Uma Infra-estrutura Colaborativa de Apoio ao Desenvolvimento Distribuído Baseado em Componentes

João Paulo F. de Oliveira, Talles Brito, Yuri Morais, Amílcar Soares, Adriana E. de Oliveira, Sebastião Rabelo Jr, Glêdson Elias

Departamento de Informática
Universidade Federal da Paraíba (UFPB) – João Pessoa, PB – Brasil

{joaopaulo,talles,yuri,amilcar,drill,sebastiao,gledson}@compose.ufpb.br

Abstract. *Several organizations have adopted the distributed software development paradigm as a mean to reduce costs and development time, and also improve quality of products. As a consequence of geographic distribution of such organizations, tools and platforms are needed to improve collaboration and to control the software production process. In such a context, this paper presents a collaborative infrastructure for supporting component-based distributed development, which integrates facilities for access control, configuration management, version control and reuse metrics management.*

Resumo. *Diversas empresas estão passando a adotar o paradigma de desenvolvimento distribuído de software, com o objetivo de reduzir os custos e o tempo de desenvolvimento, bem como melhorar a qualidade dos seus produtos. Em decorrência da distribuição geográfica dessas empresas, ferramentas e plataformas são necessárias para melhorar a colaboração e controlar o processo de produção do software. Neste sentido, este artigo apresenta uma infra-estrutura colaborativa de apoio ao desenvolvimento distribuído baseado em componentes de software, que integra facilidades de controle de acesso, gerência de configuração, controle de versões e gerência de métricas de reuso.*

1. Introdução

Tem se tornado cada vez mais oneroso desenvolver software no mesmo espaço físico, na mesma organização ou até mesmo no mesmo país [1]. Assim, em um ambiente extremamente competitivo, empresas de Tecnologia da Informação (TI) que desejam continuar crescendo, ou até mesmo sobrevivendo, necessitam mudar inevitavelmente seus processos de desenvolvimento de software. No entanto, de acordo com Aoyama [2], os métodos tradicionais de desenvolvimento de software não permitem que tais empresas possam atingir os resultados esperados.

Neste sentido, na engenharia de software, o Desenvolvimento Distribuído de Software (DDS) e o Desenvolvimento Baseado em Componentes (DBC) destacam-se como abordagens que têm o potencial de tornar realidade a redução da complexidade, do tempo e do custo de desenvolvimento, bem como melhorar a qualidade do software produzido [3] e a possibilidade de obter recursos em âmbito global [4].

Vale ressaltar que existe uma relação entre as abordagens de DBC e DDS. Esta relação explicitada na forma de um mercado de software, como definido por Wallnau [5], onde o produtor fornece componentes de software para diversos consumidores e um consumidor adquire componentes de diversos produtores, ambos (produtores e consumidores) dispersos geograficamente. É importante destacar que, nesta relação, consumidores desempenham o papel de integradores de software, desenvolvendo novos

componentes a partir do reuso e composição de componentes previamente existentes e desenvolvidos de forma independente por terceiros.

Entretanto, tanto o DBC quanto o DDS apresentam alguns fatores limitantes. No DBC, para Crnkovic [6], o desafio está na atual dificuldade de identificar, selecionar, negociar e recuperar componentes que atendam aos requisitos especificados, e, além disso, que possuam alto grau de reusabilidade e qualidade. Já no DDS, o principal desafio é inerente à própria característica da distribuição que dificulta a colaboração e a cooperação entre departamentos e pequenos grupos de desenvolvedores que trabalham em conjunto, mas estão localizados em cidades ou países diferentes [7].

Para Tommarello [8], trabalho com equipes distribuídas é sempre um desafio. No entanto, esse desafio pode ser reduzido com o uso de ferramentas e ambientes de *groupware* para coordenação e comunicação das equipes durante um processo de desenvolvimento distribuído de software.

Neste contexto, com o objetivo de superar as limitações expostas tanto para o DBC quanto para o DDS, este artigo apresenta a implementação de um serviço de repositório compartilhado e distribuído, que provê facilidades de *groupware* atuando como uma infra-estrutura colaborativa de apoio ao desenvolvimento distribuído de componentes de software. O serviço de repositório provê facilidades e suporte a modelos de negócios, certificação de componentes, aspectos de segurança (autenticação, visibilidade e controle de acesso), esquema de nomeação, controle de versões e informações de reuso.

O serviço de repositório proposto é dito ser compartilhado, pois vários produtores podem registrar seus artefatos. Por outro lado, o serviço de repositório é dito distribuído porque uma implementação aderente pode ser baseada em um conjunto de entidades geograficamente dispersas que cooperam para prover as funcionalidades do serviço de repositório.

Vale ressaltar que o serviço de repositório é projetado de forma a manipular artefatos de software (*assets*) em geral, que incluem quaisquer artefatos produzidos ou reusados no DBC. Exemplos de artefatos são: códigos executáveis, códigos fonte, especificações de interfaces, documentações, casos de uso, diagramas UML e planos de teste.

O restante deste trabalho está organizado da seguinte forma. Na Seção 2, o serviço de repositório é comparado com alguns trabalhos relacionados. A Seção 3 introduz o *framework* arquitetural *ComponentForge* [9], no qual o serviço de repositório representa a infra-estrutura colaborativa de apoio ao desenvolvimento distribuído de componentes de software. A Seção 4 apresenta as principais características e funcionalidades do serviço de repositório que provêem apoio a colaboração e cooperação. Em seguida, a Seção 5 apresenta o projeto arquitetural do serviço de repositório e sua implementação. Por fim, a Seção 6 apresenta algumas considerações finais.

