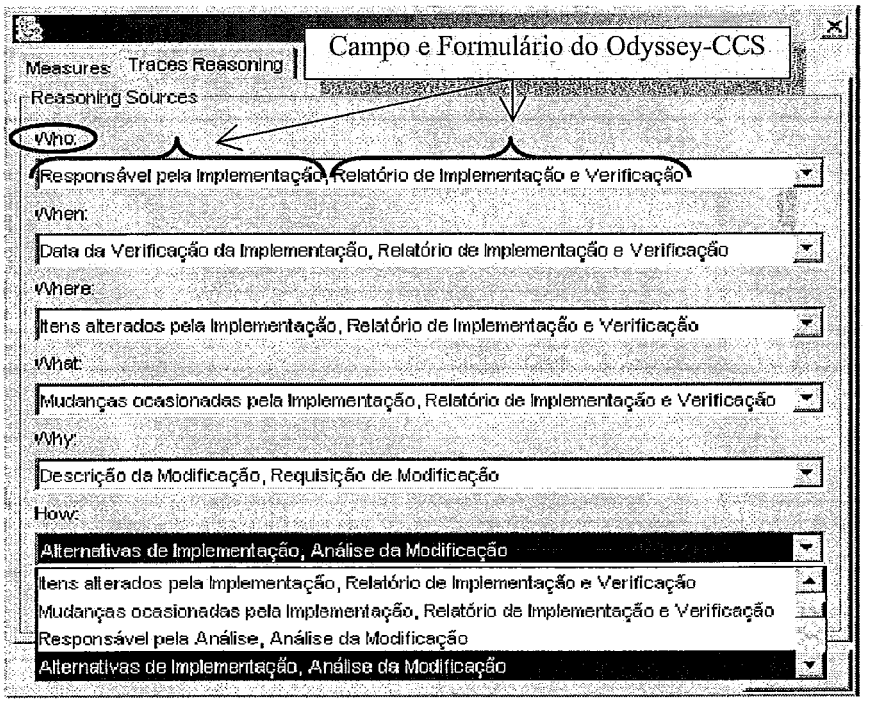


Fig. 4-30 - Configuração da fonte de informações do Odyssey-WI (DANTAS, 2005).

DANTAS (2005) apresenta um exemplo de utilização do Odyssey-WI. Nele, um desenvolvedor deve realizar uma atividade de implementação de uma modificação. Ele utiliza o ambiente Odyssey para modelagem, e trouxe para seu ambiente de trabalho local os artefatos do software que serão modificados, através de operações de *check-out* realizadas utilizando-se o cliente do Odyssey-VCS.

Uma das classes que ele deve alterar é a *Reserva*. Ao acionar a funcionalidade de mineração de rastros de modificação sobre esta classe, é apresentada uma tela com informações sobre os rastros encontrados, como mostra a Fig. 4-31. Na parte de cima desta figura são exibidos os rastros identificados. Verifica-se, por exemplo, que existe um rastro de modificação entre a classe *Reserva* e a classe *Hotel*, sendo que sua relevância é dada em termos da medida de suporte, calculada em 67%, e de confiança, calculada em 80%. Isso significa que as duas classes foram alteradas em conjunto em 67% do total de modificações realizadas, e em 80% das vezes que a classe *Reserva* foi

modificada, a classe *Hotel* também sofreu alterações. Com base nestas informações, o desenvolvedor deve ficar atento, pois uma modificação na classe *Reserva* pode gerar a necessidade de modificação da classe *Hotel*, já que estas classes foram alteradas em conjunto com certa freqüência em situações anteriores.

Na parte de baixo da figura, são exibidas as informações relativas ao rastro selecionado na parte superior. Estas informações são importantes para que o desenvolvedor compreenda o rastro encontrado, já que elas fornecem mais detalhes a respeito das modificações realizadas. Estas informações são obtidas do Odyssey-VCS e do Odyssey-CCS, de acordo com a configuração realizada, exemplificada pela Fig. 4-30. O Odyssey-WI especifica a fonte das informações apresentadas, como indicado na Fig. 4-31.

