Governança de Ativos em Ecossistemas de Software na Biblioteca Brechó

Leonardo Antunes dos Rios

Projeto Final de Curso submetido ao Departamento de Engenharia Eletrônica e de Computação da Escola Politécnica da UFRJ como parte dos requisitos obrigatórios para conclusão do curso de Engenharia de Computação e Informação.

Apresentado por:

_____
Leonardo Antunes dos Rios

Aprovado por:

_____
Profa. Cláudia Maria Lima Werner, D.Sc. (Presidente)

_____
Prof. Rodrigo Pereira dos Santos, M.Sc. (Co-Orientador)

_____
Aloysio de Castro Pinto Pedroza, D.Sc.

_____
Benno Eduardo Albert, B.Sc.

Rio de Janeiro, RJ – Brasil
Dezembro de 2013

Agradecimentos

Leonardo Antunes dos Rios

Primeiramente agradeço a Deus, por toda força nos momentos difíceis, paciência para os momentos de incompreensão e desespero, e por fim, força de vontade para sempre seguir em frente. Sem ele, com certeza, não estaria aqui hoje, me formando numa das melhores universidades do país e em um dos cursos de maior prestígio.

Agradeço a meus pais, Simone e Fernando, por todo incentivo, conselho e compreensão nos momentos de estresse. Por todo amor e carinho indispensáveis para criar um ambiente propício para minha formação. Dedico não só esse trabalho final, mas toda a minha formação a vocês, por sempre buscarem a melhor educação para mim.

A meu irmão, fiel companheiro e melhor amigo! As batalhas travadas durante a vida são sempre menos dolorosas quando sabemos que ao nosso lado há alguém que dali nunca sairá!

Um agradecimento todo especial para a minha filhinha Laura, por toda a alegria que me trouxe com o seu nascimento e saúde demonstrada desde os primeiros dias de vida...

Aos meus amigos de faculdade, sempre presentes nas horas boas e ruins, nas horas de brincadeiras e nos estudos. A vida academia seria muito mais difícil sem a presença de vocês! O meu sucesso, com certeza, teve e sempre terá a influência de vocês. Em especial, agradeço a Laura Moraes, Rodrigo Couto e Felipe Bogossian.

Aos meus orientadores Claudia Werner e Rodrigo Santos, por todo apoio e ajuda, não só neste projeto final, mas também durante os anos de iniciação científica, onde pude desfrutar de diversos conhecimentos, muito importantes para a minha formação. Agradeço também ao mestrando Benno Albert por toda ajuda no trabalho desenvolvido.

Resumo

Governança de Ativos em Ecossistemas de Software na Biblioteca Brechó

Leonardo Antunes dos Rios

Orientadores: Cláudia Maria Lima Werner e Rodrigo Pereira dos Santos

Entre as várias formas de desenvolver software, geralmente utiliza-se a metodologia de dividir um problema grande em partes menores, denominadas módulos, facilitando o seu entendimento e diminuindo a complexidade inicial. Cada módulo pode ser chamado de componente ou serviço, tendo por característica básica o caráter autocontido. Muitos dos sistemas, entretanto, possuem o mesmo domínio de conhecimento, formando um nicho de mercado ou ecossistema de software, os quais organizam soluções para um problema conhecido através do incentivo à prática de reutilizar componentes e serviços desenvolvidos, além de fornecer uma tecnologia de software central que permita o desenvolvimento e reutilização de aplicações. Para que seja possível reutilizar esses produtos, é necessário que eles estejam armazenados em algum lugar e que as suas informações básicas de utilização e funcionalidade estejam disponíveis. Com a criação dos ecossistemas, as organizações fornecedoras neles inseridas disputam o mercado entre si, o que requer maior controle sobre seus produtos. Da mesma forma, faz-se necessário que as organizações consumidoras de tais produtos tenham um controle sobre quais produtos estão utilizando e suas respectivas permissões de uso. Este trabalho tem por objetivo adaptar uma biblioteca de componentes e serviços de software para que esta possa funcionar como uma ferramenta de governança baseada na gerência de produtos adquiridos na forma de componentes, serviços e aplicações por uma organização, produtos estes chamados de ativos de software. Com esta adaptação, vislumbra-se que a biblioteca seja capaz de informar os ativos e seus fornecedores, a sua frequência de uso interno à organização, bem como a quantidade de permissões de uso (i.e., licenças) que a organização possui sobre cada ativo.

Palavras-Chave: Ecossistemas de Software, Governança de Ativos de Software, Reutilização de Software.

Abstract

Assets Governance in Software Ecosystems at Brechó Library

Leonardo Antunes dos Rios

Supervisors: Cláudia Maria Lima Werner e Rodrigo Pereira dos Santos

Among the various ways of developing software, usually the methodology of dividing a large problem into smaller parts, called modules, is used, facilitating its understanding and decreasing the initial complexity. Each module can be called component or service, with the basic characteristic of the self-contained character. Many systems however possess the same knowledge domain, forming a niche market or software ecosystem which organizes solutions to a known problem by encouraging the practice of reuse of components and services developed, and provides a software technology center for the development and reuse of applications. To be able to reuse these products, it is necessary to have it stored somewhere and its basic information about usage and functionality should be available. With the creation of the ecosystems, vendors within these compete in the market among themselves, which requires greater control over their products. Likewise, it is necessary that consumers of such products have control over which products are used and their usage permissions. Thus, this work aims at adapting a library of components and software services to enable it to function as a tool for governance based on the acquired product management in the form of components, services and applications in an organization, these products are called software assets. With this adaptation, one sees that the library is able to inform the assets and their suppliers, their frequency of use internal to the organization, as well as the amount of usage permissions (i.e., licenses) that the organization has on each asset.

Keywords: Software Ecosystems, Software Asset Management, Software Reuse.

Índice

Capítulo 1. Introdução

Com a evolução da indústria de hardware e de software, os sistemas requeridos pelos clientes e usuários passaram a ter um número maior de funcionalidades, sendo estas muitas vezes mais complexas, pois têm a função de controlar, gerenciar e/ou visualizar aspectos críticos da organização, tanto do ponto de vista histórico como em tempo real [SOFTEX, 2011b]. De acordo com Werlang & Oliveira (2006), entre as maneiras de desenvolver sistemas robustos e complexos na Engenharia de Software (ES), utiliza-se a técnica de modularização a fim de facilitar a compreensão do problema e o desenvolvimento de soluções baseadas em módulos. Ao término da produção de cada módulo, estes devem se integrar de forma a resolver o problema.

Muitos dos sistemas desenvolvidos, entretanto, têm o mesmo domínio de aplicação e, por consequência, um conjunto de soluções similares [Werlang & Oliveira, 2006]. Nesse contexto, segundo Santos (2013), as organizações passaram a tentar desenvolver módulos na forma de componentes e serviços de software, que pudessem ser reutilizáveis, tanto em um cenário intra-organizacional como em um cenário de mercado, visando estabelecer um mercado relacionado. A meta era um menor custo de tempo e recurso para se alcançar a solução desejada, uma vez que a construção de um produto não mais começaria do zero. Para viabilizar este tipo de produção de software, faz-se necessário o uso de uma biblioteca na qual um conjunto de informações bem específicas sobre esses produtos estaria disponível e bem organizado para facilitar a busca e reutilização dos mesmos [Oliveira *et al.*, 2009].

Por outro lado, a aquisição desses produtos por organizações consumidoras de tecnologia se mostra problemática, segundo Santos (2013). Questões relevantes, tais como integração entre componentes, serviços e produtos em uso dentro da organização, a interoperabilidade entre as tecnologias, o controle de qualidade e manutenção devem ser considerados no momento da aquisição. Destaca-se ainda o fato de um componente ou serviço, pelo seu caráter mais geral, nem sempre atender a uma necessidade que a empresa consumidora possa vir a ter.

1.1. Motivação

Geralmente, a disponibilização dos componentes e serviços é feita na forma de artefatos (arquivos), que são recuperados e instalados em uma plataforma compatível, sendo Utilizados conforme sua natureza, já que os serviços são acessados remotamente através de comunicação por protocolos, enquanto componentes são instalados como partes da aplicação.

Porém, há uma série de problemas tecnológicos envolvidos em tal procedimento, o que às vezes inviabiliza a concepção adequada de um mecanismo de armazenamento e de busca intra-organizacional e, consequentemente, um mercado global ou corporativo de componentes e serviços. Entre estes problemas, destacam-se a padronização, visualização de informação, diretrizes para customização e propriedade intelectual [Werner *et al.*, 2009]. Na tentativa de resolver alguns destes problemas, surge a ideia de disponibilizá-los em um nível maior de abstração, na forma de aplicações. Para Bosch (2009), aplicação é um programa de computador que tem por objetivo auxiliar quem o utiliza a realizar uma tarefa específica a fim de solucionar o problema para o qual foi concebido, a partir de uma tecnologia de software central (plataforma).

A tendência da solução de problemas por aplicativos desenvolvidos a partir de diferentes plataformas estimulou ainda mais a concorrência e/ou cooperação entre organizações fornecedoras de tecnologia [Antunes *et al.*, 2013]. Segundo Albert *et al.* (2013), estas interações fomentam a ideia de um nicho de mercado (ou rede), no qual essas organizações fornecem produtos e/ou serviços relacionados similares e disputam ou colaboram no mercado. Desta rede de fornecedores, surgem os Ecossistemas de Software (ECOS), isto é, redes de organizações que se relacionam entre si baseadas em ferramentas, sistemas e serviços por elas disponibilizados, bem como nas tecnologias empregadas, nas formas de negócio que realizam, na maneira que se relacionam com os usuários finais (clientes), entre outros fatores [Manikas & Hansen, 2013].

Para uma organização que adquire componentes, serviços e/ou aplicações, e os tratam como elementos de valor para o seu negócio, estes são tratados internamente como ativos de software. De acordo com Microsoft (2013), um ativo de software é qualquer produto de cunho computacional que é gerado durante o ciclo de vida do software. Do ponto de vista de ECOS, um ativo de software é todo componente, serviço

e/ou aplicação que pode ser consumido interna ou externamente à organização, para auxiliar as atividades recorrentes do seu dia a dia [Santos, 2013].