2. Trabalhos Relacionados

Avaliando as ferramentas e plataformas de DBC, disponíveis na academia e indústria, torna-se evidente a ausência de um ferramental de *groupware* para coordenação e comunicação das equipes durante um processo de desenvolvimento distribuído de componentes de software. Para evidenciar esta ausência de ferramental de groupware no contexto de DBC, esta seção apresenta uma breve comparação das funcionalidades providas pelo serviço de repositório proposto neste artigo com os

principais repositórios disponíveis na literatura e mercado, tais como: *SPARS-J* [10], *OSCAR* [11], *CodeBroker* [12] e *ComponentSource*[13].

A fim de facilitar a análise dos repositórios, resolvemos adotar o Modelo *Clover* [14] para classificação do apoio computacional à colaboração. Neste modelo, uma plataforma de *groupware* deve fornecer três classes de serviços complementares: *coordenação*, *produção* e *comunicação*.

As facilidades de *coordenação* estão relacionadas com os papéis que cada usuário pode exercer e como essas atividades são coordenadas de forma a alcançar o objetivo final. O *ComponentSource*, o *SPARS-J* e o *CodeBroker* adotam mecanismos para a coordenação de atividades bastante simples, pois não permitem aos usuários exercerem diferentes papéis com distintas responsabilidades. Em ambos os casos, os produtores apenas registram seus componentes implementados e prontos, não tendo a noção de equipes distribuídas que cooperam durante o processo de desenvolvimento de software.

De forma similar ao *ComponentSource*, *SPARS-J* e *CodeBroker*, o *OSCAR* também não adota mecanismos sofisticados para a coordenação de atividades. Na literatura, apenas é possível identificar que o *OSCAR* adota um simples esquema de histórico (*log*), que armazena quem acessou, qual artefato foi acessado e qual foi o propósito do acesso. No entanto, estas informações são apenas usadas para auditoria, mas não podem ser usadas como mecanismo de controle na definição de diferentes papéis aos usuários.

Ao contrário das principais propostas identificadas na literatura e indústria, o serviço de repositório proposto neste artigo adota mecanismos sofisticados para a coordenação das atividades, permitindo aos usuários exercerem diferentes papéis com distintas responsabilidades. O serviço de repositório adota o modelo de controle de acesso denominado RBAC (*Role-Based Access Control*) [15], onde os usuários são classificados em diferentes papéis e diferentes permissões de acesso são associadas a cada papel desempenhado pelo usuário. Deste modo, de forma simples e de fácil manutenção, é possível controlar que operações as equipes distribuídas de desenvolvedores podem executar durante o processo de desenvolvimento de componentes de software.

De acordo com Laurillau [14], a segunda classe de serviço requerido por uma plataforma de *groupware* está relacionada com facilidades de *produção*. Neste contexto, a produção está relacionada com os resultados das atividades desenvolvidas em grupo ou aos dados compartilhados por múltiplos usuários. No caso específico de DBC e DDS, a produção está diretamente relacionada às facilidades de gerência de configuração, controle de versão e certificação dos componentes de software. Além disso, a produção também diz respeito à quantidade de produtores e consumidores que podem fazer uso das facilidades da plataforma de *groupware*, como também à quantidade de componentes que podem ser manipulados.

Em relação às facilidades de produção, o *CodeBroker*, o *SPARS-J* e o *OSCAR* adotam uma base de dados de escala restrita a uma determinada instituição, que impede a escalabilidade do serviço oferecido, tanto em termos da quantidade de produtores e consumidores que usam os serviços, quanto na quantidade de componentes manipulados. Embora o *ComponentSource* atue provendo componentes de software comerciais (COTS – *Commercial Off-The-Shelf*) em escala global e multi-institucional, o aspecto centralizado de sua arquitetura o torna não escalável em termos da quantidade

de produtores e consumidores que usam os serviços, quanto na quantidade de componentes manipulados.

Além disso, o *CodeBroker* e o *SPARS-J* também não adotam qualquer mecanismo de gerência de configuração, controle de versão e certificação dos componentes. Logo, no *CodeBroker* e *SPARS-J*, as facilidades de produção são bastante limitadas, abrangendo apenas a possibilidade de registrar e recuperar artefatos. Considerando a gerência de configuração e o controle de versões suportados pelo OSCAR, pode-se perceber que o mesmo adota abordagens baseadas na integração de soluções de terceiros e centradas apenas no controle de código fonte, tais como CVS. Além disso, o OSCAR não trata aspectos relacionados à certificação de componentes.

Por outro lado, no que se refere às facilidades de produção, o serviço de repositório proposto neste artigo define mecanismos próprios de gerência de configuração e controle de versões, capazes de controlar de forma customizada a evolução dos diferentes tipos de artefatos, ou seja, não apenas código fonte. Além disso, o serviço de repositório suporta diferentes estratégias de certificação de componentes, cujos certificados podem ser registrados por diferentes entidades certificadoras que adotam variados modelos de certificação. Por fim, a natureza compartilhada e distribuída do serviço de repositório torna a solução escalável em termos do número de produtores e consumidores participantes, como também da quantidade de componentes manipulados.

Por fim, ainda de acordo com Laurillau [14], a terceira classe de serviço requerido por uma plataforma de *groupware* está relacionada com facilidades de *comunicação*, que trata das relações diretas entre os usuários, como por exemplo, uma possível troca de mensagens entre desenvolvedores e consumidores. O *ComponentSource*, por ser um repositório comercial, oferece *chat* e fórum para que os usuários possam obter mais detalhes sobre os artefatos que desejam adquirir. O *OSCAR*, *SPARS-J* e *CodeBroker* não oferecem nenhum apoio a facilidades de comunicação entre produtores e consumidores.

Diferentemente, em relação às facilidades de comunicação, o serviço de repositório apresentado neste artigo oferece um conjunto de informações de métricas de reuso associadas aos artefatos registrados. Estas informações contêm os comentários dos consumidores sobre suas experiências de instalação, configuração e uso de um determinado artefato, como também as respostas apresentadas pelos desenvolvedores a tais comentários. Neste sentido, as informações de reuso permitem os produtores e consumidores compartilharem informações que facilitam a manutenção dos artefatos e também auxiliam os consumidores na seleção dos mesmos.

