Um Serviço de Repositório Compartilhado e Distribuído para Suporte ao Desenvolvimento Baseado em Componentes

João Paulo F. de Oliveira, Talles Brito, Sebastião Rabelo Jr, Glêdson Elias

Departamento de Informática
Universidade Federal da Paraíba (UFPB) – João Pessoa, PB – Brasil
{joaopaulo, talles, sebastiao, gledson}@compose.ufpb.br

Resumo. *Diversos repositórios de componentes têm sido propostos com o objetivo de potencializar o reuso de software. No entanto, as propostas atuais ainda adotam abordagens locais e centralizadas, que inibem o reuso em larga escala. Neste contexto, este artigo apresenta um serviço de repositório compartilhado e distribuído, que integra facilidades de controle de acesso, controle de versão e gerência de métricas de reuso. Como inovação, o repositório proposto pode ser explorado em abordagens de desenvolvimento distribuído, nas quais equipes remotas compartilham artefatos de software.*

Abstract. *Several component repository systems have been proposed for achieving the software reuse benefits. However, current proposals adopt local and centralized approaches, which inhibit large-scale reuse. In such a context, this paper presents a shared, distributed repository service that integrates facilities for access control, version control and reuse metrics management. As an advance, the proposed repository can be explored in distributed development approaches, in which independent teams share software artifacts.*

1. Introdução

Visando reduzir os custos de desenvolvimento, minimizar a complexidade e melhorar a qualidade dos sistemas de software, abordagens de desenvolvimento centradas no reuso têm sido propostas, nas quais artefatos são organizados de modo a compor novos artefatos, que, por sua vez, podem ser reusados em outros projetos. Neste contexto, o Desenvolvimento Baseado em Componentes (DBC) tem se destacado como uma abordagem promissora.

Entretanto, o reuso de componentes apresenta alguns fatores limitantes. Bass [1] identifica a carência de componentes, padrões, certificação e métodos para a construção de componentes como os principais fatores inibidores para o uso de componentes. Já para Crnkovic [2], destaca-se a atual dificuldade em identificar, selecionar, negociar e recuperar componentes que atendam aos requisitos especificados, e, além disso, que possuam alto grau de reusabilidade e qualidade.

Assim, para viabilizar as abordagens de DBC, a adoção de um serviço de repositório é, hoje, além de uma característica desejável, uma necessidade, pois segundo Guo [3], o repositório tem o propósito de auxiliar no reuso de componentes, e, assim, melhorar as metas de qualidade, custo e produtividade. Vale ressaltar ainda que, segundo Searcord [4], o repositório atua como um elo entre o desenvolvimento para reuso e o desenvolvimento com reuso de componentes, provendo meios para armazenamento, busca e recuperação dos mesmos.

Segundo Frakes [5], além de prover facilidades para armazenamento de componentes, os repositórios também devem disponibilizar interfaces para busca de componentes, adotar metadados para representação dos componentes, prover mecanismos para certificação de qualidade, incluir facilidades de controle de versão e métricas de reuso, como também, adotar mecanismos que viabilizem o retorno de investimento na negociação de componentes.

Neste contexto, vários sistemas de repositório de componentes foram propostos a fim de solucionar os problemas identificados [6][7][8][9][10][11][12]. Apesar das relevantes contribuições, as propostas atuais adotam, geralmente, abordagens locais e centralizadas, levando a problemas de acessibilidade, escalabilidade e disponibilidade limitada de componentes. Desta forma, as propostas atuais não favorecem o reuso em larga escala de componentes desenvolvidos por terceiros.

Com o objetivo de superar as limitações expostas, o grupo COMPOSE (*Component Oriented Service Engineering*) propôs o *ComponentForge* [13], um *framework* arquitetural que provê um conjunto de serviços para suporte a abordagens de DBC. Dentre os serviços do *framework*, este artigo apresenta o projeto, a especificação e a implementação do serviço de repositório, que integra facilidades de controle de acesso, controle de versões e gerência de métricas de reuso.

O serviço de repositório proposto é compartilhado, pois vários produtores podem registrar seus artefatos. Por outro lado, o serviço de repositório é 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. Desta forma, como inovação, o repositório proposto pode ser explorado em abordagens de desenvolvimento distribuído, nas quais equipes remotas e independentes colaboram e compartilham componentes, incrementando assim o reuso em larga escala.

O restante deste artigo está organizado da seguinte forma. A Seção 2 introduz o *framework* arquitetural *ComponentForge,* identificando seus serviços e as suas funcionalidades. A Seção 3 apresenta as principais características e funcionalidades do serviço de repositório. Em seguida, a Seção 4 apresenta o projeto arquitetural, as interfaces e a implementação de um protótipo do serviço de repositório. A Seção 5 apresenta um caso de uso em que as facilidades do serviço de repositório são exploradas por equipes distribuídas. A fim de ressaltar as contribuições do serviço de repositório proposto, a Seção 6 apresenta uma comparação com alguns trabalhos relacionados. Por fim, a Seção 7 apresenta algumas considerações finais e trabalhos futuros.

2. O ComponentForge

Considerando as limitações expostas, como já mencionado anteriormente, o grupo COMPOSE propôs o *ComponentForge* [13], um *framework* arquitetural que provê um conjunto de serviços que dão suporte a abordagens de DBC. Desta forma, esta seção tem por objetivo descrever sucintamente os elementos que compõem o *framework*, contextualizando assim o papel do serviço de repositório, que é o foco deste artigo.

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, a fim de prover facilidades para armazenar, indexar,

buscar, recuperar, certificar e negociar variados artefatos de software, tais como códigos executáveis, códigos fonte, especificações de interfaces, casos de uso, diagramas UML e planos de teste. Em conjunto, os serviços que compõem o *ComponentForge* apresentam baixo acoplamento, facilitando a customização, configuração, reuso, evolução e extensão, todos considerados importantes propriedades arquiteturais de aplicações distribuídas, conforme mencionado por Fielding [14].

Figura 1. Framework Arquitetural do ComponentForge

O conjunto de ferramentas permite a interação dos consumidores, produtores e gerentes dos serviços com os demais elementos do *ComponentForge*. Podem ser identificados três tipos de ferramentas: ferramentas de integração, usadas por consumidores; ferramentas de desenvolvimento, adotadas por produtores; e ferramentas de gerenciamento, utilizadas para administração dos serviços do *ComponentForge*.

A certificação de componentes, como mencionado anteriormente, é um fator crítico, pois muitos consumidores têm receio de confiar plenamente em artefatos produzidos por terceiros. O *ComponentForge* foi concebido de forma a permitir a coexistência de diversos serviços de certificação, 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.