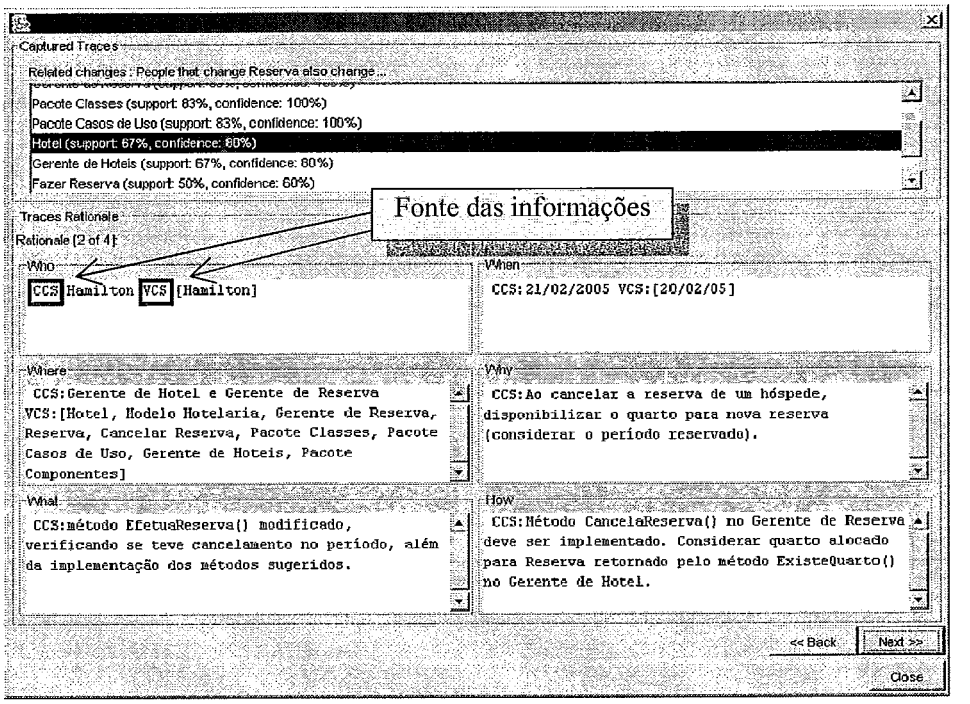


Fig. 4-31 - Visualização dos rastros de modificação da classe Reserva (DANTAS, 2005).

Este trabalho de pesquisa de mestrado realizado por Dantas é um exemplo de utilização do protótipo Odyssey-CCS, mais especificamente do seu repositório e da sua API de acesso às suas informações. Também é um exemplo da utilidade das informações coletadas por ele, utilizadas em auxílio à detecção e apresentação de informações de rastros de modificações definidos pela abordagem Odyssey-WI.

4.9 - Considerações finais

Neste capítulo, foi apresentado o protótipo Odyssey-CCS, implementado com o intuito de viabilizar a utilização da abordagem definida no Capítulo 3. A arquitetura do protótipo foi detalhada, tendo sido descritos todos os seus módulos implementados e também os reutilizados. Foram apresentados os principais aspectos da implementação, bem como o funcionamento do protótipo, através da explicação e exibição de suas telas principais.

Foi discutido com detalhes como o protótipo auxilia a modelagem de processos de controle de modificações, como são modelados os formulários para a coleta de informações, como são criados os projetos específicos aos sistemas de software que devem ter suas modificações controladas, como os processos são configurados para poderem ser utilizados, com relação à atribuição de responsabilidades e coleta de informações, como os processos são instanciados e executados, como são configuradas e enviadas as notificações durante a execução dos processos, como as informações coletadas são visualizadas, entre outros assuntos.

Além disso, foi detalhado como as informações relacionadas à reutilização de componentes, mantidas pelo Mapa de Reutilização, são disponibilizadas durante a execução dos processos de controle de modificações. Espera-se que esta característica auxilie a resolução do problema da cadeia de responsabilidades de manutenção.

Como ilustra a Tabela 3, o Odyssey-CCS atende a todos os critérios descritos no Capítulo 2. Apesar desta lista não ser completa, é possível visualizar que ele atende a critérios não atendidos por outras abordagens, o que pode ser considerado uma contribuição à área.

Todavia, apesar do esforço aplicado na implementação deste protótipo, novas funcionalidades ainda podem ser agregadas a ele. Uma delas refere-se à análise das informações coletadas. Atualmente, estas informações podem ser visualizadas, porém, a utilização de gráficos e relatórios com a sumarização das informações coletadas em um conjunto de execuções de processo seriam úteis aos Gerentes de Projeto, por exemplo, para se saber o número de modificações em aberto e finalizadas em determinado período de tempo, quantas solicitações de modificação, sejam corretivas ou evolutivas, existem para cada componente reutilizado, etc.