1.2. Problema

Do ponto de vista econômico, a criação dos ECOS beneficiou as organizações fornecedoras de software, pois ampliou o número de clientes possíveis, consequência do encurtamento geográfico proporcionado pela globalização e pela Internet. Todavia, tornou-se importante para essas organizações saber exatamente quem são seus clientes e quais de seus ativos cada um deles está adquirindo, dado que geralmente estes ativos vêm vinculados a uma licença de utilização e são estas licenças que geram os lucros para as organizações fornecedoras [Popp, 2012].

Dessa forma, a gestão dos ativos de software torna-se indispensável do ponto de vista econômico para a organização fornecedora, visto que o seu lucro é fruto direto da venda de seus produtos e das licenças a eles associadas. Por outro lado, no que diz respeito a uma organização consumidora, a gestão dos ativos adquiridos é de suma importância, pois o seu custo está vinculado às licenças de cada ativo que a organização possui. Um número maior de licenças em relação ao qual de fato é necessário gera desperdício de recurso, ao passo que o contrário pode ainda gerar penalizações na forma de multas, por exemplo. Nesse sentido, a gestão de ativos de software representa uma questão fundamental para os diferentes tipos de organizações na indústria de software, destacada sobretudo sob o ponto de vista de ECOS.

1.3. Objetivo

Este trabalho tem por objetivo adaptar uma biblioteca de componentes e serviços de software para que esta possa funcionar como uma ferramenta de governança baseada na gerência de ativos de software de uma organização consumidora de tecnologia, o que inclui as aplicações utilizadas no seu dia a dia. Neste contexto, a gerência consiste em identificar a frequência de uso de cada ativo que a organização possui, os usuários internos que os utilizam, as licenças e *releases* disponíveis de cada um, além de visualizar o atual estado de utilização dos ativos adquiridos.

1.4. Contexto

Este trabalho está inserido no contexto do Projeto Brechó, desenvolvido pela Equipe de Reutilização de Software da COPPE/UFRJ. Este projeto tem por objetivo pesquisar conceitos do desenvolvimento baseado em componentes e serviços e construir soluções para uma biblioteca de suporte denominada Brechó [Brechó, 2013]. A biblioteca está disponível na forma de um sistema de informação Web, cujas principais características são publicação, documentação, armazenamento, busca e recuperação de componentes e serviços nela armazenados.

1.5. Organização

Esse projeto final de curso foi dividido em cinco capítulos, sendo o primeiro esta breve introdução, que apresentou o contexto e a motivação do trabalho, o problema detectado e o objetivo a ser alcançado com a pesquisa realizada.

O segundo capítulo discorre sobre a revisão da literatura relacionada aos temas envolvidos no trabalho, a saber: Componentes, Processo de Aquisição na Rede de Produção de Software, Ecossistemas de Software, Gestão de Ativos de Software e Gerência de Reutilização.

O terceiro capítulo apresenta a adaptação feita na biblioteca Brechó para que esta fosse capaz de realizar a gestão de ativos de Software de uma organização consumidora, relatando problemas e soluções encontrados, além da linha de raciocínio adotada para adaptar uma biblioteca de componentes e serviços para que esta possa contemplar o conceito de aplicação e a gerência de ativos de software de uma organização consumidora.

O quarto capítulo é dedicado à parte prática do trabalho, no qual são mostradas as adaptações feitas na biblioteca Brechó para envolver a gerência de ativos de software. Por fim, o quinto capítulo traz uma breve conclusão na qual são apresentados as contribuições deste trabalho e os potenciais trabalhos futuros.

Capítulo 2. Revisão da Literatura

Nesse capítulo é feita a revisão da literatura relacionada aos temas envolvidos no trabalho, começando com um estudo sobre componentes e como é feito o processo de aquisição desses componentes na rede de produção de software, seguido do estudo de ecossistemas de software que essa rede de produção gera ao seu redor. Na continuidade é estudado como uma organização consumidora de software faz a gestão de seus ativos de software e como esses ativos são gerenciados através da gerência de reutilização.

2.1 Componentes

Componente de software é uma unidade independente que possui funcionalidades e interfaces, podendo ter o seu código fonte acessível (ou não) para uso dos desenvolvedores. Segundo Braga (2010), componentes podem ser definidos como artefatos autocontidos que realizam uma função específica, com uma interface, que possuem documentação adequada e a definição de seu grau de reutilização. A utilização de um ou mais componentes em conjunto origina os chamados sistemas baseados em componentes. De acordo com Oliveira & Paula (2009), um componente de software possui, geralmente, as seguintes características:

- Encapsulamento: esconde a lógica envolvida no código-fonte do componente, deixando visível apenas a interface das funcionalidades ou métodos que o componente possui;
- Reutilização: capacidade do componente de ser adaptado (quando necessário) e reutilizado na criação de novos sistemas, que possuem características comuns dentro do domínio do problema para o qual o componente foi desenvolvido;
- Confiabilidade: o componente deve ser independente ao sistema, sendo capaz de tratar seus erros e exceções internamente para garantir a segurança do sistema;
- Comunicação: os componentes devem ser capazes de se comunicar com outros pela troca de mensagem contendo a informação requisitada e/ou solicitada.

O ramo da Engenharia de Software (ES) que estuda esses conceitos é a Engenharia de Software Baseada em Componentes (ESBC). A ESBC trata das diversas formas de se dividir um sistema em pequenas partes funcionais e lógicas (componentes), de forma a torná-las unidades independentes e capazes de serem

reutilizadas em diversas aplicações. Cada componente se comunica com outro por meio da sua interface através de troca de mensagens contendo dados existentes no sistema [Spagnoli, 2003]. Segundo Oliveira & Paula (2009), a ESBC é uma abordagem que permite reduzir custo e tempo necessários para desenvolver um projeto de software a fim de aumentar a sua qualidade.

A ideia de se construir sistemas a partir de pequenas partes independentes foi proposta no final dos anos 1960, quando McIlroy afirmou que o desenvolvimento de software deveria produzir componentes que pudessem ser reutilizados, de forma que o desenvolvedor escolhesse o componente que melhor satisfizesse a sua necessidade. Anos depois, DeRemer propôs que os sistemas deveriam ser desenvolvidos a partir de um conjunto de módulos (componentes) independentes, que se comunicariam entre si formando um único sistema no final do processo [Oliveira & Paula, 2009].

Para esses autores, a ESBC se assemelha com a metodologia adotada em um processo normal de desenvolvimento na ES do ponto de vista dos métodos utilizados. Após a fase de identificação dos requisitos, ao invés de elaborar um *design* para o produto, na ESBC, são escolhidos os componentes que farão parte do produto, analisando-os para verificar se deverão sofrer algum tipo de alteração para atender à demanda em questão.

A esse modelo de desenvolvimento no qual os diversos componentes disponíveis são utilizados para produzir o sistema, dá-se o nome de Desenvolvimento Baseado em Componentes (DBC). O DBC tem por principio básico construir sistemas a partir de componentes independentes, que sejam capazes de serem reutilizados em diversas aplicações [Almeida *et al.*, 2003]. O DBC consiste na criação de sistemas de software que utilizam componentes e a interação entre eles permitindo adicionar, adaptar, remover e/ou substituir partes de um sistema sem necessariamente substituir todo o sistema ou implementá-lo do zero [Feijó, 2007].

De acordo com Almeida *et al.* (2003), para ocorrer a reutilização de um componente, deve-se fazer uso de técnicas, métodos e ferramentas que deem suporte a identificação e especificação dos componentes, que vai desde a captura do domínio do problema até a implementação e execução do componente no sistema desenvolvido. Nesse contexto, existem duas perspectivas em DBC:

- Desenvolvimento de Componentes: criação de novos componentes independentes, desde a documentação à implementação. Neste caso, o armazenamento em uma biblioteca de ativos reutilizáveis serve para propiciar a

reutilização em projetos futuros cujo domínio do problema seja semelhante ao que originou a criação do componente;

- Desenvolvimento com Componentes: criação de novos sistemas a partir da utilização dos componentes armazenados na biblioteca de ativos reutilizáveis da organização. Nesse sistema, os componentes estão integrados e realizam troca de informação por meio de mensagens, visando implementar as funcionalidades.

Para Brown & Wallnau (1996), o DBC possui cinco estágios. São eles:

- Seleção: processo de escolher quais são os potenciais componentes que podem ser utilizados na construção do sistema, considerando tanto componentes COTS (*Commercial Off-The-Shelf*, i.e., componente genérico adquirido no mercado), como aqueles armazenados na biblioteca de ativos reutilizáveis da organização;
- Qualificação: verifica se os componentes selecionados possuem característica de reutilização, averiguando se são capazes de se adaptar à arquitetura utilizada no sistema em construção. Considera-se como métricas de qualidade desempenho, confiabilidade e usabilidade que este componente terá em relação ao sistema;
- Adaptação: fase em que o componente passa por modificações para se adequar à necessidade do sistema em construção;
- Composição: fase na qual os diversos componentes selecionados e devidamente adaptados são integrados para construir o sistema e atender a necessidade para o qual este foi requisitado;
- Atualização: capacidade de um componente mais antigo ser substituído (atualizado) por outro similar, porém com novas funcionalidades, correções de *bugs*, entre outros benefícios/melhorias. Essa atualização deve ser feita de maneira que o desenvolvedor que usa este componente não sinta a mudança, ou seja, nenhum tipo de impacto negativo deverá ser observado após a atualização.

Apesar dos avanços registrados na ESBC, existem ainda alguns problemas a serem resolvidos. Destaca-se a carência de métodos e ferramentas de apoio ao desenvolvimento de sistemas utilizando DBC e a dificuldade de integração das técnicas e ferramentas que apoiam o uso do DBC [Almeida *et al.*, 2003]. Deve-se destacar, entretanto, a existência de diferenças entre desenvolvimento de componentes e desenvolvimento com componentes. No primeiro caso, há a elaboração e

implementação do componente a fim de que seja futuramente utilizado e reaproveitado, diminuindo custos financeiros e temporais dos projetos em que esteja presente. No segundo caso, o componente existe e está armazenado, podendo ser utilizado na construção de um novo sistema e/ou componente [Feijó, 2007].