3. ComponentForge

Considerando as limitações das propostas atualmente disponíveis, este artigo descreve as facilidades de *groupware* suportadas pela implementação de um serviço de repositório compartilhado e distribuído. Este serviço de repositório proposto atua como uma infra-estrutura colaborativa de apoio ao desenvolvimento distribuído de componentes de software. Atuando como um elemento de infra-estrutura, o serviço de repositório está inserido no contexto de um *framework* arquitetural de alto nível, denominado *ComponentForge*, que provê apoio a processos de desenvolvimento baseados em reuso de componentes. Esta sessão tem por objetivo descrever sucintamente os elementos que compõem o *framework*, contextualizando assim o papel do serviço de repositório.

Como ilustrado na Figura 1, o *ComponentForge* adota uma abordagem de arquitetura orientada a serviços (SOA – *Service-Oriented Architecture*), na qual um conjunto de serviços distribuídos, compartilhados, independentes e fracamente acoplados comunicam-se e colaboram entre si através de interfaces e protocolos de comunicação bem definidos.

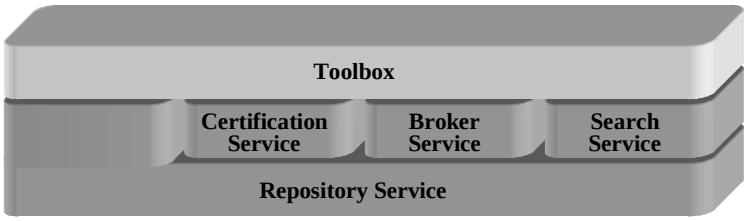

Figura 1. Framework Arquitetural do ComponentForge

Coletivamente, o conjunto de serviços do *framework* provê facilidades para armazenar, indexar, buscar, recuperar, certificar e negociar variados artefatos de software (*assets*). Além disso, o *ComponentForge* inclui facilidades de gerência de configuração, gerência de métricas de reuso, como também aspectos de segurança relacionados à autenticação e controle de acesso.

O serviço de repositório (*repository service*) é responsável por armazenar, localizar, recuperar e gerenciar *assets*. O serviço de repositório é suportado por uma coleção de entidades cooperantes e distribuídas denominadas *containers*, que, conjuntamente, provêem facilidades às demais entidades do *framework* arquitetural. Apesar de não estar explícito na Figura 1, o serviço de repositório pode se comunicar com uma ou mais instâncias de *toolbox*, *certification service*, *broker service* e *search service*.

A certificação de componentes é uma atividade que procura validar as funcionalidades e os níveis de qualidade dos componentes [16]. O *ComponentForge* foi concebido de forma a permitir a coexistência de diversos serviços de certificação (*certification service*), possibilitando a existência de múltiplos e complementares processos de certificação. Estes serviços são responsáveis por certificar não apenas componentes individuais, mas também as práticas e os processos adotados pelos produtores.

Da mesma forma como acontece na produção de softwares em geral, produtores de componentes reusáveis também podem estar interessados ou não em ter um retorno financeiro. Logo, podem-se identificar dois tipos de componentes: os *componentes sem modelo de negócio*, que podem ser distribuídos para qualquer consumidor sem restrições, e os *componentes com modelo de negócio*, cujas formas de negociação são regidas por contratos especificados nos modelos de negócios, definidos por seus respectivos produtores. Portanto, componentes podem ser adquiridos sob uma variedade de modelos de negócios.

Sendo assim, o serviço de negociação (*broker service*) tem a função de prover mecanismos para assegurar que um componente será entregue em conformidade com os modelos de negócios especificados pelo produtor. Depois que um consumidor atender todas as condições dos modelos de negócios de um componente, o serviço de negociação pode acessar o serviço de repositório para recuperar o componente e entregá-lo ao consumidor. Os produtores de componentes podem desenvolver seus próprios serviços de negociação ou indicar serviços já existentes para tratar os modelos de negócios sob os quais seus componentes podem ser negociados.

Vale ressaltar que, mesmo componentes desenvolvidos sob uma licença de software livre, também podem ter que satisfazer restrições especificadas em um determinado modelo de negócio. Neste caso, tais componentes somente podem ser adquiridos através de um determinado serviço de negociação. Nesta abordagem, por exemplo, produtores de software livre podem manter, em seus respectivos serviços de negociação, catálogos de consumidores aptos a recuperar seus componentes ou consumidores que já realizaram a recuperação de seus componentes.

Para permitir a busca e a recuperação de componentes armazenados, o *ComponentForge* define o serviço de busca (*search service*). Este serviço consulta o serviço de repositório e recupera metadados para realizar a indexação. O *ComponentForge* explora um modelo de representação de componentes, denominado X-ARM [17]. Tal abordagem torna a arquitetura mais flexível, pois permite que diferentes serviços de busca adotem diferentes algoritmos de indexação e linguagens de consulta. Desta forma, um serviço de busca pode indexar componentes de um determinado domínio de aplicação específico, e, além disso, prover uma linguagem de consulta especializada para tal domínio. Por outro lado, outro serviço de busca pode optar por indexar componentes de forma independente de domínio de aplicação.

Por fim, o conjunto de ferramentas (*toolbox*) permite a interação dos usuários com os demais serviços do *ComponentForge*. Vale ressaltar que o baixo acoplamento entre os serviços que compõem o *framework* facilita a configuração, reuso, manutenção, evolução e extensão, todos considerados importantes propriedades arquiteturais de aplicações distribuídas, conforme mencionado por [18].

4. O Serviço de Repositório como uma Infra-estrutura Colaborativa

Colaboração é uma maneira de desenvolver atividades em grupo. Visando a colaboração no desenvolvimento de componentes de software, nesta seção serão apresentadas as principais funcionalidades do serviço de repositório, evidenciando aquelas diretamente relacionadas às atividades colaborativas.