Apesar do protótipo ter sido desenvolvido com foco no contexto do DBC, ele pode também ser utilizado no contexto convencional de controle de modificações em

software. No contexto do DBC, ele pode ser utilizado tanto pelas equipes produtoras quanto pelas consumidoras de componentes. Este protótipo fornece o apoio necessário à utilização da abordagem proposta, podendo ser utilizado para auxiliar o trabalho de controle de modificações em software no contexto do DBC.

Tabela 3 – Comparação das abordagens segundo os critérios definidos no Capítulo 2

Critérios	Bugzilla, GConf, phpBugTracker, Scarab, Roundup	Mantis	ClearQuest, JIRA, Razor	StarTeam, TeamTrack	MwR-TERRA	KobrA	Select Perspective	Odyssey-CCS
Modelagem gráfica do processo de controle de modificações	-	-	+	+	-	-	-	+
Configuração da coleta de informações	-	+	+	+	-	-	-	+
Linguagem de processos padronizada	-	-	-	-	-	-	-	+
Processo de controle de modificações configurado por projeto	-	-	+	+	-	-	-	+
Envio de notificações	+	+	+	+	+	-	+	+
Pacotes de modificações	+	+	+	+	-	+	-	+
Software implementado	+	+	+	+	+	-	-	+
Responsabilidade de manutenção	-	-	-	-	-	-	+	+
Identificação dos consumidores	-	-	-	-	-	-	-	+

Capítulo 5 - Conclusão

5.1 - Visão Geral

As modificações são inevitáveis nos sistemas de software, e estas podem ocorrer em qualquer fase de seu ciclo de vida (LEON, 2000; PRESSMAN, 2005). Porém, quando as modificações são realizadas sem controle, muitos problemas podem ocorrer, como a perda de informações, o re-trabalho e atrasos na entrega do produto modificado.

Os sistemas de controle de modificações dão apoio à condução de parte das funções da GCS, que é a disciplina responsável pelo controle da evolução do software. Estes sistemas fornecem recursos que auxiliam a coordenação do trabalho e a manutenção e disseminação de informações.

Entretanto, software desenvolvido no contexto do DBC precisa de sistemas de controle de modificações com alguns recursos que não são fornecidos pelas abordagens atuais, conforme análise realizada no Capítulo 2. Estes recursos referem-se tanto à configuração do processo de controle de modificações quanto à manutenção de informações sobre a reutilização de componentes, utilizadas ao longo do processo de controle de modificações para a tomada de decisões e para a condução do trabalho.

Entre estas informações estão a identificação dos consumidores de cada componente e seus dados de contato, o produtor do componente e o contrato firmado entre as partes, para cada componente. Este conjunto de informações constitui o Mapa de Reutilização, apresentado na Seção 3.4.1, que auxilia a resolução do problema da cadeia de responsabilidades de manutenção, introduzido no Capítulo 1.

Dessa forma, foram desenvolvidos uma proposta e um sistema para o controle de modificações mais adequado ao contexto do DBC, ambos chamados Odyssey-CCS, detalhados nos Capítulos 3 e 4, respectivamente.

5.2 - Contribuições

A abordagem definida e o sistema de controle de modificações para o contexto do DBC construído como parte deste trabalho possuem as seguintes características:

- Modelagem gráfica do processo de controle de modificações por meio de recursos visuais. É fornecido um painel de modelagem que possibilita criar elementos, movê-los com o auxílio de recursos de arrastar e soltar, associá-los

através de elementos de ligação, e removê-los. Esta característica simplifica o trabalho de modelagem realizado pelo gerente de configuração, que dispõe de recursos visuais para a definição do processo de controle de modificações a ser utilizado.

- Utilização de uma linguagem de processos padrão, a SPEM, para a definição dos processos de controle de modificações, e exportação dos processos modelados também segundo esta linguagem padronizada. Esta característica permite que outros sistemas possam reutilizar os processos modelados, desde que sejam compatíveis com a SPEM. Por ser uma linguagem padronizada, documentada e pública, a abrangência de usuários que a conhecem também tende a ser maior.
- Cada projeto pode utilizar um processo de controle de modificações diferente, adequado às suas características, de acordo com o definido no Plano de Gerência de Configuração de Software. Estes processos modelados segundo a SPEM também podem ser reutilizados por qualquer projeto.
- A coleta de informações é configurável. É possível definir quais informações devem ser coletadas em cada atividade do processo de controle de modificações. A definição das informações a serem coletadas depende da política de GCS adotada pela organização. Elas devem estar explicitadas no Plano de Gerência de Configuração de Software.
- Envio de notificação configurável. Conforme definido pela função de Acompanhamento da GCS, as informações coletadas durante a realização das modificações devem ser disseminadas para as pessoas interessadas e autorizadas. É possível configurar qual mensagem será enviada, para quais pessoas, e em que momento do processo de controle de modificações.
- Rastreamento entre uma modificação lógica, representada por uma instância de execução de um processo de controle de modificações, e os artefatos alterados, através da integração com o Odyssey-VCS. Desta forma, são criados os pacotes de modificações.
- Integração com um Mapa de Reutilização. Este elemento, implementado sobre uma biblioteca de componentes existente, possui um conjunto de informações necessárias para auxiliar a identificação dos responsáveis pelas modificações nos componentes. Ele auxilia a solução do problema da cadeia de responsabilidades de manutenção, descrito na Seção 1.2. As informações