2.2 Processo de Aquisição na Rede de Produção de Software

Com os avanços tecnológicos, a maioria das organizações se adaptou a uma nova realidade, modernizando as suas áreas e deixando-as mais informatizadas [SOFTEX, 2011a]. Porém, não necessariamente todas as organizações possuem um setor específico de informática, no qual são desenvolvidos os sistemas e ativos de software que são utilizados dentro da organização. Nesse caso, as organizações que não possuem esse recurso internamente se veem obrigadas a procurá-los no mercado, terceirizando o desenvolvimento de seus sistemas para as "fábricas de software", também conhecidas como organizações fornecedoras.

Essas fábricas consistem em organizações cujo foco é produzir e desenvolver componentes, serviços e aplicações que atendam à necessidade de um nicho de mercado e/ou segmento [Castor, 2006]. Entretanto, alguns questionamentos emergem [Santos & Werner, 2012], e.g., "como saber que opção vai atender melhor a organização em suas necessidades?" e "ao término do processo, o produto entregue terá atingido o objetivo pelo qual foi adquirido?". Torna-se importante, então, estabelecer critérios para eliminar organizações fornecedoras candidatas que não serão capazes de atender à demanda, mantendo apenas os potenciais fornecedores [SOFTEX, 2011b]. No contexto deste trabalho, a palavra "adquirir" deve ser entendida sempre como "comprar". Desconsidera-se casos de produtos que são distribuídos livremente, como software livre, GPL (*General Public License*), Software de Domínio Público, entre outros.

Em certos casos, o produto adquirido é um produto pré-definido que fornece uma solução genérica para um problema conhecido, caso dos sistemas operacionais para computadores, por exemplo. Geralmente, neste tipo de aquisição, o produto vem sempre vinculado a licenças e versões [Castor, 2006]. As versões dizem respeito ao pacote de melhorias que um produto sofre ao longo do tempo e são disponibilizados em um "novo" produto como evolução do anterior. Essas melhorias podem ocorrer em nível de correção de problemas e *bugs*, mudança de tecnologia utilizada, customização, melhoria gráfica do *front-end* do produto, entre outras [Sabino & Kon, 2009]. A licença é a

permissão que a organização fornecedora dá à organização consumidora (adquirente) para que esta possa utilizar o software ou sistema adquirido. Uma licença pode ser dada de várias maneiras, são elas [Kon *et al.*, 2011]:

- Compra de licença permanente: adquire-se o produto e código serial vinculado, que permite o seu uso por tempo indeterminado. Nesse caso, não se tem o acesso ao código fonte, porém, pode-se utilizá-lo como se fosse o proprietário, com total acesso a todas as funcionalidades do produto;
- Compra de licença temporária: adquire-se um produto, porém, este vem vinculado a uma data limite de uso. Após esta data, a licença expira e há a necessidade de obtenção de nova licença (renovação da licença), ou o produto ficará inacessível, tendo suas funcionalidades automaticamente bloqueadas.

De acordo com Kon *et al.* (2011), ainda nesse tipo de aquisição, é comum ter dentro das organizações um número *X* determinado de licenças e produtos, ou seja, para cada produto adquirido, existe uma licença de uso vinculada, que pode variar entre as formas possíveis de licença e também de versão. Para Castor (2006), a segunda maneira de se adquirir um produto na indústria é buscar fábricas de software que desenvolvam um produto específico para um problema. Neste caso, o produto final não terá uma solução genérica e atenderá exclusivamente ao problema demandado pelo adquirente.

Assim sendo, a aquisição inicia com a identificação das necessidades do cliente e termina com a entrega do produto pelo fornecedor, tendo em mente que o produto deve atender às demandas especificadas inicialmente [SOFTEX, 2011a]. Buscando auxiliar este tipo de aquisição, modelos de maturidade foram desenvolvidos para apoiar tanto a organização fornecedora como a consumidora. No Brasil, o modelo CMMI (*Capability Maturity Model Integration – Acquisition*) e o Modelo MPS (Melhoria do Processo do Software Brasileiro) são os mais conhecidos, sendo este último, nacional.

No MPS, há duas vertentes para uma organização que deseja adquirir algum tipo de produto de uma fábrica de software. Há o caso em que uma fábrica de software adquire um produto (ou parte deste) e utiliza em seu produto final. Por outro lado, existe o caso em que uma organização não desenvolve nenhum tipo de produto de software, adquirindo todos eles de fábricas de software. Para o primeiro caso, tem-se o processo de aquisição vinculado ao nível F de maturidade do MPS, enquanto que no segundo caso, utiliza-se o Guia de Aquisição para auxiliar tais organizações a adquirirem um produto junto às fábricas de software.

O Processo de Aquisição descrito no nível F tem por objetivo garantir que, ao término do processo, o produto desenvolvido atenda às necessidades expressas pelo consumidor. Nem toda organização que implementa o MPS é obrigada a implementar esse processo, porém toda organização que se enquadre em um ou mais dos itens a seguir deve obrigatoriamente implementá-lo [SOFTEX, 2011b]:

- A organização contrata o desenvolvimento de algum componente de software de terceiros e este componente será entregue juntamente com o produto que esta organização desenvolve;
- A organização possui duas ou mais unidades organizacionais com processos diferentes e uma contrata a outra para o desenvolvimento de parte do software que será entregue a um cliente;
- A organização desenvolve produtos em parceria com outras organizações;
- A organização está implantando um programa de reutilização.

O Processo de Aquisição torna-se obrigatório para todas as organizações que obtém algum componente que faz parte de seu produto final, ou um componente que auxilie no seu funcionamento. Independentemente de qual nível do MPS a organização esteja implementando, ou que já tenha sido avaliada, o Processo de Aquisição deverá estar implantado [SOFTEX, 2011b]. O Processo de Aquisição apresenta oito resultados esperados [SOFTEX, 2011b]:

- **AQU1** – As necessidades de aquisição, as metas, os critérios de aceitação do produto, os tipos e a estratégia de aquisição são definidos;
- **AQU2** – Os critérios de seleção do fornecedor são estabelecidos e usados para avaliar os potenciais fornecedores;
- **AQU3** – O fornecedor é selecionado com base na avaliação das propostas e dos critérios estabelecidos;
- **AQU4** – Um acordo que expresse claramente as expectativas, responsabilidades e obrigações de ambas as partes (cliente e fornecedor) é estabelecido e negociado entre elas;
- **AQU5** – Um produto que satisfaça a necessidade expressa pelo cliente é adquirido baseado na análise dos potenciais candidatos;

- **AQU6** – A aquisição é monitorada de forma que as condições especificadas sejam atendidas, tais como custo, cronograma e qualidade, gerando ações corretivas quando necessário;
- **AQU7** – O produto é entregue e avaliado em relação ao acordado e os resultados são documentados;
- **AQU8** – O produto adquirido é incorporado ao projeto, caso pertinente.

Por outro lado, no caso de organizações que não desenvolvem nenhum tipo de software, sendo todos os sistemas e serviços utilizados adquiridos junto a desenvolvedores externos à organização, existe o Guia de Aquisição [SOFTEX, 2011a]. A intenção deste guia é auxiliar as organizações a realizar a aquisição de forma correta e segura, tentando garantir que, ao término do processo de aquisição do novo produto, os objetivos e metas sejam alcançados. Este guia é divido em quatro atividades: Preparação da Aquisição, Seleção do Fornecedor, Monitoração do Contrato, Aceitação pelo Cliente. Além disso, subatividades estão vinculadas a cada atividade. A Figura 1 mostra como o Processo de Aquisição e o Guia de Aquisição são utilizados pelas organizações.

Figura 1 – Processo de Aquisição do nível F e Guia de Aquisição do MPS.BR

No exemplo de fluxo, a Organização 1 é aquela que não desenvolve nenhum tipo de produto computacional, adquirindo todos seus sistemas de software de uma organização terceirizada. Por isso, ela deve utilizar o Guia de Aquisição, que é voltado para organizações com este perfil. A Organização 2 funciona como fábrica de software, mas também como organização consumidora. Quando ela fornece um produto para a Organização 1, ela atua como fábrica de software, e quando ela adquire um produto da Organização 3, ela atua como organização consumidora. Nesse caso, ela deve utilizar o Processo de Aquisição, uma vez que está utilizando um componente ou produto adquirido como parte do produto que ela fornece para a Organização 1. A Organização 3, no exemplo, funciona exclusivamente como uma fábrica de software, podendo ou não ter implementado o modelo MPS. Quando um produto é adquirido, seja pela

Organização 1 ou pela Organização 2, este passa a fazer parte dos ativos da organização e a gerar valor para a mesma.

2.3 Ecossistemas de Software

Um ecossistema pode ser entendido como uma comunidade em que seus indivíduos se relacionam entre si e com o próprio ambiente onde convivem, funcionando juntos como um grande sistema [Santos, 2013]. São inúmeros os ecossistemas existentes, assim como suas peculiaridades e características, destacando-se os ecossistemas ecológicos, comerciais e sociais. Um Ecossistema de Software (ECOS) seria a combinação de conceitos presentes nestes ecossistemas. Outra maneira de se entender um ECOS seria enxergá-lo como um ramo do ecossistema comercial no qual há a presença de um elemento centralizador (proprietário), uma plataforma (mercado) e os agentes de nicho (atores) [Santos & Werner, 2012].