De forma similar ao que acontece na produção de software em geral, produtores de componentes também podem estar interessados ou não em retorno financeiro. Neste sentido, o *ComponentForge* classifica os artefatos em dois tipos: *artefatos sem modelo de negócio*, que podem ser livremente distribuídos sem restrições; e *artefatos com modelo de negócio*, cujas formas de negociação são regidas por contratos especificados nos modelos de negócios, definidos por seus respectivos produtores. Portanto, artefatos podem ser adquiridos sob uma variedade de modelos de negócios.

Assim, o serviço de negociação tem a função de prover mecanismos para assegurar que um componente será adquirido e entregue em conformidade com os modelos de negócios especificados pelo produtor. Após 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.

Para permitir a busca e a recuperação de componentes armazenados, o *ComponentForge* define o serviço de busca. Este serviço consulta o serviço de repositório e recupera metadados para realizar a indexação. 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.

O serviço de repositório atua como um *middleware* de propósito especial, provendo meios para armazenar, localizar e recuperar componentes. O serviço de repositório é suportado por uma coleção de entidades cooperantes e distribuídas, denominadas *containers,* que, conjuntamente, provêem facilidades às demais entidades do *ComponentForge* através de interfaces bem definidas. Esta abordagem permite que implementações dos serviços de certificação, negociação e busca sejam independentes de plataforma e realizadas por distintas entidades.

Vale ressaltar que o *ComponentForge* explora um modelo de representação de componentes, denominado X-ARM [15]. Este modelo inclui informações que podem ser exploradas pelos consumidores na identificação, aquisição, instalação e uso de artefatos. Além disso, o X-ARM também representa informações sobre os modelos de certificação e de negócio adotados pelos produtores dos artefatos, viabilizando as funcionalidades providas pelos serviços de certificação, negociação e busca.

3. O Serviço de Repositório

O serviço de repositório é de fundamental importância, pois, além de prover facilidades para armazenamento e recuperação de artefatos aos demais serviços e ferramentas, internamente inclui facilidades de controle de versões e gerência de métricas de reuso, como também aspectos de segurança relacionados à autenticação, controle de acesso e visibilidade. A seguir, as facilidades providas pelo serviço de repositório são descritas de forma mais detalhada, e, posteriormente, a Seção 4 detalha o projeto arquitetural, a especificação de interfaces e a implementação do protótipo piloto.

3.1 Esquema de Nomeação

Considerando sua natureza compartilhada e distribuída, o serviço de repositório adota um esquema de nomeação que facilita a organização dos artefatos. Neste esquema, ilustrado na Figura 2, os nomes são organizados em uma árvore hierárquica onde as folhas são os artefatos armazenados. Por sua vez, os nós internos correspondem a dois tipos de entidades (*zonas* e *domínios*). Zonas representam produtores de artefatos e suas famílias de produtos. 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 ou produto.

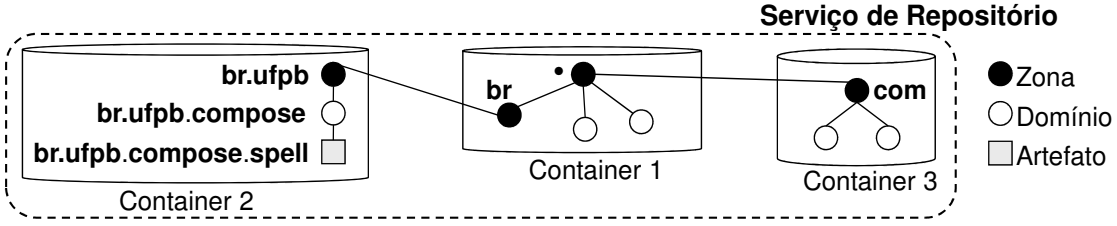

Figura 2. Esquema de Nomeação

Na composição dos identificadores hierárquicos, os nomes de zonas, domínios e artefatos são separados por pontos. Assim, a Figura 2 ilustra uma árvore de nomes hierárquicos em que *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*.

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.

3.2 Autenticação e Controle de Acesso

No serviço de repositório, existem diferentes usuários que desempenham diferentes papéis e executam tarefas distintas. O *gerente de container* realiza tarefas administrativas, registrando zonas autorizadas e verificando dados estatísticos relacionados ao *container*. Um *administrador de zona* gerencia uma determinada zona, criando e removendo domínios, bem como registrando, removendo e atualizando informações sobre desenvolvedores. Um *desenvolvedor* é responsável por registrar, atualizar e remover artefatos em sua respectiva zona. Já um *consumidor* pode recuperar artefatos armazenados.

Logo, considerando que os usuários possuem distintas funções, o serviço de repositório necessita de mecanismos de autenticação para certificar que tais usuários são realmente quem eles afirmam ser. Para prover autenticação, os *containers* adotam técnicas de assinatura digital. Neste contexto, os *containers* armazenam informações de segurança (chaves públicas) dos usuários, serviços autorizados e outros *containers*.

O controle de acesso no serviço de repositório adota um mecanismo semelhante ao modelo RBAC (*Role-Based Access Control*) [16], no qual permissões de acesso são associadas aos papéis, que, por sua vez, são atribuídos aos usuários. 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.

3.3 Visibilidade Externa

Para facilitar o compartilhamento de artefatos e controlar o acesso de diferentes equipes de desenvolvedores, o serviço de repositório explora o conceito de visibilidade. Diferentemente do controle de acesso, apresentado anteriormente, a visibilidade controla o acesso a artefatos de uma zona por usuários não cadastrados naquela zona. Ou seja, usuários que estão cadastrados em uma zona, possivelmente remota e diferente da zona a qual o artefato pertence. Assim, para cada artefato, podem ser definidos esquemas de visibilidade que indicam as outras zonas cujos desenvolvedores são autorizados ou proibidos de recuperar o artefato. Vale ressaltar que a visibilidade só pode ser modificada pelos desenvolvedores da zona a qual o artefato pertence.

Esse mecanismo 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. Vale ressaltar que, se um dado artefato não possui um esquema de visibilidade definido, então qualquer usuário pode recuperá-lo. No entanto, apenas os desenvolvedores permitidos no domínio onde o artefato está registrado podem executar operações que manipulam ou atualizam as informações do artefato. Ou seja, desenvolvedores de outras zonas não podem atualizar o artefato.

3.4 Controle de Versões

Segundo Holanda *et al* [17], é necessário um mecanismo que controle as diversas versões de um mesmo artefato, estabelecendo o relacionamento entre as versões e um controle das modificações de cada versão. O serviço de repositório oferece esse tipo de suporte durante o processo de desenvolvimento, permitindo, assim, que os desenvolvedores pertencentes a um mesmo grupo de trabalho e distribuídos geograficamente, participem da produção de um dado artefato. O esquema adotado é semelhante aos adotados em outros sistemas de controle de versão. Todavia, a diferença fundamental é que o esquema adotado no serviço de repositório não gerencia apenas código fonte, mas qualquer tipo de artefato produzido no processo de desenvolvimento.