sobre os consumidores dos componentes também são necessárias para a realização da análise e avaliação das RMs nos processos de controle de modificações utilizados no contexto do DBC.

- Implementação do protótipo com tecnologia para Internet, facilitando a sua utilização por pessoas geograficamente distribuídas, fato comum em muitas empresas, e, principalmente, na comunidade de software de código aberto.

O Odyssey-CCS foi utilizado pela abordagem Odyssey-VCS, definida no trabalho de OLIVEIRA (2005) e explicada na Seção 4.8.1, e pela abordagem Odyssey-WI, fruto do trabalho de DANTAS (2005), conforme descrito na Seção 4.8.2. O fato de o Odyssey-CCS possuir uma API para consulta sobre os formulários e campos modelados e sobre as informações coletadas facilitou a sua utilização.

O principal objetivo deste trabalho, que consiste na definição de uma abordagem para o controle de modificações adequada ao DBC, foi alcançado através da elaboração da proposta e da implementação do Odyssey-CCS. Ele permite a modelagem de processos de controle de modificações adaptados para o contexto do DBC, permite a configuração da coleta de informações, e provê um mecanismo que auxilia a resolução do problema da cadeia de responsabilidades de manutenção, disponibilizando as informações do Mapa de Reutilização.

5.3 - Limitações e trabalhos futuros

Neste trabalho, foram detectadas algumas limitações e foram identificados possíveis trabalhos futuros que podem dar continuidade à pesquisa realizada, descritos a seguir:

5.3.1 - Limitações

- **Busca de informações:**

O Odyssey-CCS permite a visualização das informações coletadas, mas elas são acessadas somente a partir da instância do processo de controle de modificações e atividade na qual foi coletada. Todas as instâncias registradas são exibidas em uma lista, o que dificulta encontrar a informação desejada quando várias instâncias de processo já foram iniciadas.

Existe a carência de um mecanismo de busca das informações mantidas pelo sistema. Por exemplo, poderia ser implementado ou reutilizado um mecanismo de

busca por texto livre (BAEZA-YATES, RIBEIRO-NETO, 1999), em conjunto com os próprios campos definidos nos formulários do sistema. Assim, seria possível buscar, por exemplo, todas as instâncias de processo onde o executor da atividade Análise foi o usuário Camila, ou todas as instâncias iniciadas em determinado período de tempo, ou todas as modificações lógicas que tenham impactado um determinado artefato ou que tenham sido iniciadas por um determinado consumidor.

- **Melhoria da integração com o Odyssey-VCS:**

Atualmente, o Odyssey-CCS está integrado ao Odyssey-VCS, como detalhado na Seção 4.8.1. Antes de iniciar o trabalho de modificação nos artefatos, são apresentados para ele os identificadores das modificações lógicas em aberto no Odyssey-CCS, e o desenvolvedor deve escolher aquela na qual ele está trabalhando.

Contudo, não é verificado neste momento se o desenvolvedor possui permissão para realizar a modificação, mas a responsabilidade sobre as atividades está registrada no Odyssey-CCS, através da associação de papéis e usuários às atividades. Além disso, poderia ser exibido ao desenvolvedor, além do identificador da modificação lógica, a sua descrição e o nome da atividade pendente.

- **Elementos SPEM não contemplados:**

O módulo de auxílio à modelagem de processos de controle de modificações não contempla todos os elementos SPEM. Exemplos de elementos não considerados são o Ciclo de Vida, Fase e Disciplina. Ao incorporar todos os elementos SPEM, este módulo poderia ser utilizado para a modelagem de outros tipos de processos, como processos de desenvolvimento de software, por exemplo.