Segundo Yu & Deng (2011), um ECOS pode ser definido como um conjunto de organizações e a inter-relação entre elas dentro do contexto de um nicho de mercado de produtos ou serviços de software relacionados. Para Jansen *et al.* (2009), um ECOS é um conjunto de atores funcionando como uma unidade que interage como um mercado de software compartilhado, considerando também os relacionamentos existentes entre eles. Por fim, Campbell & Ahmed (2010) discutem ECOS como uma forma de incentivar desenvolvedores externos a utilizarem a plataforma disponibilizada por uma organização a fim de contribuir no desenvolvimento de produtos sobre ela. Tendo em vista a última definição, nota-se que não apenas a organização fornecedora da plataforma tem sua importância, mas que desenvolvedores externos passaram a ter papel importante no processo de desenvolvimento, direta ou indiretamente. O resultado dessa mudança de foco interfere, por exemplo, na linha de produção de software, causando efeitos no modelo tradicional de produção das indústrias de software.

Um ECOS se diferencia de um modelo tradicional de indústria no que diz respeito à forma como as relações são estabelecidas. Numa cadeia de valor de software tradicional (fechada), existe a relação direta entre os clientes/usuários finais e o fornecedor, enquanto numa cadeia de valor aberta, o fornecedor provê uma plataforma e depende de desenvolvedores externos para gerar valor [Popp, 2012]. Segundo Santos & Werner (2012), no modelo tradicional, o elemento chave é a organização fornecedora, responsável por manter a plataforma e desenvolver e prover os artefatos. Por sua vez, no

ECOS, as organizações consumidoras dos artefatos passam a ter papel importante, podendo atuar inclusive no desenvolvimento destes e também na plataforma, sendo conhecidos como agentes de nicho, pois ora atuam como consumidoras, ora como fornecedoras de produtos [McGregor, 2012]. De acordo com Yu & Deng (2011), a transição entre o modelo tradicional de estrutura e relacionamento da indústria e o modelo de ecossistema provavelmente causará impactos nos modelos de negócio, bem como nas escolhas de qual tecnologia utilizar.

Há ainda outro modelo de desenvolvimento chamado de FOSS (*Free and Open-Source Software*), no qual a organização centralizadora é a própria comunidade e seus participantes, sendo o desenvolvimento de software realizado em processo aberto em que todos têm acesso aos códigos-fonte em um ambiente distribuído [Santos, 2013]. Apesar desses modelos, mesmo tendo suas bases apoiadas em ecossistemas existentes e trazendo destes muitos termos, métodos e princípios, o ECOS ainda não possui um corpo de conhecimento capaz de dar diretrizes aos pesquisadores da área e, sendo assim, o assunto vem recebendo atenção especial por parte dos especialistas, ganhando crescente destaque no número de pesquisas atuais [Santos, 2013].

Os ECOS são formados basicamente por uma plataforma, um conjunto de atores que participam e interagem nessa plataforma por meio de regras de negócio, e uma rede social, canal de troca de informação entre os participantes. Outra visão sobre como os ECOS são organizados foi sugerida por Campbell & Ahmed (2010), na qual um ECOS teria três dimensões:

- **Arquitetura**: dimensão mais técnica com foco na plataforma envolvida no ECOS (i.e., tecnologia, infraestrutura, linguagem de programação, software). Nesta dimensão, são tratados os detalhes técnicos dos produtos e serviços, assim como os requisitos, os atores, as permissões e os níveis de acesso;
- **Negócio**: o foco está no conhecimento que circula no ECOS, considerando as expectativas em torno da plataforma e produtos, os impactos que podem causar na produtividade, os modelos de negócio envolvidos e as oportunidades geradas;
- **Social**: tem seu foco nos atores do ECOS e as relações estabelecidas entre eles e a comunidade, tentando gerenciar seus membros, as vantagens e desvantagens de se permanecer no ECOS, o conhecimento trocado entre os participantes e o seu armazenamento, entre outros.

De acordo com Jansen *et al.* (2009), o ECOS possui três níveis de escopo (Figura 2): (a) no primeiro nível, os objetos de estudo são os atores e suas ligações dentro da organização; (b) no segundo nível, os objetos de estudo focam na rede de produção de software, do inglês *Software Supply Network* (SSN), e nas suas diferentes relações; e (c) no terceiro nível, os objetos de estudo são os próprios ECOSs e as suas relações. Partindo dos níveis de escopo, pode-se observar algumas peculiaridades oriundas destes tipos de agrupamento:

- Um ator pode estar inserido em mais de um ECOS, com diferentes direitos e deveres em cada um. Além disso, ao adquirir um ativo de uma organização de software, o ator pode estar consumindo transitivamente artefatos que constituem este ativo, de diferentes fornecedores, inserindo-o em diferentes SSNs;
- Um fornecedor de software pode estar associado, direta ou indiretamente, a mais de uma organização de software, fazendo parte, assim, de mais de uma SSN;
- Parte de um ECOS pode estar inserido em outro ECOS. Isso pode ocorrer quando uma mesma tecnologia, por exemplo, é utilizada por um ou mais elementos presentes em um ECOS. Além da tecnologia, pode-se citar também o uso de um mesmo modelo de negócio, nicho de mercado, público alvo, entre outros fatores técnicos e sociais que podem estar presentes.

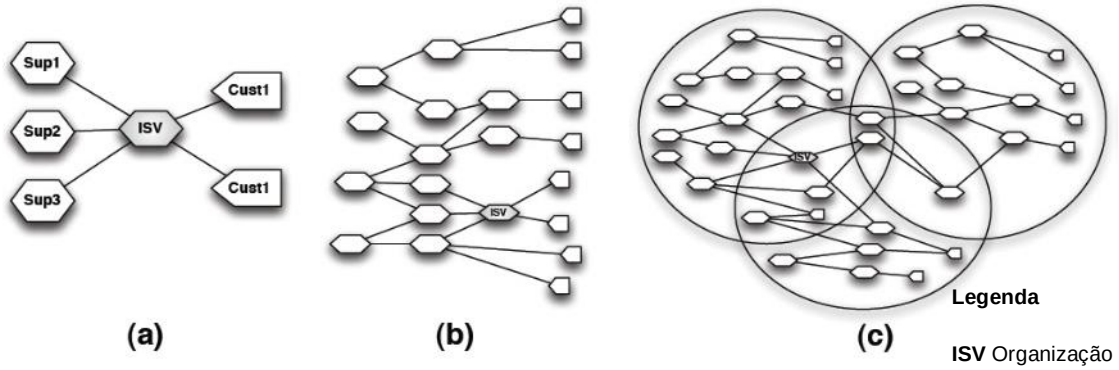

Figura 2. Níveis de escopo. Fonte: (Jansen et al., 2009)

Além disso, é comum encontrar um ECOS dentro de outros ECOS. É o que acontece quando analisamos o ECOS Microsoft e o ECOS Apple (Figura 3), por exemplo. Em ambos, o ECOS mais externo (e maior) é a própria organização enquanto os ECOS contidos dentro dele (menores) são ECOS constituídos muitas vezes por seus produtos e serviços [Santos, 2013].

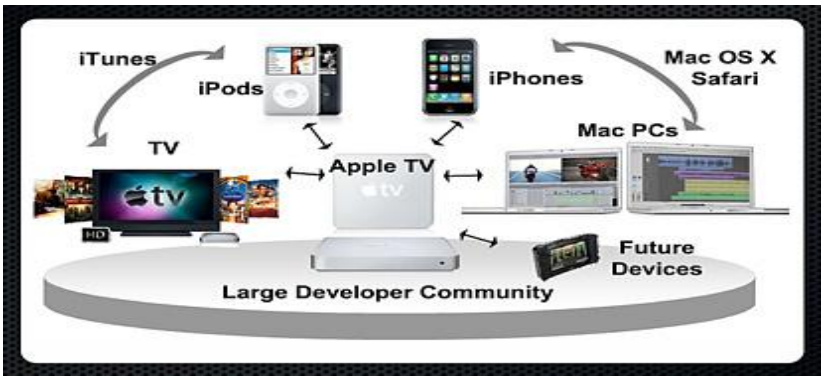

Figura 3. ECOS da Apple1

Diversos estudos vêm sendo realizados a respeitos de ECOS. Em 2011, Santos & Werner (2011) publicaram uma sumarização de artigos que haviam sido redigidos até o momento a respeito de ECOS, apresentando seus conceitos e estudos relacionados. Ao longo dos anos, alguns benefícios da sua utilização foram encontrados. Segundo Santos (2013), destacam-se:

- A contribuição dos atores quando inseridos em um ECOS gera benefícios tanto para o ator como para a comunidade do ECOS na qual este está inserido, seja de forma a gerar valor para o ECOS, seja mantendo-o ativo e produtivo, ou ainda de forma lucrativa para o próprio ator;
- O processo de desenvolvimento e evolução do software sofre aceleração em todo o seu processo, uma vez que diversos atores podem contribuir ao mesmo tempo, efetuando correções e melhorias;
- Redução dos custos, principalmente devido à redução de erros e retrabalho;
- Funcionalidades mais completas, principalmente pela troca de conhecimento presente dentro dos ECOS;
- Maior troca de informação e, portanto, maior geração de conhecimento, melhor e mais completo.

Buscando uma forma de analisar os ECOSs, Santos & Werner (2012) desenvolveram um framework denominado ReuseECOS, no qual as três dimensões dos ECOSs são expandidas por passos (conjunto de atividades que analisa um aspecto de uma dimensão do ECOS) e uma quarta dimensão é proposta. As três dimensões do

[1] Fonte: http://www.todaysiphone.com/2011/12/should-apple-license-out-software/

ReuseECOS são chamadas de Técnica, Transacional e Social, ao passo que a quarta é chamada de E&G (Engenharia e Gerenciamento) como mostrado na Figura 4. A dimensão técnica tem seu foco no elemento plataforma, o que inclui as tecnologias, a infraestrutura e os métodos e modelos utilizados para desenvolvimento. A dimensão transacional tem o foco no elemento conhecimento envolvido na plataforma, i.e., recursos, artefatos e informações. A dimensão social foca no elemento ator presente no ECOS a fim de entender como a rede social influencia na proposição do valor e na gestão dos artefatos da plataforma. A dimensão E&G foca em intermediar a comunicação e fluxo de informação entre os passos e atividades das outras dimensões a fim de disponibilizar um ambiente favorável para o ECOS se estabelecer e manter.