Para a classificação das funcionalidades do serviço de repositório utilizaremos novamente o Modelo *Clover* [14]. Como mencionado, de acordo com este modelo, uma plataforma de *groupware* deve prover três classes de serviços: *coordenação, produção* e *comunicação*.

Nas facilidades de coordenação, a estruturação de uma equipe inclui a definição dos papéis dos usuários, da hierarquia, dos subgrupos e das permissões dos participantes[19]. A fim de gerenciar quais usuários terão permissão para executar determinadas operações e quais serviços terão permissões de acesso aos artefatos, o serviço de repositório possui um sistema de controle de acesso. Além disso, o serviço de repositório provê um esquema de visibilidade para que o desenvolvedor possa restringir o acesso a seus artefatos.

No trabalho em grupo, as facilidades de produção estão relacionadas à operação conjunta dos participantes no espaço compartilhado, visando a realização de tarefas [19]. Além disso, as facilidades de produção devem permitir que os desenvolvedores possam manipular, refinar e organizar os artefatos produzidos. Assim, no serviço de repositório, o controle de versões tem por objetivo permitir que diversos desenvolvedores possam colaborar na evolução de um mesmo artefato de forma organizada e sem conflitos. Além disso, o esquema de nomeação proporciona a recuperação não ambígua de artefatos, facilitando assim a organização da produção de todo o conteúdo do serviço de repositório.

Através das facilidades de comunicação, o grupo debate pontos de vista para alinhar e refinar as idéias [19]. Neste contexto, o serviço de repositório define o que chamamos de informações de reuso, onde os consumidores podem expor suas avaliações a respeito dos artefatos recuperados.

Neste ponto, apenas identificamos as principais funcionalidades do serviço de repositório que estão diretamente relacionadas às atividades colaborativas. A seguir, as próximas subseções detalham os conceitos e funcionalidades sobre nomeação, controle de acesso aos usuários e demais serviços do *framework* , visibilidade dos artefatos, controle de versões dos artefatos e informações de reuso.

4.1 Esquema de Nomeação

O serviço de repositório é composto por um conjunto de entidades cooperantes e distribuídas denominadas *containers*. A Figura 2 ilustra a visão distribuída e compartilhada do serviço de repositório, em que o mesmo é uma entidade representativa do conjunto de facilidades providas pelos *containers*.

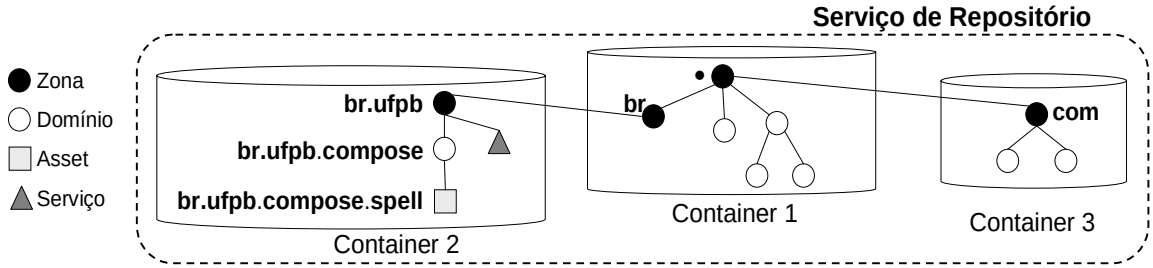

Figura 2. Árvore de Nomes Hierárquicos

Assim, considerando a natureza distribuída e compartilhada do serviço de repositório, as atividades de *groupware* relacionadas às facilidades de produção dos artefatos armazenados no serviço de repositório utilizam um esquema que permite a recuperação não ambígua de artefatos. Em tal esquema, os nomes são organizados em uma árvore hierárquica onde as folhas são os artefatos armazenados e serviços oferecidos. Nesta árvore, os nós internos correspondem a dois tipos de entidades (zonas e domínios).

Zonas representam produtores de artefatos e suas famílias de produtos e serviços. A fim de reduzir o esforço de coordenação e possibilitar uma melhor organização dos artefatos, uma zona também pode ser subdividida em uma estrutura hierárquica, na qual os nós são denominados domínios, cuja função é agrupar um conjunto de artefatos para um determinado domínio de aplicação.

Na composição dos identificadores hierárquicos, os nomes de zonas, domínios, serviços e artefatos são separados por pontos. Assim, a Figura 2 ilustra uma árvore de nomes hierárquicos onde *br.ufpb* refere-se à zona da UFPB, que possui o domínio *br.ufpb.compose*, que representa o grupo COMPOSE e este possui um artefato denominado *br.ufpb.compose.spell*. Vale ressaltar que a zona raiz do serviço de repositório é identificada apenas por um ponto.

Além de definir identificadores globalmente únicos, o esquema hierárquico proposto também provê transparência de localização, pois os nomes não referenciam a localização física, mas apenas a organização lógica de produtores, artefatos e serviços.

4.2 Controle de Acesso

O controle de acesso provido pelo serviço de repositório é um mecanismo relacionado às atividades de *groupware* associadas às facilidades de coordenação. O controle de

acesso gerencia as operações que podem ser executadas por um usuário, e, também é o responsável pela definição das permissões de acesso dos serviços aos artefatos, zonas e domínios.

Os usuários do serviço de repositório podem executar tarefas distintas. O gerente de container realiza tarefas administrativas, registrando zonas autorizadas e verificando dados estatísticos relacionados com o container. Um administrador de zona gerencia uma determinada zona, criando e removendo domínios, assim como registrando, removendo e atualizando informações sobre desenvolvedores autorizados. Um desenvolvedor é responsável por registrar, atualizar e remover artefatos em sua respectiva zona. Já um consumidor pode procurar e adquirir artefatos previamente armazenados. Além disso, os serviços de certificação, negociação e busca também podem acessar as facilidades providas pelo serviço de repositório.