5.3.2 - Trabalhos futuros

- **Análise das informações coletadas:**

Como os demais sistemas de controle de modificações, o Odyssey-CCS mantém uma rica fonte de informações relacionadas à evolução do software, muito úteis para a gerência dos projetos (CRNKOVIC, WILLFÖR, 1998).

Podem ser utilizados gráficos e relatórios que sumarizem e apresentem estas informações. Exemplos de análises são o número de modificações lógicas em aberto, o número de modificações lógicas realizadas em determinado período de tempo, o tempo utilizado para a execução das modificações, o número de modificações que afetaram um determinado componente, entre outros.

Outro exemplo de manipulação destas informações é o realizado pela abordagem Odyssey-WI.

- **Apoio à análise de impacto:**

No DBC, as aplicações são construídas reutilizando-se componentes. Até mesmo os componentes podem reutilizar outros componentes. Na atividade de análise de impacto, que é parte do processo de controle de modificações, um dos objetivos é identificar quais são os artefatos afetados por uma RM, o que inclui os componentes reutilizados. Existem na literatura alguns trabalhos que tratam desta questão (LARSSON, 2000; LEBSACK *et al.*, 2001; KNETHEN, GRUND, 2003). Entretanto, o Odyssey-CCS não possui recursos que auxiliem a condução desta atividade.

Uma forma de prover indícios sobre os componentes afetados por uma RM é através do mapeamento dos requisitos do software com os componentes utilizados para implementá-los. Assim, ao identificar o requisito do software afetado pela RM, são identificados também os possíveis componentes que deverão ser modificados.

- **Melhoria do monitoramento dos processos de controle de modificações:**

O protótipo ainda pode ser melhorado no que diz respeito ao monitoramento dos processos de controle de modificações. Atualmente, é possível que o gerente do projeto obtenha informações sobre o estado atual de cada instância de processo de controle de modificações, através da visualização das atividades pendentes, desde que ele exerça também os papéis associados às atividades.

Entretanto, seria interessante a visualização gráfica do estado de cada instância de processo em execução, desenhado segundo a mesma notação SPEM utilizada no modelador de processos. Assim, para cada instância, o gerente poderia visualizar graficamente quais atividades já foram executadas e por quem, e qual a atividade está em execução.

- **Verificação das conclusões obtidas:**

O Odyssey-CCS foi utilizado por duas abordagens, a Odyssey-VCS e a Odyssey-WI, como descrito na Seção 4.8. Porém, existe ainda a necessidade de realização de estudos experimentais para a verificação prática das reais contribuições do uso da abordagem proposta e das suas deficiências. O protótipo implementado facilitará a realização destes estudos.

É importante que a abordagem seja verificada em projetos reais que apliquem o DBC, onde os problemas discutidos nesta dissertação e os profissionais responsáveis pelas modificações são encontrados.

Referências Bibliográficas

ALUR, D., CRUPI, J., MALKS, D., 2001, *Core J2EE Patterns: Best Practices and Design Strategies*, 1st ed., Palo Alto, California, Pearson Education.

ANDREWS, T., CURBERA, F., DHOLAKIA, H., *et al.*, 2003, *Business Process Execution Language for Web Services Version 1.1.*

APPERLY, H., HOFMAN, R., LATCHEM, S., *et al.*, 2003, *Service- and Component-based Development: Using Select Perspective™ and UML*, 1st ed., UK, Addison Wesley.

ASKLUND, U., BENDIX, L., 2002, "A Study of Configuration Management in Open Source Software", *IEE Proceedings - Software*, v. 149, n. 1 (February), pp. 40-46.

ATKINSON, C., BAYER, J., BUNSE, C., *et al.*, 2001, *Component-Based Product Line Engineering with UML*, 1st ed., Addison-Wesley.

ATKINSON, C., BAYER, J., LAITENBERGER, O., *et al.*, 2000, "Component-Based Software Engineering: The KobrA Approach". In: *3rd International Workshop on Component-based Software Engineering*, pp. 289-310, Limerick, Ireland, June.

ATLASSIAN, 2006a, "JIRA - Bug tracking, issue tracking and project management software". In: http://www.atlassian.com/software/jira/docs/latest/, accessed in 10/02/2006.

ATLASSIAN, 2006b, "JIRA Customers". In: http://www.atlassian.com/software/jira/customers.jsp, accessed in 19/02/2006.

BAEZA-YATES, R., RIBEIRO-NETO, B., 1999, *Modern Information Retrieval*, 1st ed., ACM Press.