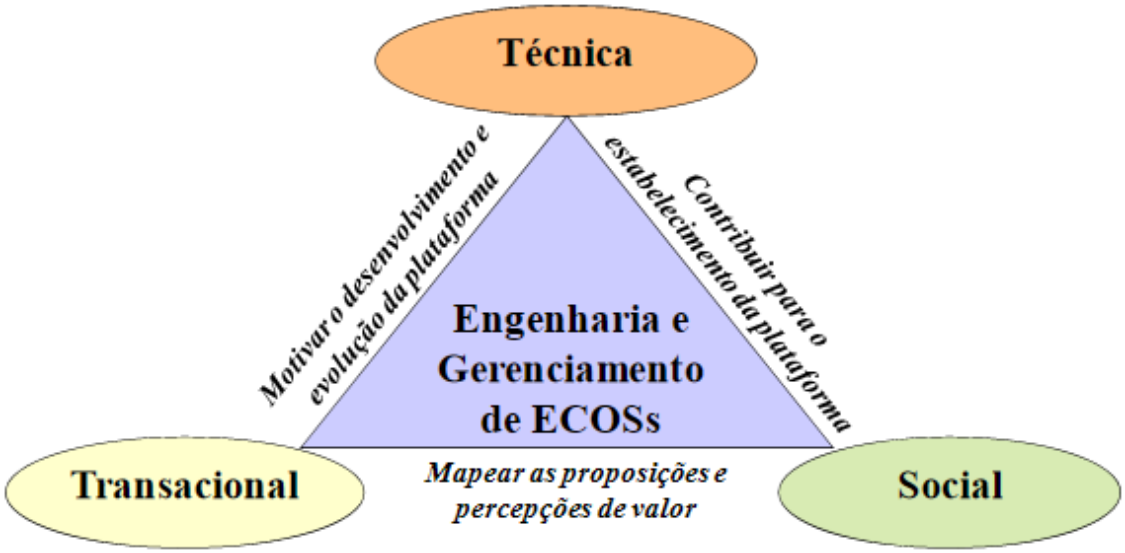

Figura 4 - Framework "3+1" para E&G de ECOSs. [Santos & Werner, 2012]

Foram detectadas também algumas barreiras e dificuldades a serem verificadas nos ECOS. Destacam-se [Santos, 2013]:

- Modelar os relacionamento existentes entre os atores participantes do ECOS;
- Gerir as diversas licenças de software existentes;
- Construir ferramenta de apoio à interação social dentro do ECOS.

2.4 Gestão de Ativos de Software

Como já definido anteriormente, o ativos de software são artefatos produzidos ou adquiridos e armazenados por uma organização. Esses ativos são trechos de código fonte, componentes, serviços, aplicações desktop, Web ou *mobile*, servidores de e-mail,

hardware, software e todo tipo de tecnologia que, direta e/ou indiretamente, auxilia na execução das atividades diárias. A Gestão de Ativos de Software (do inglês *Software Assets Management*, ou SAM) é uma prática comercial que envolve a gerência e otimização das práticas de aquisição, implantação, uso e manutenção dos ativos de software de uma organização [Williams & O'Connor, 2011]. Segundo a Microsoft (2013), SAM é um conjunto de processos que dá condições de gerenciar, controlar e proteger os ativos de software existentes dentro de uma organização. O objetivo central da SAM é ser parte estratégica da área de Tecnologia da Informação (TI), buscando reduzir custos e riscos associados à aquisição e utilização dos produtos implantados, além de melhorar o tempo de resposta e a produtividade [ADOTI, 2011].

SAM pode desempenhar diversas funções dento de uma organização, dependendo do ambiente de TI e dos produtos de software utilizados. Basicamente, a maioria das organizações utiliza SAM para gerir as licenças dos produtos, pois estes são ativos, fruto de investimentos, e devem ser gerenciados [Microsoft, 2013]. De acordo com a Microsoft (2013), as licenças são responsáveis por utilizar grande parte do orçamento destinado à área de TI de uma organização. Isso pode ocorrer tanto para o caso de haver mais licenças do que o necessário, ou o contrário, quando há menos licença do que ativos que estão sendo utilizados correndo assim o risco da organização ser penalizada através de multas.

Ter um controle sobre o número de licenças necessárias e em uso por um mecanismo de inventário, por exemplo, garante que os produtos de software que a organização possui estão sendo utilizados efetivamente no dia a dia de trabalho, avaliando se o orçamento previsto para a área está sendo corretamente aplicado [Williams & O'Connor, 2011]. Outros objetivos de se utilizar SAM podem ser [Microsoft, 2013]:

- Reduzir o custo de software e manutenção;
- Cumprir regras de segurança e padrões desktop;
- Melhorar produtividade da equipe de TI;
- Estabelecer procedimentos e políticas de aquisição, implantação, utilização e suporte dos ativos de software da organização;
- Garantir a conformidade de licenças;
- Centralizar o rastreamento dos ativos de TI;
- Minimizar o uso de software pirata, produtos sem licenças ou fora do prazo.

Segundo Williams & O'Connor (2011), Microsoft (2013) e Coutinho & Rangel (2010), uma boa prática de SAM compreende:

- Começar com a produção de um inventário no qual constam todos os atuais ativos de software da organização e como cada um é utilizado. Dessa forma, pode-se ter uma visão inicial da situação da organização em relação aos ativos, possibilitando que algumas decisões sejam tomadas;
- Após essa fase inicial, é necessário que seja reavaliado como cada ativo de software vem sendo ou não utilizado dentro da organização, a fim de encontrar onde os gastos estão ocorrendo (ou se não está havendo utilização dos recursos em sua máxima capacidade). Encontrados tais gargalos, uma primeira medida a ser tomada pode ser reduzir o investimento no ativo em questão, aumentando os recursos financeiros para áreas em que o quadro se mostre o oposto;
- A próxima etapa é associar os ativos às suas respectivas licenças de uso, caso o ativo tenha uma licença vinculada. Dedica-se uma etapa do processo em direção a boas práticas de SAM ao gerenciamento das licenças, pois, conforme mencionado, este tem sido um dos elementos de TI que mais tem impactado no orçamento destinado a área, tornando a sua gerência um ponto determinante para cortes de custos e investimentos;
- A etapa seguinte diz respeito ao desenvolvimento de procedimentos de gestão dos ativos propriamente ditos, sendo especificadas as documentações relacionadas aos ativos, as políticas de uso, os marcos importantes relacionados às licenças, informações gerais de cada ativo, entre outros;
- Por fim, deve-se estabelecer mecanismos de coleta de informação sobre os ativos e seu uso, possibilitando que decisões sejam tomadas em tempo real e que todo o processo de gerência não tenha que ser reiniciado. Esse processo pode ser feito por meio da coleta de métricas e controle de utilização.

Com a crescente utilização de SAM, a *International Organization for Standardization* (ISO) e a *International Electrotechnical Commission* (IEC) começaram a desenvolver em 2003 um conjunto de normas para a melhor prática de gerenciamento de ativos de software. Foi criada então, em 2006, a SAM ISO/IEC 19770, norma que foi subdividida em duas partes, sendo a primeira referente a ambiente de controle, planejamento, implantação, inventário, verificação, conformidade e ciclo de vida dos

ativos de software, enquanto a segunda parte se refere à prática da identificação dos ativos pelo método de etiquetação dos ativos, visando melhorar o processo de busca e recuperação [Microsoft, 2013]. Assim, SAM ISO/IEC 19770 tem por objetivo beneficiar as organizações da seguinte maneira [Microsoft, 2013]:

- Permitir a identificação de oportunidades que irão trazer melhorias a longo prazo para a organização;
- Alinhar as boas práticas de SAM com o gerenciamento de serviço da ISO/IEC 20000, assim como a estrutura da ITIL (*Information Technology Infrastructure Library*);
- Permitir que novas ferramentas e metodologias sejam desenvolvidas pelo setor de TI utilizando a norma e tendo como *feedback* as avaliações de risco dessas novas implementações;
- Proporcionar uma boa maneira de fazer a governança corporativa na área de TI.

Estudos mostram que as auditorias de software estão aumentando cada vez mais, com destaque para organizações como IBM, Adobe, Microsoft e Oracle, organizações que mais auditorias realizaram em 2012 na área de ativos de software. Além disso, as pesquisas indicam que boa parte das auditorias foi parar nos tribunais, por conta de licenças vencidas ou até mesmo a falta delas. O estudo relata ainda que boa parcela dos fabricantes de software não tinham conhecimento referente a qual versão de seus produtos os clientes estavam utilizando atualmente [Rangel, 2013].

Estes problemas poderiam ser evitados ou pelo menos amenizados utilizando modelos escalonáveis de crescimento, previstos nas boas práticas de gerenciamento de ativos de software. Esses modelos permitem rápida adaptação às oportunidades de mercado, prevendo as necessidades futuras com o crescimento previsto [ADOTI, 2011 e Coutinho & Rangel, 2010]. Existe hoje no mercado um conjunto de ferramentas que se propõem a auxiliar a gerência de ativos de uma organização. Essas ferramentas nem sempre se limitam a gerenciar o uso das licenças dentro das organizações, mas também dão uma visão ampla de todos os ativos relacionados à área de TI [ADOTI, 2011].

De acordo com Coutinho & Rangel (2010), as ferramentas de SAM estão focadas em ambientes complexos de TI, onde o acompanhamento e controle sobre os ativos da área são importantes fatores para a tomada de decisão estratégica. A gerência ocorre em nível de software e hardware, permitindo que a organização tenha uma visão

geral de todo seu inventário e como este vem sendo utilizado. Além disso, em algumas ferramentas, é possível a especificação do local onde cada ativo se encontra.