Considerando as diversas versões dos artefatos, o serviço de repositório permite a coexistência de diversas linhas de evolução (*branches*) para cada artefato. Diferentes versões do artefato podem ser simultaneamente manipuladas pelos desenvolvedores. Por exemplo, como mostrado na Figura 3, a versão *2.0* representa uma versão inicial que cria uma linha de evolução (*branch)*. A partir da versão inicial de um *branch*, novas versões podem ser desenvolvidas. Com isso, a partir da versão *2.0*, novas versões seguem a seqüência *2.1*, *2.2*, *2.3* e assim por diante. A qualquer momento, o desenvolvedor pode criar um novo *branch*, que pode ser baseado ou não em um *branch* já existente. Na Figura 3, a versão *2.3* do *branch 2* dá origem aos *branches 3 e 2.3*.

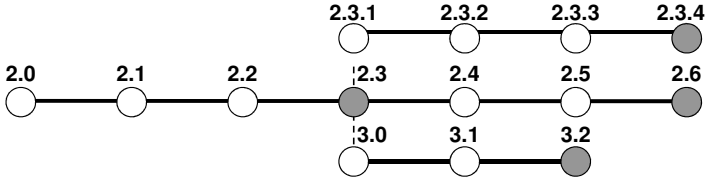

Figura 3. Linhas de Evolução no Processo de Desenvolvimento

Além disso, no serviço de repositório, é possível que um conjunto de desenvolvedores trabalhe em uma determinada versão de um dado artefato sem que seja preciso torná-la pública aos consumidores. Isto é controlado através da marcação do artefato como *registrado* ou *publicado*. Um artefato no estado registrado somente pode ser recuperado pelos desenvolvedores, e pelos serviços de busca e certificação autorizados. Já artefatos no estado publicado também podem ser recuperados por consumidores, desde que respeitem o modelo de negócio do artefato. Na Figura 3, os círculos brancos representam artefatos registrados, enquanto os círculos cinza representam artefatos publicados.

3.5 Informações de Reuso

Com a finalidade de facilitar a tomada de decisão dos consumidores sobre a adoção e uso de um dado artefato, o serviço de repositório gerencia um conjunto de *informações de reuso*. Para um dado artefato, suas informações de reuso identificam: o grau de satisfação dos consumidores que já o adquiriram; os comentários destes consumidores sobre o seu uso; as áreas de aplicação nas quais foi usado; além das respostas dos desenvolvedores do artefato aos comentários dos consumidores.

É importante ressaltar que as informações de reuso ficam à disposição de futuros consumidores, que, segundo Preiss [18], é de fundamental importância, pois permite aos consumidores se valerem das experiências relatadas por outros consumidores para ajudar no processo de escolha de um determinado artefato.

4. Projeto, Especificação e Implementação do Serviço de Repositório

Considerando a gama de funcionalidades oferecidas, o projeto arquitetural do serviço de repositório também adota uma abordagem orientada a serviços. A seguir, descreveremos a arquitetura definida para o serviço de repositório, juntamente com a função e delimitação de cada um de seus serviços internos. Em seguida, identificamos as funcionalidades providas pelas interfaces destes serviços internos. Por fim, apresentamos uma visão geral da implementação do protótipo piloto.

4.1 Projeto Arquitetural e Especificação de Interfaces

Como mencionado anteriormente, o serviço de repositório é composto por um conjunto de entidades, denominadas *containers*. Como ilustrado na Figura 4, cada *container* possui um projeto arquitetural organizado em três camadas: a *camada de acesso* trata as questões funcionais do serviço de repositório, incluindo, por exemplo, facilidades para armazenamento e recuperação de artefatos; a *camada de distribuição* trata as questões não funcionais, especificamente aspectos relacionados à distribuição e segurança; e a *camada de armazenamento* é responsável por gerenciar a persistência dos artefatos.

A seguir, as funcionalidades de cada camada e dos seus dos seus respectivos serviços internos são identificadas, mostrando sucintamente quais interfaces destes serviços internos são responsáveis por prover tais funcionalidades. Vale ressaltar que, em conjunto, as interfaces dos serviços internos provêem mais de 800 operações. Logo, por questão de espaço, não é possível mencionar e detalhar cada uma destas operações.

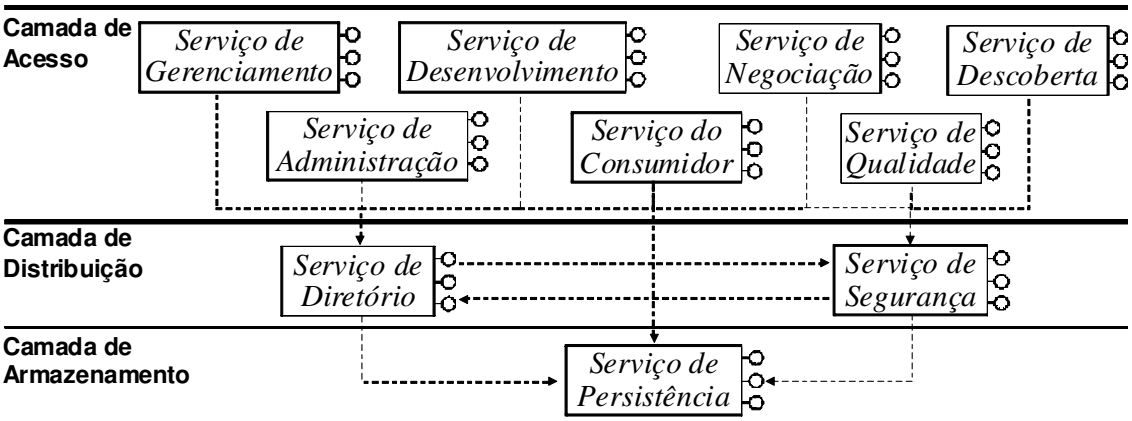

Figura 4. Arquitetura em Camadas e Serviços do Container

4.1.1 Camada de Acesso

Como mostrado na Figura 4, a camada de acesso é composta por um conjunto de serviços que viabilizam a interação das ferramentas e dos outros serviços do *ComponentForge* (serviços de certificação, negociação e busca) com o *container*, e, assim, com o serviço de repositório como um todo. Em geral, como será descrito a seguir, cada serviço da camada de acesso é utilizado por um diferente tipo de usuário do *container* ou tipo de serviço do *ComponentForge*. Também é importante destacar que todos os serviços da camada de acesso exploram os mecanismos de autenticação e controle de acesso para assegurar que somente entidades autorizadas podem acessá-los.