No contexto de controle de acesso, o serviço de repositório adota um mecanismo semelhante ao modelo RBAC (*Role-Based Access Control*) [15], onde as permissões de acesso são associadas a papéis, que, por sua vez, são atribuídos aos usuários e serviços. Assim, quando um novo usuário ou serviço é cadastrado, o mesmo é associado a um conjunto de papéis, e, assim, suas permissões de acesso são automaticamente atribuídas. Consequentemente, o gerenciamento do controle de acesso torna-se mais simples e viável.

Permissões de acesso associadas para usuários e serviços são diferentes. No caso dos usuários, permissões de acesso simplesmente referem-se para operações permitidas. Entretanto, no caso de serviços, permissões de acesso referem-se a zonas e domínios permitidos ou bloqueados. A Figura 3 exemplifica tais permissões para serviços. A zona *br.ufpb* tem o papel *authservice*, composto de um conjunto de permissões de acesso que controlam quais componentes podem ser acessados pelos serviços de busca *com.google* e *com.yahoo*. Neste caso, é importante enfatizar que os serviços *com.google* e *com.yahoo* também devem estar registrados no serviço de repositório.

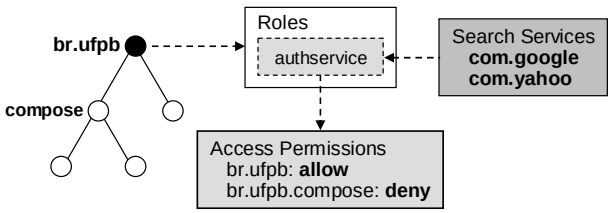

Figura 3. Permissões de Acesso para Serviços

Nas permissões de acesso, é possível permitir ou bloquear acesso para componentes específicos ou todos os componentes dos domínios indicados de uma maneira recursiva. Na Figura 3, veja que os serviços de busca *com.google* e *com.yahoo* têm permissão para acessar todos os artefatos da zona *br.ufpb*, recursivamente incluindo todos os seus domínios abaixo, mas recursivamente excluindo o domínio *br.ufpb.compose* e todos abaixo dele.

4.3 Visibilidade Externa

A fim de facilitar o desenvolvimento colaborativo e controlar o acesso aos artefatos por diferentes equipes distribuídas de desenvolvedores, o serviço de repositório explora o conceito de visibilidade. Assim, para cada artefato registrado em uma zona ou domínio, podem ser definidos esquemas que indicam as outras zonas ou domínios cujas equipes

de desenvolvedores são autorizadas ou proibidas de recuperar o artefato. Vale ressaltar que este esquema só pode ser modificado pelos desenvolvedores que o configuraram.

A visibilidade é considerada um mecanismo relacionado com as atividades de *groupware* associadas às facilidades de coordenação, pois proporciona um melhor controle dos artefatos de uma zona ou domínio, evitando que artefatos não concluídos sejam recuperados por outros desenvolvedores não autorizados e até mesmo por consumidores.

Quando um desenvolvedor autorizar ou proibir uma determinada zona ou domínio, esta autorização ou proibição é recursivamente aplicada a todos os domínios filhos em qualquer nível de profundidade, exceto quando esta autorização ou proibição é explicitamente negada para algum destes subdomínios.

Vale ressaltar que se um artefato não possui um esquema de visibilidade configurado, qualquer consumidor pode recuperar este artefato. Neste caso, apenas desenvolvedores permitidos no domínio onde o artefato está registrado podem executar operações envolvendo a manipulação das informações do artefato.

A Figura 4 ilustra o uso do esquema de visibilidade. Neste caso, o artefato *br.ufpb.compose.spell-1.0* pode ser acessado por todos os desenvolvedores dos domínios *br.domainX* e *br.domainZ*, incluindo aqueles dos seus respectivos subdomínios de qualquer nível de profundidade, exceto do subdomínio *br.domainX.domainY*.

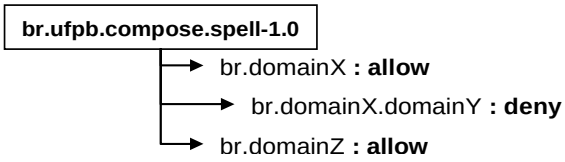

Figura 4. Esquema de visibilidade

4.4 Controle de Versões

No serviço de repositório, diversos desenvolvedores podem estar manipulando um mesmo artefato. Desta forma, um mecanismo para controle de versões dos artefatos é fundamental.

O serviço de repositório oferece este tipo de apoio durante o processo de desenvolvimento dos artefatos, permitindo assim, que os desenvolvedores pertencentes a um mesmo grupo de trabalho e distribuídos geograficamente, participem da produção de um mesmo artefato simultaneamente. Cada desenvolvedor pode trabalhar em uma versão base e submeter a sua versão modificada, sem correr o risco de sobrescrever a versão submetida por outros desenvolvedores. Para tal, o serviço de repositório oferece um mecanismo de bloqueio de artefatos.

Quando o desenvolvedor recupera um artefato que servirá como versão base para o desenvolvimento, o serviço de repositório bloqueia este artefato para que outros desenvolvedores não possam modificá-lo. Sendo assim, este artefato somente poderá ser atualizado por outros desenvolvedores depois que o mesmo for desbloqueado pelo desenvolvedor que o bloqueou. Assim, o registro de uma nova versão de um artefato automaticamente desbloqueia a versão base deste artefato. Este esquema de proteção garante o controle de versões para os artefatos que estão em desenvolvimento no serviço de repositório. O controle de versões adotado é semelhante aos empregados em outros sistemas de controle de versão comerciais, diferenciando pelo fato de que artefatos não são somente código, mas qualquer elemento produzido durante o processo de desenvolvimento.