2.5 Gerência de Reutilização

Além da gerência de ativos de software em nível corporativo, é necessário que seja gerenciado também o ciclo de vida de cada um deles, em nível mais técnico. Nesse caso, no modelo MPS, há o processo de Gerência de Reutilização (GRU), cujo objetivo é exatamente realizar a gerência do ciclo de vida dos ativos reutilizáveis [SOFTEX, 2011c]. Um ativo reutilizável, muitas vezes, não é um produto totalmente novo, sendo baseado em outros e adaptados às necessidades da organização. A reutilização dos ativos faz com que a produtividade da organização aumente, além de tornar os processos cada vez mais confiáveis e seguros. Para que um ativo possa ser reutilizado, devemos tê-lo armazenado para que possa ser localizado. Geralmente, utiliza-se o mecanismo de biblioteca para armazenar os ativos, onde é ainda armazenado todo tipo de informação que possa ser útil na sua busca e recuperação [Filho *et al*., 2008].

Além de armazenar o ativo reutilizável, a biblioteca tem papel de gerenciar a evolução e as versões do ativo no decorrer do tempo. Isso ocorre devido a solicitações de novas demandas por melhoria, adaptações e correções de *bugs* que devem ser armazenadas na biblioteca, mantendo-se o histórico das versões. A importância de se manter tal histórico vem do fato do ativo estar presente em alguma aplicação e a sua exclusão ocasionar, possivelmente, algum problema na execução da aplicação [SOFTEX, 2011c].

O primeiro passo para se fazer a Gerência de Reutilização, segundo a SOFTEX (2011c), é definir o tipo de ativo que será reutilizado, atribuindo papéis aos recursos, realizando um planejamento para todo o processo e definindo critérios de avaliação aos quais os ativos serão submetidos, para que estes possam fazer parte da biblioteca. Periodicamente, esta biblioteca deve ser revisada, verificando se os ativos presentes estão de acordo com o propósito central da Gerência de Reutilização.

O Processo GRU se relaciona diretamente também com outros processos do MPS como, por exemplo, o processo de Gerência de Projetos, que apoia e dá suporte ao planejamento que deve ser feito na fase inicial, e a Gerência de Configuração, que dá suporte as novas versões dos ativos. Além destes, há também os processos de Verificação, Garantia da Qualidade, entre outros [SOFTEX, 2011a]. Em contra partida,

o Processo GRU pode fornecer suporte a outros processos do MPS, como a Gerência de Configuração, em que o controle de ativos e o armazenamento das versões utilizam como catálogo a biblioteca de ativos reutilizáveis [SOFTEX, 2011c].

O Processo GRU do MPS possui cinco resultados esperados, que visam auxiliar as organizações que forem implementar o nível E do modelo de maturidade a atenderem os requisitos da avaliação. São eles [SOFTEX, 2011c]:

- **GRU 1** – Uma estratégia de gerenciamento de ativos é documentada, contemplando a definição de ativo reutilizável, além dos critérios para aceitação, certificação, classificação, descontinuidade e avaliação de ativos reutilizáveis.

A organização define o escopo de uso dos ativos reutilizáveis, estipulando quais ativos são passíveis de reutilização e em que ponto do processo essa reutilização ocorrerá. De acordo com o tipo de ativo, a biblioteca será organizada de maneiras diferentes, sempre tendo em mente privilegiar o procedimento de busca e seleção. O planejamento e execução do Processo GRU é então realizado, avaliando custos, estabelecendo marcos, alocando os recursos necessários, definindo os papéis presentes e os critérios de avaliação aos quais os ativos devem ser submetidos para serem armazenados na biblioteca. Ao ser aceito na biblioteca, o ativo deve ser classificado visando facilitar sua busca e seleção.

- **GRU 2** – Um mecanismo de armazenamento e recuperação de ativos reutilizáveis é implantado.

O mecanismo a ser utilizado para recuperar e utilizar o ativo é definido de acordo com as informações contidas na documentação. O mecanismo utilizado deve ser compatível com o que foi planejado, dado que, idealmente, o processo de planejamento considerou a melhor maneira de localizar um ativo para a sua utilização. A implantação desse mecanismo significa disponibilizar a biblioteca para uso.

- **GRU 3** – Os dados de utilização dos ativos reutilizáveis são registrados.

A biblioteca deve gerar um registro de utilização para mapear quem utiliza qual ativo quando for utilizado por algum consumidor/usuário. Essa relação entre ativo e

usuário é importante, pois caso ocorra algum tipo de modificação no ativo, o usuário deve ser notificado, alertando a mudança que ocorrerá no ativo. Além da notificação, esse tipo de informação pode fornecer também o número de usuários que dispõem daquele recurso, para verificar a longo prazo se, de fato, aquele ativo deve ou não fazer parte da biblioteca de ativos.

- **GRU 4** – Os ativos reutilizáveis são periodicamente mantidos, segundo os critérios definidos, e suas modificações são controladas ao longo do seu ciclo de vida.

Resultado esperado que reitera a ligação existente entre o Processo GRU e os processos de Gerência de Configuração e de Garantia de Qualidade, ressaltando a importância de se armazenar na biblioteca todas as versões que o ativo possa ter.

- **GRU 5** – Os usuários de ativos reutilizáveis são notificados sobre problemas detectados, modificações realizadas, novas versões disponibilizadas e descontinuidade de ativos.

A partir do controle de utilização dos ativos, seus respectivos usuários são notificados, caso algum tipo de modificação tenha sido feita ou venha a ser realizada. Isso mantém os consumidores atualizados a respeito do ativo que estão adquirindo, sendo possível prevenir-se sobre possíveis mudanças.

Capítulo 3. Adaptação da biblioteca Brechó para gestão de ativos de Software

O escopo deste trabalho baseia-se em adaptar uma biblioteca de componentes e serviços para que esta possa funcionar também como um ferramental de governança para apoiar a gerência de ativos de software de uma organização consumidora de tecnologia. Para facilitar o entendimento e também diferenciar um ativo de software dos demais artefatos presentes nesse tipo de biblioteca, define-se que todo artefato que seja um ativo de software deve pertencer a um grupo denominado SECOGov (*Software Ecosystem Governance*). Dessa forma, os ativos cadastrados na biblioteca pertencentes a uma organização que deseja realizar uma gestão sobre eles devem obrigatoriamente estar inseridos nesse grupo de artefatos. A partir deste objetivo, um conjunto de requisitos foi identificado para garantir minimante que uma organização consumidora seja capaz de gerenciar seus ativos. Os requisitos são:

- **R1** – Coexistência: Os ativos de software devem coexistir com os demais artefatos presentes na biblioteca de componentes e serviços, não influenciando e nem sofrendo influência direta do seu funcionamento;
- **R2** – SECOGov: Todos os ativos devem ter minimamente um conjunto de informações que possibilite a gestão por parte das organizações. Dessa forma, ficou decidido que os seguintes dados devem ser solicitados no ato de cadastro de um ativo: Fornecedor (*Vendor*), Natureza (*Nature*), Tecnologia (*Technology*), Maturidade (*Maturity*), Data (*Date*), Ativos Produzidos (*Produced Asset*), Versões (*Releases*), Licenças (*Licenses*), Usuários (*Users*) e URI (*Uniform Resource Identifier*);
- **R3** – Campo Data: A data do ativo de software do grupo SECOGov deve ser sempre menor ou igual à data atual do cadastro do ativo;
- **R4** – Campo Ativo Produzido: Obrigatoriamente, um tipo de resultado deve ser associado a um ativo de software do grupo SECOGov por meio do campo Ativos Produzidos. Esse resultado pode ser entendido como o produto que é gerado ao se utilizar esse ativo na organização, por exemplo, o aplicativo de e-mail *Outlook* que, ao ser utilizado, gera (produz) um sistema de e-mail, que corresponde ao ativo produzido;

- **R5** – Versões: Deve-se permitir que uma ou mais versões de um ativo de software sejam cadastradas;
- **R6** – Licenças: Deve-se informar a quantidade total de licenças disponíveis de cada versão de cada ativo de software;
- **R7** – Usuários: Deve-se informar quem são os usuários que utilizam as licenças cadastradas de cada ativo de software;
- **R8** – Log: Todo ativo de software do grupo SECOGov deve possuir um histórico de utilização, a fim de verificar a sua frequência de uso mensal, por meio do campo URI. Nesse campo, deve-se informar o local onde o arquivo de *log* se encontra, assim como o nome do arquivo e extensão, que deve ser no formato .txt;
- **R9** – Pesquisa: O sistema deve permitir a realização de buscas nos ativos de software do grupo SECOGov. A busca pode ser feita por meio do nome ou tecnologia utilizada pelo ativo de software ou pelo fornecedor, ou ainda por um usuário que o utiliza;
- **R10** – Gráficos: Criação de gráficos com foco em diferentes áreas de análise:
    - **R10.1** – Gráfico de Pizza: Número de licenças utilizadas e não utilizadas dos ativos de software de um fornecedor. Por se tratar de um gráfico não temporal e que deseja-se apresentar apenas dois tipos de informação (usa ou não usa), definiu-se o tipo de gráfico de pizza para esse caso;
    - **R10.2** – Gráfico de Pizza: As licenças utilizadas e não utilizadas das versões do ativo de software. Como no caso anterior, a mesma lógica para definir o tipo gráfico foi utilizada;
    - **R10.3** – Gráfico de Barras: A utilização de todos os ativos de software cadastrados que produzem o mesmo tipo de ativo. Nesse caso, a intenção é permitir a percepção de utilização dos ativos de software, sendo o gráfico de barras aquele que se mostrou mais apropriado para a análise;
    - **R10.4** – Gráfico de Grafo: ECOS, onde o nó central é a tecnologia utilizada e ao redor são dispostos os ativos de software que utilizam essa tecnologia.

- **R11** – Listagem de Componentes: Na listagem de componentes, as informações do fornecedor, natureza e tecnologia devem ser exibidos também para os ativos de software do grupo SECOGov;
- **R12** – Detalhamento de Componente: Deve ser possível verificar os dados cadastrais de um ativo de software;
- **R13** – Edição de Componente: Deve ser possível editar as informações do ativo de software cadastrado.