Serviço de gerenciamento. O serviço de gerenciamento é utilizado pelo gerente do *container* para gerenciar as zonas autorizadas no *container*. Este serviço provê um conjunto de quatro interfaces. A interface *IManageZone* permite cadastrar as zonas

autorizadas do *container*. Usando operações definidas na interface *IRootContainer*, é possível registrar a zona raiz do esquema de nomeação hierárquico, que requer operações especiais, pois é de fundamental importância no processo de resolução de nomes. A interface *IContainerStatistics* permite consultar informações estatísticas de uso do *container*, como por exemplo, o espaço ocupado pelas zonas. Por fim, as chaves públicas do *container* e do próprio gerente do *container*, que são usadas para autenticação, podem ser cadastradas e alteradas ativando operações da interface *IManageContainer*.

Serviço de administração. O serviço de administração permite ao administrador de zona configurar sua zona e seus respectivos domínios. As funcionalidades deste serviço são providas por um conjunto de sete interfaces. A interface *IManageZone* permite configurar e recuperar informações da zona e estruturar os seus domínios. As operações definidas na interface *IManageUser* permitem cadastrar novos usuários, tais como outros administradores e desenvolvedores, e configurar suas permissões de acesso na zona. Também é possível consultar informações estatísticas da zona através da interface *IZoneStatistics*. Para facilitar a estruturação da árvore hierárquica de nomes, a interface *ITransformNode* permite transformar zona em domínio e vice-versa. Além disso, as operações da interface *IManageHierarchicalTree* permitem associar uma zona à sua zona pai, e, assim, viabilizar o processo de resolução de nomes. Por fim, para controlar o acesso dos serviços de alto nível do *ComponentForge* (serviços de certificação, negociação e busca), as interfaces *IManageService* e *IManageServicePermission* permitem, respectivamente, cadastrar os serviços providos pela zona e configurar as permissões de acesso dos serviços providos por outras zonas.

Serviço de desenvolvimento. O serviço de desenvolvimento provê um conjunto de dez interfaces aos desenvolvedores. Em uma dada zona, cada desenvolvedor possui um conjunto de informações, incluindo sua chave pública e as permissões de acesso que definem quais operações podem ser ativadas pelo mesmo. A interface *IUpdateDeveloper* possui operações para atualizar as informações e recuperar as permissões de acesso dos desenvolvedores.

O desenvolvedor é responsável pelo registro das diversas versões dos artefatos no repositório. Para tal, o serviço de desenvolvimento provê as interfaces *IManageAsset* e *IConfigurationMgmt*, cujas operações viabilizam o registro, a remoção e o controle de versão dos artefatos, permitindo gerenciar linhas de evolução (*branches*), alterar os estados (registrado/publicado) e controlar o bloqueio de artefatos. Para controlar o acesso de desenvolvedores de outras zonas, o desenvolvedor pode gerenciar o esquema de visibilidade com operações providas pela interface *IAssetVisibility*.

Como mencionado anteriormente, o serviço de repositório adota o X-ARM para representação dos artefatos. No serviço de desenvolvimento, a interface *IAssetMetaModel* permite recuperar as informações da descrição X-ARM dos artefatos. No X-ARM, artefatos são hierarquicamente classificados em domínios de aplicação. A interface *IAssetClassification* provê operações para recuperar e atualizar a classificação dos artefatos. O X-ARM dispõe de mecanismos para representar diferentes modelos de certificação e negócio. Para possibilitar o registro e a atualização de certificados de qualidade e regras de negociação dos artefatos, o serviço de desenvolvimento provê as interfaces *IAssetCertification* e *IAssetBusinessModel*, respectivamente.

Considerando as informações de reuso dos artefatos, os desenvolvedores podem controlar o número de comentários do consumidor, como também, apresentar respostas aos comentários. Para tal, o serviço de desenvolvimento provê a interface *IAssetReuseInformation*. Por fim, o desenvolvedor pode obter, via a interface *IAssetStatistics*, informações estatísticas dos artefatos aos quais tem acesso, como por exemplo, o número de vezes que o artefato foi adquirido.

Serviço do consumidor. O serviço do consumidor provê mecanismos para recuperação de artefatos que não adotam modelos de negócios, e, portanto, podem ser distribuídos livremente. Assim, este serviço não adota mecanismos de autenticação e o único controle de acesso está relacionado a assegurar que os artefatos recuperados não adotam modelos de negócios e não possuem esquemas de visibilidade definidos.

O serviço do consumidor possui seis interfaces. A recuperação de artefatos e de suas respectivas descrições X-ARM é realizada através das interfaces *IRetrieveAsset* e *IRetrieveMetaModel*, respectivamente. Vale ressaltar que os consumidores também podem recuperar descrições X-ARM de artefatos com modelos de negócio, possibilitando aos mesmos identificar as formas de negociação dos artefatos. Para que os consumidores possam identificar o histórico de evolução dos artefatos e suas diferentes versões, o serviço do consumidor disponibiliza a interface *IAssetHistory*.

No serviço do consumidor, as informações de reuso, incluindo o grau de satisfação e os comentários do consumidor, podem ser registradas e recuperadas usando a interface *IAssetReuseInformation*. Para que os consumidores possam descobrir quais são os serviços de certificação, negociação e busca autorizados a certificar, negociar e indexar uma determinada zona, domínio ou artefato, o serviço do consumidor oferece a interface *IServiceInformation*. Por fim, os consumidores podem obter, via a interface *IAssetStatistics*, informações estatísticas dos artefatos, como por exemplo, o número de vezes que o artefato foi adquirido.

Serviço de negócio. O serviço de negócio é utilizado pelo serviço de negociação do *ComponentForge* para recuperar artefatos com modelos de negócio. É importante ressaltar que o serviço de negócio não é responsável por aplicar as regras de negociação do artefato. Tal responsabilidade é atribuída ao serviço de negociação do *ComponentForge*. Assim, quando um consumidor deseja recuperar um artefato com modelo de negócio, o consumidor deve adquiri-lo via o serviço de negociação, que, por sua vez, recupera o artefato através do serviço de negócio, que provê duas interfaces. A interface *IRetrieveNegotiatedAsset* contém operações para recuperar artefatos com modelos de negócio. Por outro lado, a interface *IRetrieveBusinessModel* possibilita a identificação e recuperação dos modelos de negócio adotados por cada artefato.

Serviço de qualidade. O serviço de qualidade é acessado pelo serviço de certificação do *ComponentForge* para registrar e recuperar certificados dos artefatos. O serviço de qualidade não é responsável por aplicar o modelo de certificação adotado nos certificados. Esta responsabilidade é do serviço de certificação. Assim, quando uma entidade certificadora deseja registrar ou recuperar um certificado de um dado artefato, esta entidade interage com o serviço de qualidade através da única interface provida, denominada *IManageCertification*.

Serviço de descoberta. O serviço de descoberta é usado pelo serviço de busca do *ComponentForge* para navegar na hierarquia de zonas e domínios, e, em paralelo, recuperar as descrições X-ARM dos artefatos ou os próprios artefatos para indexação. O

serviço de descoberta possui três interfaces. A interface *IHierarchicalTree* viabiliza a navegação na hierarquia de zonas e domínios. Já a interface *IAssetInformation* é usada para recuperar as descrições X-ARM dos artefatos ou os próprios artefatos. Por fim, a interface *IRetrieveStatistics* permite obter informações estatísticas dos artefatos, como por exemplo, o número de vezes que foi adquirido.