No serviço de repositório, é possível que um conjunto de desenvolvedores trabalhem cooperativamente em uma versão de um artefato sem que seja preciso torná-la pública aos consumidores. Para tal, os artefatos são marcados como *registrados* ou *publicados*. Um artefato no estado registrado somente pode ser recuperado pelos desenvolvedores e serviços de busca e certificação autorizados. Por outro lado, um artefato no estado publicado pode ser recuperado pelos desenvolvedores e todos os serviços autorizados (serviços de busca, certificação e negociação), como também pelos consumidores. Para Gerosa [19], este tipo de mecanismo é um importante aspecto que deve ser disponibilizado por plataformas de *groupware*.

Vale ressaltar que o controle de versões provido pelo serviço de repositório também permite identificar e recuperar um histórico de evolução dos artefatos. Com isto, edições anteriores podem ser facilmente recuperadas, possibilitando ao usuário do serviço a comparação de versões.

4.5 Informações de Reuso

Estudos demonstram que a prática de reutilização ainda é muito dependente de comunicação entre as equipes produtoras e consumidoras de componentes. A documentação é suficiente para descrever o conhecimento de experiências, mas a comunicação direta é necessária para resolver diversos problemas e estabelecer relações de confiança entre os produtores e consumidores de software [20].

Então, para o desenvolvedor de artefatos é importante receber um retorno dos consumidores o mais cedo possível. Além disso, a possibilidade de o consumidor opinar sobre um determinado artefato, pode fazer com que o mesmo se sinta parte do processo de desenvolvimento do artefato [21].

Com a finalidade de estabelecer um canal de comunicação entre consumidores e desenvolvedores, o serviço de repositório define o que chamamos de informações de reuso. Estas consistem de um conjunto de informações referentes ao grau de satisfação de consumidores que já adquiriram um determinado artefato, seus comentários sobre a aquisição, área de aplicação do artefato, além de respostas emitidas pelos desenvolvedores do artefato sobre os comentários do consumidor. Neste contexto, as informações de reuso consistem em um elemento facilitador da comunicação e da cooperação entre usuários e desenvolvedores de maneira a aumentar a qualidade dos artefatos desenvolvidos.

É importante ressaltar que estas informações de reuso ficam a disposição de futuros consumidores, que podem se valer da experiência relatada por outros consumidores para ajudar no processo de escolha de um determinado artefato.

5. Projeto e Implementação do Serviço de Repositório

Em conjunto, as interfaces do serviço de repositório definem mais de 800 operações, que, atualmente, estão implementadas no protótipo piloto. No entanto, devido à limitação de espaço, as interfaces e o projeto do protótipo não serão descritos. A seguir, descreveremos a arquitetura definida para o repositório juntamente com a função e delimitação de cada um de seus serviços. Por fim, uma visão geral sobre a implementação desta arquitetura será apresentada.

5.1 Projeto Arquitetural

Como mencionado anteriormente, as facilidades do serviço de repositório são providas pelos *containers*, que possuem um projeto arquitetural no estilo em camadas, onde cada

uma destas provê um conjunto de componentes, que também adotam a abordagem de serviços. A Figura 5 ilustra a arquitetura em camadas do *container*.

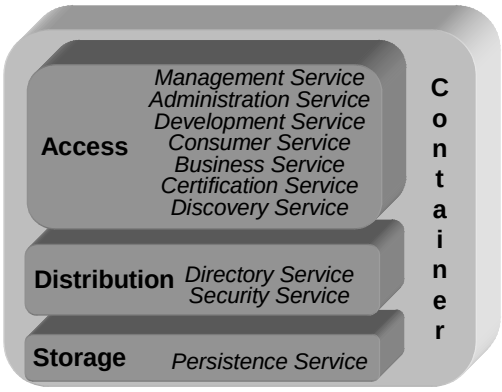

Figura 5. Camadas e serviços do container

A camada de acesso (*access layer*) permite a interação das ferramentas e dos outros serviços do *ComponentForge* com o container e com o serviço de repositório como um todo. Nesta camada, o serviço de gerenciamento (*management service*) oferece um conjunto de interfaces que permite a administração do container, tais como o registro, ajuste e remoção de zonas. O serviço de administração (*administration service*) provê operações para o gerenciamento de zonas e domínios. O serviço do desenvolvedor (*development service*) provê operações para o registro, atualização, remoção e gerenciamento de artefatos. O serviço do consumidor (*consumer service*) provê mecanismos para recuperação de artefatos que não adotam modelos de negócios, enquanto que, o serviço de negociação (*business service*) permite a recuperação de artefatos que adotam modelos de negócios. O serviço de certificação (*certification service*) trata aspectos da certificação de artefatos. Por fim, o serviço de descoberta (*discovery service*) permite a recuperação de artefatos para posterior indexação em mecanismos de busca.

A camada de distribuição (*distribution layer*) trata de aspectos não funcionais do serviço de repositório, resolvendo questões de distribuição e segurança, tornando possível a descoberta transparente e a recuperação segura de artefatos armazenados. A fim de prover tais facilidades, a camada de distribuição oferece o serviço de diretório (*directory service*) e o serviço de segurança (*security service*).

Os *containers* também são responsáveis pelo armazenamento físico dos artefatos, formando uma base de dados distribuída. Dessa forma, cada *container* possui a camada de armazenamento (*storage layer*), que disponibiliza o serviço de persistência (*persistence service*). Este serviço é acessado pelos serviços da camada de acesso e de distribuição, oferecendo meios para armazenar, atualizar e recuperar artefatos.