Como arquitetura da solução deste trabalho, tem-se o esquema da Figura 5. Inicialmente, tem-se uma biblioteca de componentes e serviços que é adaptada e torna possível também o armazenamento de aplicações de software, enquadrando-se dessa forma na nova realidade das organizações consumidoras que adquirem componentes, serviços e aplicações na busca por uma solução de um dado problema. Além disso, alguns *inputs* e *outputs* serão necessários para que a ferramenta possa auxiliar a organização consumidora na gestão de seus ativos de software. Um dos *inputs* diz respeito ao mecanismo de busca voltado para os conceitos da SAM, no qual quatro filtros são disponibilizados: nome do ativo de software cadastrado, fornecedor desse ativo de software, tecnologia utilizada no ativo de software e por fim, listagem de ativos que um usuário está utilizando.

Outro *input* está relacionado ao grupo SECOGov, que contém as informações dos ativos de software e é o responsável por definir se o ativo de software que está sendo inserido na biblioteca é ou um não ativo pertencente ao grupo SECOGov. Um terceiro *input* está vinculado ao histórico de utilização (*log*) de um ativo de software ao longo do tempo, no qual a informação de qual ativo é produzido serve de insumo para a elaboração de uma das formas gráficas disponíveis para realizar a gestão dos ativos de software da organização consumidora. Como *output*, tem-se quatro possíveis análises em forma de gráfico: a primeira forma tem foco em um fornecedor que pode ser de um ou mais ativos de softwares; a segunda tem foco voltado para as versões que um ativo possa vir a ter; a terceira tem foco em avaliar os ativos que são produzidos ao longo do tempo por um determinado ativo de software; e, por fim, a quarta tem foco em avaliar o ECOS que se forma através da utilização de um conjunto de ativos com a mesma tecnologia de software central.

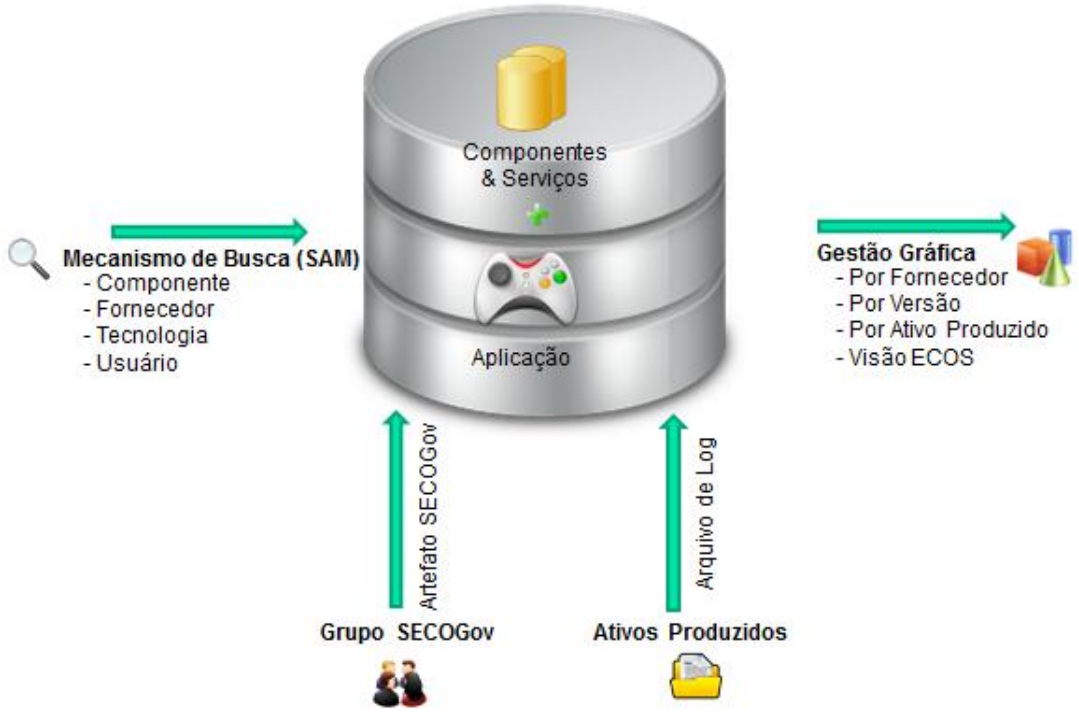

Figura 5 - Arquitetura da Solução

3.1 Biblioteca Brechó

Tendo por base que a organização pretende usar esse ferramental para gerenciar os ativos de software que são adquiridos de terceiros, considera-se que todo artefato inserido nessa biblioteca, que tiver como um dos parâmetros o item "Fornecedor", será considerado ativo de software e, por consequência, estará no grupo SECOGov. Nesse sentido, o escopo deste trabalho final de graduação é expandir a biblioteca Brechó de forma a permitir que a mesma atue tanto como repositório de componentes e serviços como uma ferramenta de gerência de ativos de software de uma organização, contemplando ainda os conceitos de ECOS e os requisitos do SECOGov.

A biblioteca Brechó do Grupo de Reutilização de Software da COPPE/UFRJ consiste em um sistema Web para gerência de componentes e serviços de software que fornece mecanismos de documentação, publicação, armazenamento, busca, recuperação, controle de versão e evolução, gerenciamento dos usuários e de licenças, entre outros [Werner *et al.*, 2007]. De acordo com Marinho *et al.* (2009), a Brechó contém um mecanismo especial de documentação e publicação que leva em consideração um conceito flexível de componentes, no qual todos os possíveis artefatos produzidos durante o desenvolvimento são armazenados (e.g., código-fonte, arquivo binário, especificações etc.). Segundo Lopes *et al.* (2007), essa flexibilidade influencia diretamente nos mecanismos de busca e recuperação dos componentes. A busca é feita

através de categorias e/ou palavras-chave, e ao encontrar o componente, este é disponibilizado para *download*, de acordo com as políticas estabelecidas.

A biblioteca é organizada internamente em cinco níveis (Figura 6), sendo o primeiro denominado **Componente**, no qual os artefatos armazenados são representados conceitualmente. O segundo nível é **Distribuição**, que representa o conjunto de funcionalidades relacionadas aos artefatos armazenados, que podem ser obtidas pelos usuários. O terceiro nível é **Release**, que representa, temporalmente, as diferentes versões que um componente pode ter na biblioteca. No quarto nível estão **Pacotes** e **Serviços**, sendo Pacote responsável por possibilitar o agrupamento de diversos artefatos para atender a necessidade de um usuário, e Serviço, que possibilita a reutilização de uma *release* na forma de serviços Web. Por fim, no último nível, tem-se **Licença**, que define os direitos e deveres sobre um artefato ao obtê-lo [Werner *et al.*, 2007].

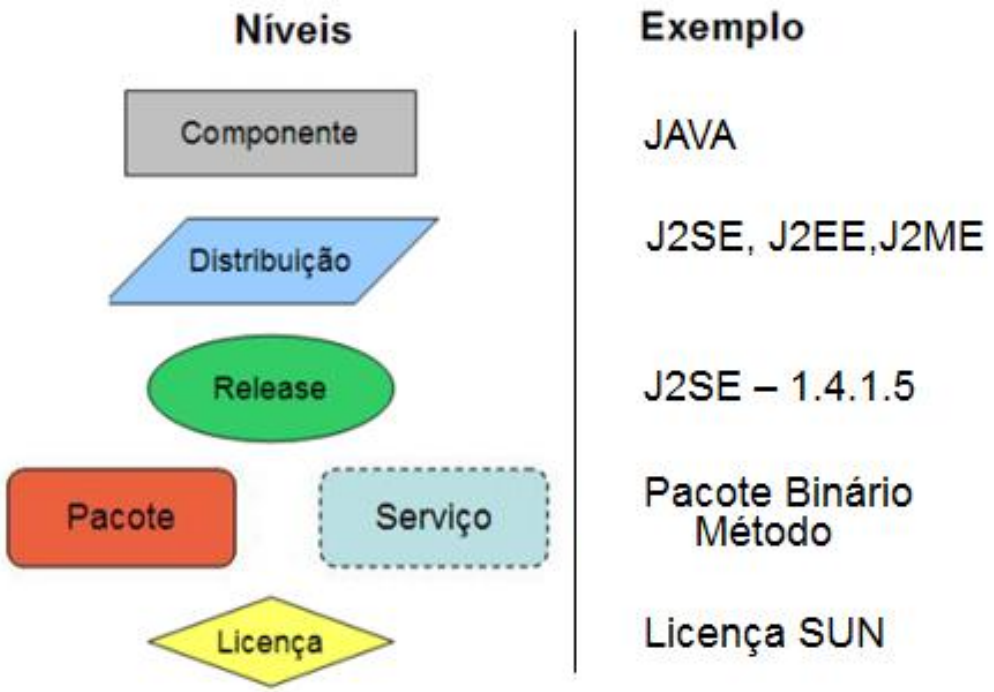

Figura 6 - Estrutura da biblioteca Brechó [Werner et al., 2007]

A biblioteca possui ainda um mecanismo de controle de acesso e registro de utilização através de autenticação, importante para a etapa de utilização de uma biblioteca uma vez que possibilita a geração de informação tanto para os usuários como os fornecedores por *feedback* e estatísticas, além de possibilitar serviços tarifados, entre

outros [Marinho *et al*, 2009]. A Brechó possui alguns tópicos de pesquisa que são: [Brechó, 2013]:

- Desenvolvimento Baseado em Componentes;
- Documentação, Empacotamento e Avaliação de Componentes;
- Busca, Recuperação e Tarifação de Componentes;
- Gerência de Configuração de Software;
- Desenvolvimento Baseado em Serviços;
- Composição de Serviços e Arquiteturas Orientadas a Serviço;
- Mercados de Componentes;
- Rede Social.

A plataforma Brechó é desenvolvida em Java, tendo como servidor o Apache Tomcat. Além disso, utiliza o framework Hibernate para conexão ao banco MySql e o Struts como manipulação dos dados no lado do servidor. Utiliza-se também um conjunto de plug-ins e recursos como JavaScript e CSS (Cascading Style Sheets) para deixar as páginas JSP (JavaServer Pages) mais interativas e com melhor navegabilidade.