4.1.2 Camada de Distribuição

A camada de distribuição trata aspectos não funcionais do serviço de repositório relacionados à distribuição e segurança, tornando possível a descoberta transparente da localização dos artefatos e a recuperação segura dos mesmos. Estes aspectos não funcionais são tratados pelos dois serviços da camada de distribuição: o serviço de diretório e o serviço de segurança.

Serviço de segurança. Este serviço é utilizado pelos serviços da camada de acesso para autenticar e controlar o acesso das entidades que se comunicam com o *container*. Para tal, o serviço de segurança provê cinco interfaces. A interface *IKeyMgt* possui operações para cadastrar e atualizar as chaves públicas das entidades que se comunicam com o *container*. Estas chaves são utilizadas pelas operações da interface *ICipherMgt* para assinar e cifrar mensagens trocadas com o *container*. As interfaces *IUserMgt* e *IServiceMgt* são usadas para realizar o controle de acesso de usuários (gerente do *container*, administradores de zona e desenvolvedores) e serviços (certificação, negociação e busca) autorizados, respectivamente. Por fim, a interface *IAssetVisibilityMgt* é usada para controlar a visibilidade dos artefatos.

Serviço de diretório. O serviço de diretório é responsável pelo processo de resolução de nomes, que traduz nomes de artefatos para os identificadores dos respectivos *containers* em que estão armazenados. Para desempenhar tais funções, este serviço define quatro interfaces. Usando operações da interface *IContainerMgt* é possível manipular informações sobre o *container* que mantém a zona raiz do esquema de nomeação hierárquico, que é de fundamental importância no processo de resolução de nomes. As interfaces *IZoneMgt*, *IDomainMgt* e *IServiceMgt* estão associadas ao gerenciamento da hierarquia de nomes, permitindo o registro, remoção e atualização de zonas, domínios e serviços, respectivamente. Além disso, a interface *IZoneMgt* também possui as operações utilizadas no processo de resolução de nomes.

4.1.3 Camada de Armazenamento

A camada de armazenamento do serviço de repositório constitui-se de um único serviço, o serviço de persistência, que disponibiliza aos demais serviços das camadas de acesso e distribuição facilidades para armazenar e recuperar dados manipulados por tais serviços. Este serviço provê independência dos demais serviços em relação ao sistema de armazenamento adotado (SGBD ou sistema de arquivos).

Serviço de persistência. Este serviço possui sete interfaces. A interface *IUserStg* possibilita a manipulação de informações dos administradores de zona e desenvolvedores cadastrados no *container*, incluindo a configuração de seus papéis. As chaves públicas e privadas são manipuladas através da interface *IKeyStg*, enquanto as informações de controle de acesso e visibilidade são tratadas pela interface *IPermissionStg*. Existem ainda a interface *ITreeStg* para gerenciar informações das zonas, domínios e serviços, e também a interface *IAssetStg* para manusear artefatos e suas informações. A interface *IStorageStg* provê algumas operações para configurar

limites de parâmetros do sistema, como por exemplo, o limite máximo de espaço para zonas e o número de comentários por consumidor. Por fim, na interface *IStatisticsStg*, operações são definidas para recuperação das estatísticas de zonas, domínios e artefatos.

4.2 Implementação do Protótipo Piloto

Com o objetivo de concretizar a arquitetura proposta, o protótipo foi implementado utilizando componentes *Enterprise JavaBeans* [19] e *Web Services* [20], executando em um servidor de aplicações *JBoss Application Server* [21]. Vale ressaltar que a implementação do protótipo possui cerca de 57.000 linhas de código, distribuídas em 450 classes e 40 pacotes Java. A Figura 5 apresenta uma visão geral da implementação do serviço de repositório, identificando as diversas tecnologias adotadas. Em conjunto, tais tecnologias atendem satisfatoriamente as necessidades do serviço de repositório, e, nos testes realizados, têm se mostrado adequadas em relação a requisitos como manutenibilidade, confiabilidade e escalabilidade.

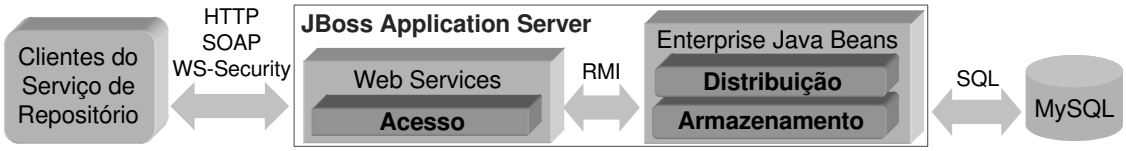

Figura 5. Visão Geral da Implementação

A camada de acesso foi implementada usando *stateless beans,* pois os serviços desta camada não precisam manter informações de estado dos clientes. Todas as interfaces da camada de acesso são expostas como *Web Services*, proporcionando assim interoperabilidade, pois permite que instâncias implementadas em diferentes plataformas e adotando distintas linguagens de programação possam se comunicar simplesmente expondo suas interfaces. Além disso, a tecnologia *Web Services* favorece o acesso ao serviço de repositório de diferentes maneiras, podendo incluir, por exemplo, interfaces com o usuário baseadas em navegadores *Web* ou aplicações *desktop*.

A camada de acesso coordena o fluxo de execução de chamadas às operações das camadas inferiores. Neste caso, dados podem ser recuperados da camada de distribuição e da camada de armazenamento. Os dados da camada de distribuição são mantidos em memória e os dados da camada de armazenamento são mantidos em meios persistentes.

A camada de distribuição também foi implementada utilizando *stateless beans.* Neste caso, um tipo especial de *bean*, chamado *MBean*, foi usado. Os *MBeans* representam um recurso *JMX* (*Java Management Extensions*) [22], que é gerenciável em tempo de execução. No caso do serviço de repositório, os *MBeans* mantêm em memória dados importantes para os serviços de diretório e de segurança, de forma a tornar a manipulação desses dados mais eficiente. *MBeans* são *beans* providos como objetos *singleton*, que asseguram a existência de uma única instância do mesmo em cada máquina virtual. Além disso, toda questão de sincronização e transações sobre esses dados é gerenciada pelo próprio *JBoss Application Server*. Assim, a resolução de nomes executada pelo serviço de diretório, bem como os processos de verificação de assinaturas e autenticação dos usuários executados pelo serviço de segurança, têm melhor desempenho e consistência.