5.2 Implementação

A Figura 6 apresenta uma visão geral da implementação do serviço de repositório, identificando as diversas tecnologias envolvidas. Devido ao grande volume de dados manipulados pelo serviço de repositório, surgiu a necessidade de uma plataforma que proporcione manutenibilidade, confiabilidade e escalabilidade ao sistema. A especificação *Java Platform Enterprise Edition* (Java EE) [22] consegue satisfazer estes requisitos, pois tem como principal objetivo o desenvolvimento em n-camadas. Para execução de um sistema Java EE é necessário um servidor de aplicações, e para isso foi selecionado o *JBoss Application Server* [23]. Este preenche os requisitos de qualquer

servidor de aplicações poderoso, como *clustering*, tolerância a falhas, balanceamento de cargas, *caching* e está de acordo com as mais recentes especificações do *Enterprise JavaBeans* (EJB).

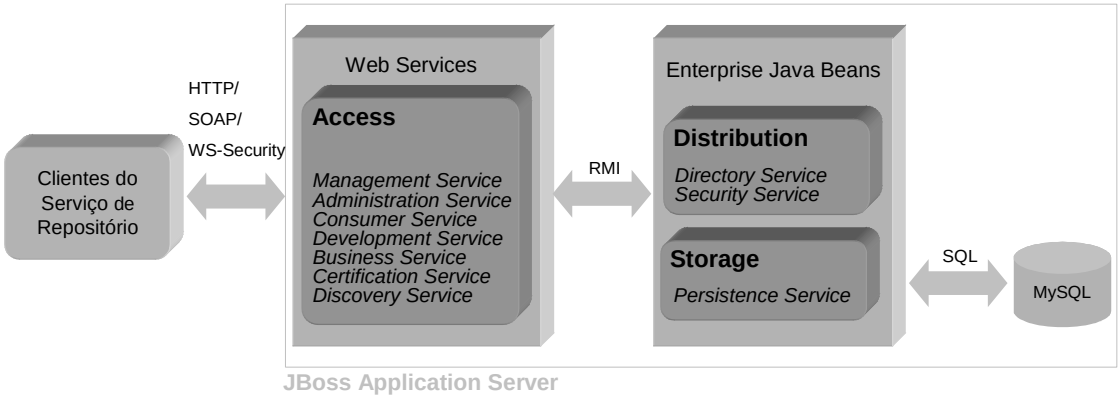

Figura 6. Visão geral sobre a implementação do serviço de repositório.

Um dos requisitos chaves para sistemas colaborativos é que estes sejam compatíveis e adaptáveis aos variados ambientes de seus usuários [24]. Em função disso, e, pelo fato do *ComponentForge* ser baseado em uma arquitetura orientada a serviços (SOA), decidimos disponibilizar seus serviços via *Web Services* [25]. Esta decisão proporciona interoperabilidade, permitindo que instâncias implementadas em diferentes plataformas e linguagens de programação possam se comunicar simplesmente expondo suas interfaces. Além disso, *Web Services* favorece o acesso de diferentes maneiras, que podem incluir, por exemplo, interfaces do usuário baseadas em navegadores *Web*, aplicações *desktop* e dispositivos móveis.

Com o uso de *Web Services*, questões sobre segurança, integridade e confidencialidade constituem fatores críticos nas trocas de mensagens. Assim, se faz necessário adotar meios para tratar estas questões, que incluem autenticação, autorização e criptografia [26]. Para preencher esta lacuna, a OASIS (*Organization for the Advancement of Structured Information Standards*) desenvolveu a especificação WS-Security [27]. Este padrão define uma forma de tratar os vários cenários de segurança na comunicação baseada em mensagens SOAP. WS-Security define um único modelo que abstrai serviços de segurança, separando as características funcionais de segurança do sistema de sua implementação específica [26]. Para realizar o controle de acesso (Sessão 4.2), o serviço de repositório deve suportar mecanismos de autenticação dos usuários e integridade das informações. Para tal, a implementação adota as facilidades providas pelo WS-Security.

Por fim, em função da natureza compartilhada e distribuída do *ComponentForge*, mecanismos para o gerenciamento de dados persistentes e não persistentes do serviço de repositório são de fundamental importância. Assim, devido ao serviço de repositório adotar um esquema de *cache* de resolução de nomes, alguns dados devem permanecer em memória principal em vez de serem armazenados em memória persistente, proporcionando um melhor desempenho na resolução de nomes. Toda a questão de sincronização e transações destes dados é gerenciada pelo próprio *JBoss Application Server*, de maneira que ganhamos em produtividade ao eliminar qualquer tipo de programação propensa a erros.

Além disto, o ComponentForge manipula e armazena uma grande quantidade de dados persistentes. Assim, para oferecer suporte a acessos simultâneos e em larga escala, fez-se necessária a utilização de um SGBD robusto, rápido, multi-threaded e multi-usuário. Com isso, dentre os principais SGBDs hoje existentes, o MySQL (5.0) atendeu a esses requisitos [28], além de dar suporte às necessidades do projeto quanto às questões de armazenamento e acesso concorrente.

6. Considerações Finais

O Desenvolvimento Distribuído de Software, assim como o Desenvolvimento Baseado em Componentes, vêm ganhado cada vez mais importância na Engenharia de Software em função da qualidade dos produtos gerados quando são adotados processos, técnicas e ferramentas aderentes a tais abordagens de desenvolvimento. A interseção entre essas duas abordagens oferece um vasto campo para pesquisa, visto a necessidade cada vez mais latente de ferramentas que dêem apoio a ambas abordagens.

Consequentemente, a principal contribuição deste artigo é propor um serviço de repositório que define uma infra-estrutura colaborativa que oferece suporte à coordenação, comunicação e produção de componentes de software. Por exemplo, o serviço de repositório provê facilidades de controle de versão, autenticação, controle de acesso e visibilidade, que são fundamentais para o apoio à colaboração entre equipes distribuídas de desenvolvimento.

Além disso, em função da componentização, o serviço de repositório define uma arquitetura que proporciona manutenibilidade, reuso, composição, extensibilidade, integração e escalabilidade. Segundo Greenberg [29], o uso de componentes que encapsulam as complexidades do desenvolvimento de facilidades de *groupware* é uma maneira de impulsionar o desenvolvimento desta tecnologia.