BARNSON, M.P., WEISSMAN, T., HERNANDEZ, T., *et al.*, 2005, *The Bugzilla Guide - 2.20 Development Release*, The Bugzilla Team.

BORLAND, 2005, *StarTeam User's Guide*, Borland Software Corporation.

BRAGA, R.M.M., 2000, *Busca e Recuperação de Componentes em um Ambiente de Reuso*, Tese de D.Sc., COPPE, UFRJ, Rio de Janeiro, RJ.

BRERETON, P., BUDGEN, D., 2000, "Component-based systems: a classification of issues", v. 33, n. 11, pp. 54-62, November.

BROWN, A.W., 2000, *Large Scale Component Based Development*, 1st ed., Prentice Hall PTR.

BUGZILLA, 2006, "Bugzilla - Installation List". In: http://www.bugzilla.org/installation-list/, accessed in 17/01/2006.

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*, 1st ed., Addison-Wesley.

CHRISSIS, M.B., KONRAD, M., SHRUM, S., 2003, *CMMI®: Guidelines for Process Integration and Product Improvement*, 1st ed., Boston, MA, Addison Wesley Professional.

CMCROSSROADS, 2006, "Defect Tracking Software". In: http://resources.cmcrossroads.com/cmcrossroads/search/tabbrowse/software/1715/1/index.jsp?vf=, accessed in 10/02/2006.

COLLINS-SUSSMAN, B., FITZPATRICK, B.W., PILATO, C.M., 2004, *Version Control with Subversion*, 1st ed., O'Reilly.

COMPONENTSOURCE, 2006, "ComponentSource - Reporting a Problem". In: http://www.componentsource.com/Services/ReportingAProblem.asp, accessed in 15/02/2006.

CRNKOVIC, I., WILLFÖR, P., 1998, "Change Measurements in an SCM process". In: *Proceedings of the SCM-8 Symposium on System Configuration Management*, v. 1439, pp. 26-32 Brussels.

D'SOUZA, D., WILLS, A., 1998, *Objects, components, and frameworks with UML: The catalysis approach*, 1st ed., Addison Wesley.

DANTAS, A.R., VERONESE, G.O., CORREA, A.L., *et al.*, 2002, "Suporte a Padrões no Projeto de Software". In: *XVI Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas*, pp. 450-455, Gramado, RS, Brasil, Outubro.

DANTAS, C.R., 2005, *Odyssey-WI: Uma Abordagem para a Mineração de Rastros de Modificação de Modelos em Repositórios Versionados*, Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

DANTAS, C.R., MURTA, L.G.P., WERNER, C.M.L., 2005, "Consistent Evolution of UML Models by Automatic Detection of Change Traces". In: *International Workshop on Principles of Software Evolution (IWPSE)*, pp. 144-147, Lisbon, Portugal, September.

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.

DIRCKZE, R., 2002, *Java Metadata Interface (JMI) Specification - Version 1.0*, Unisys Corporation and Sun Microsystems.

EIA, 1998, *EIA 649 - National Consensus Standard for Configuration Management*, Electronic Industries Alliance.

ESTUBLIER, J., 2000, "Software Configuration Management: a Roadmap". In: *International Conference on Software Engineering, The Future of Software Engineering*, pp. 279-289, Limerick, Ireland, June.

ESTUBLIER, J., LEBLANG, D., VAN DER HOEK, A., *et al.*, 2005, "Impact of Software Engineering Research on the Practice of Software Configuration Management", *ACM Transactions on Software Engineering and Methodology*, v. 18, n. 4 (October), pp. 1-48.

FIGUEIREDO, S.M., SANTOS, G., ROCHA, A.R., 2004, "Gerência de Configuração em Ambientes de Desenvolvimento de Software Orientados à Organização". In: *III Simpósio Brasileiro de Qualidade de Software*, Brasília - DF, Junho.

FSF, 2006, "Free Software Foundation". In: http://www.fsf.org/, accessed in 10/02/2006.

GAMMA, E., HELM, R., JOHNSON, R., *et al.*, 1995, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed., Addison Wesley.

GAO, J.Z., TSO, H.-S.J., WU, Y., 2003, *Testing and quality assurance for component-based software*, 1st ed., Artech House.

HATCHER, E., LOUGHRAN, S., 2002, *Java Development with Ant*, 1st ed., 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, 1998, *Std 828 - IEEE Standard for Software Configuration Management Plans*, Institute of Electrical and Electronics Engineers.