Para realizar o controle de acesso, o serviço de repositório deve suportar mecanismos de autenticação dos usuários e integridade das informações. Para lidar com

tais questões, o serviço de segurança utiliza a *API JBossWS*, que é uma implementação da especificação *WS-Security* [23]. Esta especificação define um modelo que abstrai as facilidades de segurança, separando as funcionalidades de segurança do sistema de sua implementação específica. Desta forma, a especificação *WS-Security* trata vários aspectos de segurança, tais como criptografia e assinaturas digitais, que podem ocorrer na comunicação baseada em mensagens *SOAP* entre os *containers* e os clientes do repositório. A criação e verificação de assinaturas digitais, bem como a cifragem das mensagens, adotam certificados digitais X.509 com o algoritmo RSA.

Por fim, a camada de armazenamento foi implementada utilizando *entity beans*, juntamente com um sistema de gerenciamento de banco de dados (SGBD). Como esta camada manipula e armazena uma grande quantidade de dados com acessos simultâneos e em larga escala, se faz necessário um SGBD robusto e rápido. Para tal, utilizamos o *MySQL* [24], que, nos testes realizados, tem se mostrado adequado às necessidades.

5. Um Caso de Uso

Para facilitar e consolidar o entendimento das funcionalidades do serviço de repositório, esta seção descreve um caso de uso no qual o repositório é adotado por três equipes distribuídas (*A*, *B* e *C*) com o objetivo de compartilhar e reusar um conjunto de artefatos, desenvolvidos durante um projeto de software.

No caso de uso, devido à simplicidade e facilidade de uso, consideramos que as equipes adotam o processo *UML Components* [25]. É importante destacar que, para simplificar o caso de uso, consideramos que as equipes estão envolvidas apenas nas etapas *requisitos* e *especificação* do processo.

Cada equipe tem responsabilidades distintas dentro das etapas *requisitos* e *especificação* do processo de desenvolvimento, manipulando e gerando diferentes tipos de artefatos. Vale ressaltar que o *UML Components* divide a etapa *especificação* em três estágios: *identificação*, *interação* e *especificação de componentes*. A equipe *A* é responsável pelas atividades da etapa *requisitos* e do estágio *identificação de componentes*. A equipe *B* é responsável pelo estágio *interação de componentes*, e, por fim, a equipe *C* assume a responsabilidade pelo estágio *especificação de componentes*.

Embora todas as equipes estejam geograficamente distribuídas, as equipes *A* e *B* manipulam artefatos armazenados em um mesmo *container*. Já a equipe *C* manipula artefatos armazenados em um outro *container*. Além disso, as equipes *A* e *C* usam localmente os seus *containers*, pois os mesmos estão fisicamente no local de trabalho destas equipes. Por outro lado, embora use o mesmo *container* da equipe *A*, a equipe *B* usa este *container* remotamente.

A Figura 6 mostra a organização hierárquica e a localização das zonas nos vários *containers* considerados no caso de uso. Como pode ser observado, em cada *container* cria-se zonas para cada equipe. Para tal, inicialmente, utilizando operações da interface *IManageZone* do serviço de gerenciamento, os gerentes dos *containers* autorizam a existência das zonas. Em seguida, os administradores das zonas devem cadastrar suas zonas nas respectivas zonas pai, permitindo que a resolução de nomes possa ser realizada corretamente. Para cadastrar na zona pai, os administradores de zona devem utilizar operações da interface *IManageHierarchicalTree* do serviço de administração.

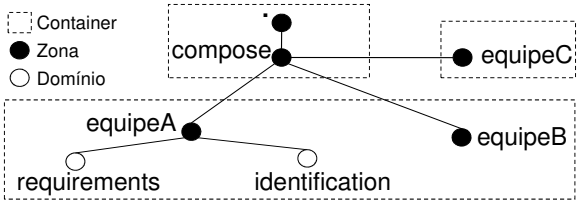

Figura 6: Hierarquia de Nomes

Também é possível observar na Figura 6 que a equipe *A* agrupa artefatos da etapa *requisitos* e do estágio *identificação de componentes* em diferentes domínios (*requirements* e *identification*, respectivamente). Para registrar estes domínios, o administrador utiliza operações da interface *IManageZone* do serviço de administração.

Após configurar as zonas, ativando operações da interface *IManageUser* do serviço de administração, os administradores de zona devem cadastrar os desenvolvedores de cada equipe. Neste caso de uso, assumimos que a equipe *A* é dividida em dois times de desenvolvedores (*A.1* e *A.2*), que atuam separadamente na etapa *especificação* e no estágio *identificação de componentes*. Na equipe *A*, para controlar quais artefatos gerados por um time de desenvolvedores podem ser acessados pelo outro time, o administrador de zona deve utilizar operações da interface *IManageUser* do serviço de administração para definir papéis, atribuir permissões e associar os desenvolvedores aos seus respectivos papéis.

A Figura 7 ilustra a configuração de papéis e permissões de acesso da equipe *A*. Nesta configuração, o administrador de zona define dois papéis: *requerimentsRole* e *identificationRole*. Em seguida, o administrador deve definir as permissões de acesso, que são especificadas na forma de operações permitidas por domínio.

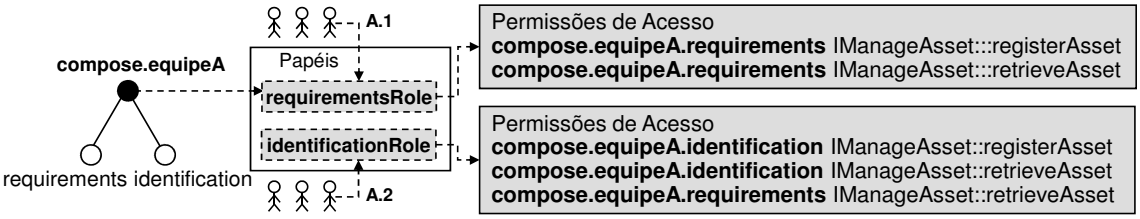

Figura 7. Permissões de Acesso

Na Figura 7, pode-se observar que os desenvolvedores associados aos papéis *requerimentsRole* e *identificationRole* podem armazenar e recuperar seus artefatos nos domínios *requirements* e *identification*, respectivamente. Além disso, considerando que, no *UML Components*, os artefatos gerados na etapa *especificação* são reusados posteriormente no estágio *identificação de componentes*, pode-se observar que o time alocado ao papel *identificationRole* pode recuperar, via a operação *retrieveAsset*, os artefatos associados ao domínio *requirements*, que são produzidos pelo time alocado ao papel *requirementsRole*. Por fim, após criar os papéis e atribuir as permissões de acesso, o administrador de zona deve associar os desenvolvedores aos seus respectivos papéis (Figura 7). Para as equipes *B* e *C*, a configuração de papéis e permissões de acesso também deve ser realizada, mas, por limitação de espaço, não é mostrada na Figura 7.