Vale ressaltar que o compartilhamento e a distribuição providos pelo serviço de repositório, em conjunto, têm o potencial de promover o aumento da quantidade de componentes disponíveis no repositório. Conforme mencionado por Frakes[30], a disponibilidade de uma grande quantidade de componentes tem o potencial de incrementar o reuso de componentes.

Embora a implementação do serviço de repositório já contemple importantes requisitos funcionais, outras facilidades devem ainda devem ser incluídas em versões futuras, tais como histórico de ações (*logs*), sistema de mensagens síncronas (*chat*), apoio ao processo de gerenciamento de projetos, bem como a especificação e implementação dos demais serviços que compõem o *ComponentForge*.

7. Referências

[1] Prikladnicki, R.; Marczak, S.; and Audy, J. L. (2006). “MuNDDoS: A Research Group on Global Software Development”. In International Conference on Global Software Engineering (ICGSE'06)

[2] Aoyama, M. (1998). “New age of software development: How component-based software engineering changes the way of software development”. In International Workshop on Component-Based Software Engineering(CBSE’98)..

[3] Szyperski, Clemens (2002). “Component Software: Beyond Object-Oriented Programming”. Second Edition, Addison-Wesley.

[4] Herbsleb, J.D., Grinter, R.(1999) “Splitting the organization and integrating the code: Conway's Law revisited”, In 21th International Conference on Software Engineering (ICSE’99).

[5] Wallnau, K. C.; Hissam, S. A.; Seacord, R. C.,(2001), "Building Systems from Commercial Components", SEI Series in Software Engineering, Addison-Wesley
[6] Crnkovic, Ivica. (2003) "Component-Based Software Engineering–New Challenges in Software Development". In Information Technology Interfaces (ITI'03).
[7] Kiel, L. (2003)."Experiences in Distributed Development: A Case Study", In: Workshop on Global Software Development at ICSE 2003", Oregon, EUA.
[8] Tommarello, J. D.; Deek, Fadi P. (2002). "Collaborative Software Development: A Discussion of Problem Solving Models and Groupware Technologies". In 35th Annual Hawaii International Conference on System Sciences (HICSS'02)
[9] Oliveira, João Paulo F.; Santos, Michael S.; Elias, Gledson. (2006). "ComponentForge:Um Framework Arquitetural para Desenvolvimento Distribuído Baseado em Componentes. VI WDBC. Recife – PE.
[10] Inoue, K.; *et al*. (2003) "Component Rank: Relative Significance Rank for Software Component Search". In 25th International Conference on Software Engineering (ICSE)
[11] Nutter, D.; Boldyreff, C.; Rank, S.(2003) "An Artefact Repository to Support Distributed Software Engineering". In: 2nd Workshop on Cooperative Supports for Distributed Software Engineering Processes.
[12] Ye, Yunwen.(2001) "Supporting Component-Based Software Development with Active Component Repository Systems". PhD thesis, University of Colorado.
[13] Component Source.(2007) http://www.componentsource.com.
[14] Laurillau, Yann; Nigay, Laurence. (2002). "Clover Architecture for Groupware". In Computer Supported Cooperative Work (CSCW'02), New Orleans, Louisiana.
[15] Ferraiolo, D. F.; *et al*.. (2001) "Proposed NIST Standard for Role-Based Access Control". ACM Transactions on Information and Systems Security, Vol. 4, No. 3
[16] Lau, K., (2001), "Component Certification and System Prediction: Is there a Role for Formality?". In 4th International Workshop on Component-Based Software Engineering (CBSE'01), Toronto, Canada.
[17] Schuenck, Michael. (2006) "X-ARM: Um Modelo de Representação de Artefatos de Software". Dissertação de Mestrado, DIMAp-UFRN, Natal-RN.
[18] Fielding, Roy Thomas. (2000). "Architectural Styles and the Design of Network-based Software Architectures". PhD thesis, University of. University of California.
[19] Gerosa, Marco Aurélio. (2006) "Desenvolvimento de groupware componentizado com base no modelo 3C de colaboração" – Tese (doutorado) – PUC – Rio de Janeiro.
[20] Grinter, R., (2001).From local to global coordination: lessons from software reuse. In: Proc. of the 2001 International ACM SIGGROUP, Colorado, USA.
[21] Dangelmaier, W., Hamoudia, H. and Klahold, R. (1999) "CIPD – On Workflow-Based Client Integration" in ACM SIGGROUP Bulletin, Vol 20, pag 20-25.
[22] Sun Microsystems. (2006) "Java™ Platform, Enterprise Edition 5" http://java.sun.com/javaee/5/docs/API
[23] JBoss Reference Guide(2007) http://docs.jboss.com/jbportal/v2.6/reference-guide/en/html/
[24] Grundy J., Wang X. and Hosking J. **"**Building multi-device, component-based, thin-client groupware: issues and experiences". In Australian Computer Science Communications. Australian Computer Society, Inc, Darlinghurst, Australia.(2002)
[25] Stal, M. (2002)."Web Services : Beyond Component Based Computing". Communications of the ACM, Vol. 45, Issue 110, pp 71-76.
[26] Hartman, Bret. Flinn , Donald J. Beznosov , Konstantin. Kawamoto , Shirley. (2003) "Mastering Web Services Security". Wiley Publishing Inc.
[27] OASIS Standard Specification. (2006). "Web Services Security". http://docs.oasis-open.org/wss/v1.1/
[28] MySQL 5.0. (2007). "Reference Manual". Copyright 1997-2007 MySQL AB.
[29] Greenberg, S. (2007) "Toolkits and Interface Creativity", Journal of Multimedia Tools and Applications, Volume 32 , Issue 2, pag 139-159.
[30] Frakes, William. Kang, Kyo.(2005). "Software Reuse Research: Status and Future". IEEE Transactions On Software Engineering, Vol.31, Nº 7, July.