IEEE, 2005, *Std 828 - IEEE Standard for Software Configuration Management Plans*, Institute of Electrical and Electronics Engineers.

ISO, 1995a, *ISO 10007, Quality Management - Guidelines for Configuration Management*, International Organization for Standardization.

ISO, 1995b, *ISO/IEC 12207 - Information technology - Software life cycle processes*, International Organization for Standardization

JENNINGS, N.R., SYCARA, K., WOOLDRIDGE, M.J., 1998, "A Roadmap of Agent Research and Development", *Journal of Autonomous Agents and Multi-Agent Systems*, v. 1, n. 1, pp. 7-38.

KNETHEN, A., GRUND, M., 2003, "QuaTrace: A Tool Environment for (Semi-) Automated Impact Analysis Based on Traces", pp. 246-255, Amsterdam, Netherlands, September 22-26

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, Deparment of Information Technology, Uppsala University, Sweden.

LEBSACK, C.S., MROCZEK, A.J., MUELLER, C.J., 2001, "Controlling Configuration Items in Component Based Software Development". In: *Tenth International Workshop on Software Configuration Management (SCM-10)*, Toronto, Canada, May, 2001.

LEHMAN, M.M., 1980, "Programs, life cycles and laws of software evolution". In: *Proceedings of the IEEE*, v. 68, pp. 1060-1076, Setembro.

LEHMAN, M.M., 1996, "Laws of Software Evolution Revisited". In: *Software Process Technology, 5th European Workshop, EWSPT'96*, v. 1149, pp. 108-124, Nancy, França, 9-11 de Outubro.

LEON, A., 2000, *A Guide to Software Configuration Management*, 1a ed., Norwood, MA, Artech House Publishers.

MANTIS, 2006, "Mantis Bug Tracker". In: http://www.mantisbt.org/, accessed in 19/02/2006.

MATULA, M., 2006, "NetBeans Metadata Repository". In: http://mdr.netbeans.org, accessed in 15/02/2006.

MAYER, R.J., MENZEL, C.P., PAINTER, M.K., *et al.*, 1995, *Information Integration for Concurrent Engineering (IICE) IDEF3 Process Description Capture Method Report*, Knowledge Based Systems, Inc.

MILER, N., 2000, *A Engenharia de Aplicações no Contexto da Reutilização baseada em Modelos de Domínio*, M.Sc., COPPE/UFRJ, Rio de Janeiro, Brasil.

MKGNU.NET, 2006, "mkgnu.net | Kristis' website". In: http://www.mkgnu.net/, accessed in 14/02/2006.

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.

MURTA, L.G.P., BARROS, M.O., WERNER, C.M.L., 2001, "FrameDoc: Um Framework para a Documentação de Componentes Reutilizáveis". In: *IV International Symposium on Knowledge Management/Document Management (ISKM/DM'2001)*, pp. 241-259, Curitiba, Brasil, Agosto.

MURTA, L.G.P., VASCONCELOS, A.P.V., BLOIS, A.P.T.B., *et al.*, 2004, "Run-time Variability through Component Dynamic Loading". In: *XVIII Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas*, pp. 67-72, Brasília, DF, Brasil, Outubro.

MYSQL AB, 2006, "MySQL AB". In: www.mysql.com, accessed in 10/02/2006.

OBJECTEERING, 2006, "Objecteering Software". In: http://www.objecteering.com/, accessed in 15/02/2006.

ODYSSEY, 2006, "Projeto Odyssey". In: http://reuse.cos.ufrj.br/odyssey/, accessed in 10/02/2006.

OLIVEIRA, H., MURTA, L.G.P., WERNER, C.M.L., 2005, "Odyssey-VCS: a Flexible Version Control System for UML Model Elements". In: *International Workshop on Software Configuration Management (SCM-12)*, pp. 1-16, Lisbon, Portugal, September.

OLIVEIRA, H.L.R., 2005, *Odyssey-VCS: Uma Abordagem de Controle de Versões para Elementos da UML*, Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

OMG, 2001, *OMG Unified Modeling Language Specification, Version 1.4*, Object Management Group.

OMG, 2002a, "MOF 2.0 Query / Views / Transformations RFP". In: http://www.omg.org/docs/ad/02-04-10.pdf, accessed in 02/01/2006.

OMG, 2002b, "MOF 2.0 Versioning and Development Lifecycle RFP". In: http://www.omg.org/cgi-bin/doc?ad/02-06-23, accessed in 15/02/2006.