Uma vez configurado as zonas e domínios, como também os papéis e permissões, os desenvolvedores podem seguir as atividades definidas no *UML Components*, produzindo, reusando e registrando no repositório os seus respectivos

artefatos. Para registrar e reusar os artefatos, os desenvolvedores devem ativar operações da interface *IManageAsset* do serviço de desenvolvimento.

Vale ressaltar que o esquema de permissões de acesso não controla o acesso para equipes alocadas a diferentes zonas. Uma vez que as equipes *A*, *B* e *C* estão alocadas a zonas diferentes e precisam reusar artefatos entre si, os desenvolvedores também devem configurar esquemas de visibilidade para os artefatos. Para tal, devem usar operações da interface *IAssetVisibility* do serviço de desenvolvimento.

A Figura 8 mostra a configuração de visibilidade. Nesta configuração, em conformidade com o *UML Components*, a equipe *C*, alocada ao estágio *especificação de componentes*, tem acesso ao artefato *modelo de tipo de negócio* (*btm-1.0*) gerado pelo time *A.1*, alocado à etapa *requisitos*. Já a equipe *B*, alocada ao estágio *interação de componentes*, tem acesso ao artefato *interfaces de negócio* (*businessinterface-1.0*) gerado pelo time *A.2*, alocado ao estágio *identificação de componentes*.

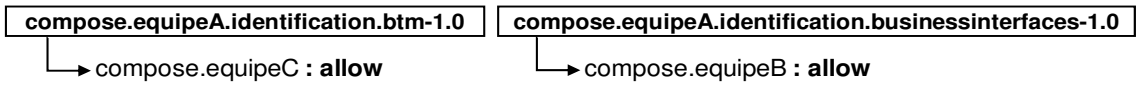

Figura 8. Esquemas de Visibilidade

Por fim, vale ressaltar que o *UML Components* é um processo iterativo, ou seja, ao fim de cada iteração as equipes podem avaliar seus resultados e, sempre que necessário, registrar novas versões dos seus artefatos. Por exemplo, caso fossem identificados novos requisitos durante o processo de desenvolvimento, a equipe *A* poderia gerar e registrar novas versões do artefato *casos de uso* usando as operações das interfaces *IManageAsset* e *IConfigurationMgmt* do serviço de desenvolvimento.

6. Trabalhos Relacionados

Esta seção apresenta uma breve comparação das funcionalidades providas pelo serviço de repositório proposto, com os principais repositórios disponíveis na literatura e mercado. Para facilitar a análise, somente consideramos repositórios que provêem meios para armazenar, buscar e recuperar componentes: *SPARS-J* [6], *CodeBroker* [7], *ComponentSource* [8], *SourceForge* [9], *Xtras.Net* [10], *CompoNex* [11] e *OSCAR* [12].

Em um mercado global de software, no qual consumidores e produtores estão geograficamente dispersos, requisitos de distribuição e escalabilidade são fundamentais para potencializar a quantidade de componentes disponíveis. Entretanto, a maioria dos repositórios existentes não trata este requisito, apesar da sua importância. Além do repositório proposto, somente o *OSCAR* provê uma arquitetura distribuída, porém restrito à instituição que o utilizar, limitando assim a quantidade de componentes disponíveis. É importante ressaltar que, apesar do *SourceForge* prover suporte a equipes distribuídas, assim como os demais repositórios, sua arquitetura é centralizada.

Como indicado por Frakes [5], além de componentes, repositórios devem conter metadados que os descrevam. Nesse sentido, o *CompoNex* provê diferentes informações dos componentes, tais como, preço, domínios de aplicação, modelo de componente, funcionalidades e implementação. O *OSCAR* e o *ComponentSource* adotam metadados que, além de descrever as funcionalidades dos artefatos, apenas incluem informações sobre mudanças em relação a versões anteriores e compatibilidade. Além disso, o *ComponentSource* também inclui informações sobre os modelos de negócios. O *CodeBroker* não adota nenhum metadado, mas indexa os documentos HTML gerados

pela ferramenta *Javadoc*, a fim de obter informações sobre os componentes. Já o *SPARS-J, Xtras.Net, SourceForge* não adotam nenhum metadado.

De forma mais abrangente, o serviço de repositório proposto adota o X-ARM como modelo de descrição dos artefatos. O X-ARM é explorado pelos consumidores na identificação, aquisição, instalação e uso dos artefatos. Além disso, o X-ARM também representa os modelos de certificação e de negócio adotados pelos produtores, viabilizando as funcionalidades providas pelos serviços de certificação, negociação e busca. Vale ressaltar que, além de descrever as funcionalidades dos artefatos, o X-ARM também inclui informações sobre o processo de desenvolvimento adotado, histórico de evolução, modelos de componentes e classificação em domínios de aplicação.

Como diversos usuários podem compartilhar artefatos, mecanismos de controle de versões são de fundamental importância. Nos repositórios analisados, somente o *OSCAR* e o *SourceForge* possuem mecanismos para controle de versões. No entanto, ambos adotam abordagens baseadas na integração de soluções de terceiros. Por outro lado, o serviço de repositório proposto define mecanismos próprios de controle de versões, capazes de controlar de forma customizada as diversas linhas de evolução dos diferentes tipos de artefatos, ou seja, não apenas código fonte.

Além do controle de versão, também é preciso controlar os usuários e as entidades autorizadas a recuperar e manipular artefatos. O *ComponentSource*, *CompoNex* e *Xtras.Net* apenas requerem que usuários e empresas estejam cadastrados, fazendo assim um intercâmbio entre os consumidores e produtores de componentes. O *OSCAR* adota um simples esquema de *login* e senha, além de histórico (*log*), que armazena quem acessou, qual artefato foi acessado e qual foi o propósito do acesso. Já o *CodeBroker* e o *SPARS-J* não adotam nenhum controle de acesso.

O serviço de repositório proposto, assim como o *SourceForge*, adota o modelo de controle de acesso denominado RBAC (*Role-Based Access Control*), no qual entidades são classificadas em diferentes papéis e diferentes permissões de acesso são associadas a cada papel desempenhado pelas entidades. Deste modo, de forma simples e de fácil manutenção, é possível controlar quais operações as equipes distribuídas de desenvolvedores podem executar durante o processo de desenvolvimento.

7. Considerações Finais

No DBC, para tornar realidade o reuso de software em larga escala, não basta que os componentes existam. É necessário que estejam amplamente disponíveis e acessíveis, permitindo incrementar a produtividade, reduzir os custos e melhorar a qualidade. Neste contexto, vários repositórios de componentes têm sido propostos [6][7][8][9][10][11][12]. No entanto, apesar das contribuições e avanços, as propostas atuais ainda apresentam deficiências, que, em conjunto, são consideradas inibidores do reuso em larga escala. Com o objetivo de superar as limitações das propostas existentes, este artigo propôs um serviço de repositório compartilhado e distribuído, apresentando seu projeto arquitetural e especificação de interfaces, bem como descrevendo características do protótipo piloto desenvolvido. Além disso, o artigo também descreveu um caso de uso, no qual equipes distribuídas adotam o repositório com o objetivo de compartilhar e reusar artefatos, desenvolvidos na fase inicial de levantamento de requisitos e especificação de componentes de um projeto de software.

Como importante contribuição, o serviço repositório proposto pode ser explorado em abordagens de desenvolvimento distribuído. Nestas abordagens, equipes de desenvolvedores, em localidades remotas e independentes, colaboram e compartilham componentes, incrementando assim o reuso em larga escala, pois, conjuntamente, o compartilhamento e a distribuição têm o potencial de incrementar a quantidade de componentes disponibilizados. Outra importante contribuição do repositório proposto é a integração simultânea de diversas facilidades que, em geral, são identificadas e encontradas nas propostas existentes de forma isolada e parcial. Exemplos destas facilidades incluem: segurança (autenticação, privacidade, integridade e controle de acesso), controle de visibilidade de artefatos, controle de versões e gerência de métricas de reuso. Além disso, o serviço proposto explora um modelo de descrição de artefatos, denominado X-ARM, que permite a adoção de variados modelos de certificação e negociação de artefatos.

Em conclusão, por adotar uma arquitetura orientada a serviços (SOA), o repositório proposto tem o potencial de incrementar o grau de manutenibilidade, extensibilidade, interoperabilidade, acessibilidade, disponibilidade, escalabilidade e reusabilidade. A atribuição de responsabilidades bem definidas aos vários serviços internos confere facilidades de manutenção, evolução e extensão, tornando as implementações bastante flexíveis. Por natureza, SOA agrega facilidades de interoperação, permitindo que diferentes implementações possam se comunicar. A arquitetura distribuída automaticamente particiona o conjunto de artefatos em diferentes *containers*. Desta forma, os artefatos tornam-se mais acessíveis e disponíveis, uma vez que falhas em um *container* não inviabilizam o serviço como um todo. É importante também destacar que a adoção dos *containers* distribuídos incrementa a escalabilidade da arquitetura, em suas várias dimensões (populacional, geográfica e administrativa), possibilitando a manipulação de um grande volume de artefatos, armazenados em localidades fisicamente remotas e distantes, e cujo esforço de administração é fracionado entre os diversos atores. Por fim, por ser compartilhado, o repositório proposto atua como indutor do reuso, pois vários produtores podem cooperar, colaborar e compartilhar artefatos durante o processo de desenvolvido.

Embora o serviço de repositório proposto já contemple importantes requisitos funcionais, outras facilidades devem ser incluídas em versões futuras, tais como: histórico de ações (*logs*), sistema de mensagens síncronas (*chat*), bem como o projeto e a especificação dos demais serviços e ferramentas que compõem o *ComponentForge*.

Referências Bibliográficas

[1] Bass, L.; Buhman, C.; Dorda, S.; Long, F; Robert, J.; Seacord, R.; Wallnau, K. (2000) "Market Assessment of Component-Based Software Engineering". SEI, Technical Report CMU/SEI-2001-TN-007.

[2] Crnkovic, I. (2003) "Component-Based Software Engineering–New Challenges in Software Development". Information Technology Interfaces, pp. 127-133, Croatia.

[3] Guo, J.; Luqi. (2000). "A Survey of Software Reuse Repositories". 7th IEEE International Conference and Workshop on the Engineering of Computer Based Systems. pp. 92-100.

[4] Seacord, R. (1999) "Software Engineering Component Repository". International Workshop on the Engineering of Computer Based Systems, Los Angeles.

[5] Frakes, W.; Kang, K. (2005) "Software Reuse Research: Status and Future". IEEE Transactions on Software Engineering, Vol.31, Nº 7, July.

[6] Inoue, K.; Yokomori, R.; Fujiwara, H.; Yamamoto, T.; Matsushita, M.; Kusumoto, S. (2003) "Component Rank: Relative Significance Rank for Software Component Search". International Conference on Software Engineering.

[7] Ye, Y. (2001) "Supporting Component-Based Software Development with Active Component Repository Systems". PhD Thesis, University of Colorado.

[8] Component Source (2007). http://www.componentsource.com.

[9] SourceForge Enterprise Edition (2007). http://www.vasoftware.com/sourceforge.

[10] Xtras.Net (2007). http://xtras.net

[11] CompoNex (2007). http://www.componex.biz

[12] Boldyreff, C.; Nutter, D.; Rank, S. (2002) "Open-Source Artifact Management". Workshop Open Source Software Engineering, USA.

[13] Oliveira, J.P.F.; Santos, M.S.; Elias, G. (2006) "ComponentForge: Um Framework Arquitetural para Desenvolvimento Distribuído Baseado em Componentes. VI Workshop de Desenvolvimento Baseado em Componentes. Recife-PE.

[14] Fielding, R. T. (2000) "Architectural Styles and the Design of Network-based Software Architectures". PhD Thesis, University of California.

[15] Schuenck, M. (2006) "X-ARM: Um Modelo de Representação de Artefatos de Software". Dissertação de Mestrado, DIMAp-UFRN, Natal-RN.

[16] Ferraiolo, D. F.; Sandhu, R.; Gavrila, S.; Kuhn, D. R.; Chandramouli, R. (2001) "Proposed NIST Standard for Role-Based Access Control". ACM Transactions on Information and Systems Security, Vol. 4, No. 3, pp. 224-274.

[17] Holanda, C.B.S.; Souza, C.A.A.; Melo, W.L. (2001) "ProReuso: Um Repositório de Componentes para Web Dirigido por um Processo de Reuso". Simpósio Brasileiro de Engenharia de Software.

[18] Preiss, O.; Wegmann, A. (2002) "A System Perspective on the Quality Description of Software Components", 6th World Multi-Conference on Systemics, Cybernetics and Informatics, USA.

[19] Sun Microsystems. (2006) "Java™ Platform, Enterprise Edition 5". http://java.sun.com/javaee/5/docs/API

[20] Stal, M. (2002) "Web Services: Beyond Component Based Computing". Communications of the ACM, Vol. 45, Issue 110, pp. 71-76.

[21] JBoss Reference Guide (2007). http://docs.jboss.com/jbportal/v2.6/reference-guide/en/html.

[22] JMX (Java Management Extensions) "Java Management Extensions". http://java.sun.com/j2se/1.5.0/docs/guide/jmx.

[23] Hartman, B.; Flinn, D.J.; Beznosov, K.; Kawamoto, S. (2003) "Mastering Web Services Security". Wiley Publishing Inc.

[24] MySQL 5.0 (2007) "Reference Manual". MySQL AB.

[25] Cheesman, J.; Daniels, J. (2001) "UML Components: A Simple Process for Specifying Component Based Software". Addison-Wesley.