OMG, 2005, "Software Process Engineering Metamodel (SPEM), Version 1.1". In: http://www.omg.org/technology/documents/formal/spem.htm, accessed in 10/02/2006.

OMG, 2006, "Object Management Group". In: http://www.omg.org accessed in 15/02/2006.

OSELLUS, 2006, "IRIS Suite". In: http://www.osellus.com/products/iris_suite/iris_suite.html, accessed in 15/02/2006.

OSI, 2006, "Open Source Initiative". In: http://www.opensource.org/index.php, accessed in 15/02/2006.

PFLEEGER, S.L., 2004, *Engenharia de software: teoria e prática*, 2a ed., Pearson Education do Brasil.

PHPBUGTRACKER, 2006, "phpBugTracker - A web-based bug tracking system", 31/01/2006.

PRESSMAN, R.S., 2005, *Software Engineering: A Practitioner's Approach*, 6th ed., McGraw-Hill.

ROUNDUP, 2006, "Roundup Issue Tracker". In: http://roundup.sourceforge.net/index.html, accessed in 15/02/2006.

SAMETINGER, J., 1997, "Software Engineering with Reusable Components", *Springer-Verlag*.

SCHLENOFF, C., GRUNINGER, M., TISSOT, F., *et al.*, 2000, *The Process Specification Language (PSL) Overview and Version 1.0 Specification*, National Institute of Standards and Technology, NISTIR 6459.

SCOTT, J.A., NISSE, D., 2001, "Software Configuration Management", *Guide to Software Engineering Body of Knowledge*, chapter 7, IEEE Computer Society Press.

SELECT BUSINESS SOLUTIONS, 2006, "Select Business Solutions - Component Based Development Modeling Tools for your Business". In: http://www.selectbs.com/, accessed in 25/01/2006.

SERENA, 2004, *TeamTrack 6.2 - TeamTrack Reviewer Guide*, Serena Software, Inc.

SLEEPYCAT, 2006, "Sleepycat Software: Berkeley DB". In: www.sleepycat.com, accessed in 10/02/2006.

SOFTEX, 2005, *MPS.BR - Melhoria de Processo do Software Brasileiro - Guia Geral (Versão 1.0)*.

SPARXSYSTEMS, 2006, "Enterprise Architect - UML Design Tools and UML CASE tools for software development". In: http://www.sparxsystems.com.au/ea.htm, accessed in 15/02/2006.

SUN, M., 2006a, "Java". In: http://www.java.sun.com accessed in 15/02/2006.

SUN, M., 2006b, "Java 2 Platform, Enterprise Edition (J2EE) 1.4". In: http://java.sun.com/j2ee/1.4/, accessed in 15/02/2006.

TIGRIS, 2006, "Scarab". In: http://scarab.tigris.org/ accessed in 19/02/2006.

TROLLTECH, 2006, "TrollTech - Qt Product Overview". In: http://www.trolltech.com/products/qt/index.html, accessed in 10/02/2006.

VERONESE, G.O., CORREA, A.L., JEZINNI, F., *et al.*, 2002, "ARES: Uma Ferramenta de Engenharia Reversa Java-UML". In: *XVI Simpósio Brasileiro de Engenharia de Software, Sessão de Ferramentas*, pp. 347-352, Gramado, RS, Brazil, Outubro.

VILLELA, K., OLIVEIRA, K., SANTOS, G., *et al.*, 2003, "Cordis-FBC: An Enterprise Oriented Software Development Environment". In: *Workshop Learning Software Organization*, pp. 91-96, Luzern, Switzerland, April.

VISIBLE, 2003, *Razor - Release Management, File Version Control, Problem Tracking*, Visible Systems Corporation.

WEBER, D.W., 1997, "Change sets versus change packages: Comparing implementations of change-based SCM." In: *Proceedings of the 7th International Workshop on Software Configuration Management (SCM-7)*, v. 1235, Boston, MA, May.

WEBSTER, M., 2004, *Worldwide Software Configuration Management Tools 2003 Vendor Shares*, IDC, Doc # 31689.

WFMC, 1999, "Interface 1: Process Definition Interchange". In: http://www.wfmc.org/standards/docs/TC-1016-P_v11_IF1_Process_definition_Interchange.pdf, accessed in 15/02/2006.

WHITE, B.A., 2000, *Software Configuration Management Strategies and Rational ClearCase: A Practical Introduction*, 1st ed., Addison-Wesley.

WINGERD, L., 2005, *Practical Perforce*, 1st ed., O'Reilly Media, Inc.