3.2 SECOGov

A fase inicial deste trabalho consistiu em analisar a plataforma Brechó e verificar todos os seus recursos e funcionalidades, buscando a melhor maneira para que a adaptação fosse feita sem afetar diretamente a estrutura existente, assim como interferir no modelo de negócio em vigor. A Brechó possui um conceito de Categoria, que permite agrupar e classificar componentes os quais, pela flexibilidade do conceito, representam os ativos de software, i.e., componentes, serviços e aplicações, conforme discutido no Capítulo 2. Cada categoria está associada a um *template* de formulário, que possui uma série de informações as quais os componentes associados a essa categoria devem obrigatoriamente informar, o que justifica este componente fazer parte desta. Sendo assim, dois novos requisitos fizeram-se necessários no presente trabalho, uma vez que para cadastrar um componente novo na plataforma, obrigatoriamente esses dois recursos devem pré-existir. Os requisitos são:

- **Req 14** – *Template*: Criar um *template* contendo as informações necessárias para que seja possível realizar a gestão dos ativos de software da organização, considerando os conceitos da biblioteca a ser utilizada;

- **Req 015** – Categoria: Criar uma categoria para o grupo SECOGov, onde serão inseridos os ativos de software da organização, provendo como formulário o *template* SECOGov;

Além disso, o banco de dados da aplicação é representado pelo seguinte modelo relacional (Figura 7):

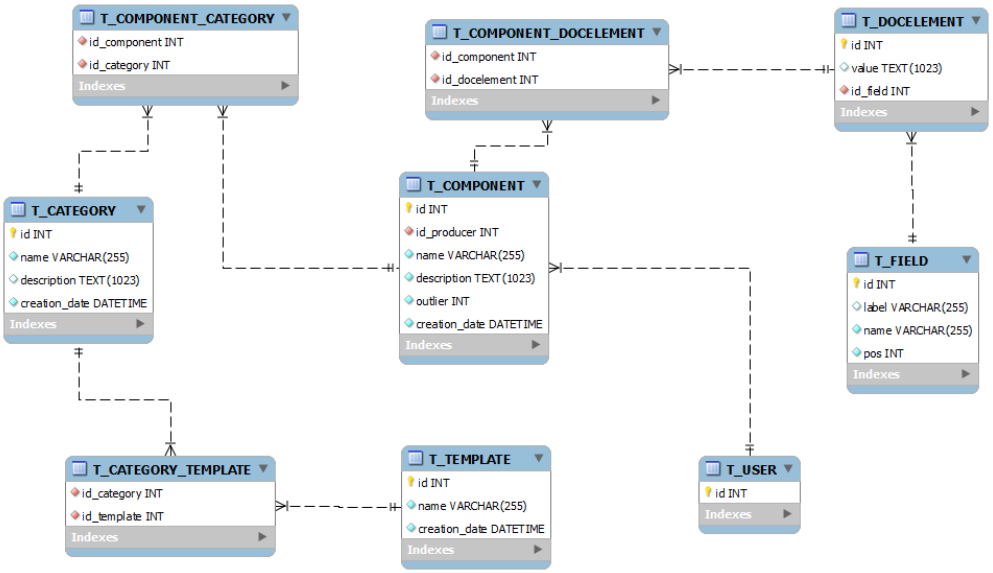

Figura 7 - Modelo Entidade-Relacionamento

Um componente, então, está associado a uma categoria que, por sua vez, tem a ela atrelada um *template* de formulário. O componente possui elementos de documentação (*docelements*), que são os valores preenchidos no formulário no ato de cadastrar um componente na plataforma. A recuperação dos *docelements* de um componente é feita na tabela *t_component_docelement* através da chave primaria (*id*) do componente, sendo retornado uma lista de *docelements* no resultado da busca no banco de dados.

Analisando os requisitos, verificou-se que o primeiro passo seria modelar um *template* a ser utilizado nos ativos SECOGov. O *template*, escrito em linguagem XML (*eXtensible Markup Language*), foi modelado inicialmente da seguinte forma:

1. Fornecedor: Como a quantidade de possíveis fornecedores de ativos de software é ilimitada, decidiu-se utilizar um campo de texto livre;
2. Natureza: Utilizou-se uma *combobox* para informar o tipo de natureza. Definiu-se, inicialmente, quatro possíveis valores para a natureza, que podem ser

atualizados ao longo do tempo por modificação do *template*: Adquirido, Não Adquirido, Desenvolvido ou Em Evolução;

3. Tecnologia: Utilizou-se uma *combobox* para informar o tipo de tecnologia. Definiu-se, inicialmente, cinco possíveis valores para as tecnologias utilizadas, que podem ser atualizados ao longo do tempo por modificação do *template*: Sistema Operacional Móvel, Sistema Operacional, Armazenamento em Nuvem, Gerenciamento de Ativos de Software ou E-mail Coorporativo;
4. Maturidade: Utilizou-se uma *combobox* para informar o tipo de Maturidade. Definiu-se, inicialmente, sete possíveis valores para as tecnologias utilizadas, que podem ser atualizados ao longo do tempo por modificação do *template*: Embrionária, Emergente, Adolescente, Pouco Difundida, Muito Difundida, Legado ou Obsoleto;
5. Data: Utilizou-se texto livre, no qual será inserido a data em que o ativo de software foi adquirido, contendo o seguinte *template*: DD/MM/AAAA;
6. Ativo Produzido: Utilizou-se texto livre, no qual será informado o tipo de resultado que o ativo de software produz ao ser utilizado;
7. Versão: Utilizou-se texto livre, no qual será informada a versão do ativo de software;
8. Licença: Utilizou-se texto livre, no qual será informada a quantidade de licença que a organização possui deste ativo de software;
9. Usuários: Utilizou-se texto livre, no qual serão informados os usuários que utilizam o ativo de software. Os usuários devem vir separados obrigatoriamente por ponto e vírgula (;);
10. URI: Utilizou-se texto livre, no qual será informado o endereço onde o arquivo de *log* de utilização do ativo de software se encontra.

Com o *template* configurado, criou-se uma categoria denominada SECOGov e um ativo fictício para testar a frequência de uso da biblioteca e do novo *template*. Ao realizar o cadastro, entretanto, alguns problemas foram encontrados:

1. Os campos Fornecedor, Data e Ativos Produzidos foram definidos, em nível de *template*, como requeridos, ou seja, o preenchimento com algum valor valido é obrigatório, porém, no ato de cadastrar o componente, a biblioteca aceitou que nenhum valor fosse inserido nos campos;

2. O campo Data aceitava letras e números, além do formato estar fora dos padrões de data, aceitando mais caracteres do que realmente precisa;
3. Para todo componente cadastrado na biblioteca, são gerados automaticamente uma versão *default* e um pacote (com licença) *default*, sendo, neste caso, desnecessário estas informações estarem associadas ao *template* da categoria SECOGov.

Para resolver o problema 1, verificou-se no código-fonte onde os valores informados no formulário eram recebidos. No *template*, definiu-se o campo como requerido, através de um parâmetro *booleano* (verdadeiro = obrigatório, falso = opcional), ou seja, ou um campo é ou não requerido. Para corrigir o erro, para cada valor recebido do formulário, passou-se a verificar o valor do parâmetro, que define se o elemento é ou não de preenchimento obrigatório. Nos casos em que o valor do parâmetro era verdadeiro, verifica-se se o campo do formulário retorna algum valor válido. Caso afirmativo, o cadastro do componente prossegue, caso contrário, emite-se uma mensagem de erro, informando que o preenchimento do campo é obrigatório.

Para o caso especial do campo Data (problema 2), quando o valor recebido pelo formulário é do campo Data, o valor é passado por uma verificação para garantir que o valor informado atende as seguintes exigências:

- A data deve estar no formato DD/MM/AAAA e são aceitos apenas números;
- A data deve obedecer às regras de um calendário padrão, levando em conta os anos bissextos e os meses que terminam no dia 30 e 31;
- A data deve ser inferior à data atual do sistema.

Por fim, chegou-se a conclusão que, para resolver o problema 3, devia-se remover do *template* os campos destinados a Versão e Licença do componente. Entretanto, o campo Usuários perderia sentido, uma vez que o usuário está vinculado não apenas a um componente, mas também à licença de uma versão de um componente e, caso permanecesse no *template*, estaria vinculado apenas ao componente. Dessa forma, o campo Usuários foi removido do *template*. O *template* final do SECOGov ficou configurado como mostra a Figura 8.

Quando se cadastra um novo componente, automaticamente, a Brechó redireciona para a página de Listagem de Componentes. Nessa página, um dos requisitos listados é fornecer alguns dados dos artefatos que são do grupo SECOGov,

sendo estes dados o Fornecedor do ativo de software, bem como Natureza e Tecnologia utilizadas. Os campos desejados foram então criados, tanto para as versões em português e em inglês da biblioteca. Entretanto, as informações necessárias não estavam armazenadas na tabela *t_component*, mas na tabela *t_docelement* e, por isso, a exibição do valor na tela necessitava de tratamento em nível de código-fonte no servidor da aplicação.

```
<component>
    <text label="Vendor" name="vendor" size="30" required="true"/>

    <select label="Nature" name="nature" size="3" multiple="true">
        <option label="Acquired" value="Acquired" selected="true"/>
        <option label="Not Acquired" value="Not Acquired"/>
        <option label="Developed" value="Developed"/>
        <option label="Under Evaluation" value="Under Evaluation"/>
    </select>

    <select label="Technology" name="technology" size="3" multiple="true">
        <option label="Mobile Operating System" value="Mobile Operating System" selected="true"/>
        <option label="Operating System" value="Operating System"/>
        <option label="Cloud Storage" value="Cloud Storage"/>
        <option label="Software Asset Management" value="Software Asset Management"/>
        <option label="Corporate Email" value="Corporate Email"/>
    </select>

    <select label="Maturity" name="maturity" size="3" multiple="true">
        <option label="Embryonic" value="Embryonic" selected="true"/>
        <option label="Emerging" value="Emerging"/>
        <option label="Adolescent" value="Adolescent"/>
        <option label="Early Mainstream" value="Early Mainstream"/>
        <option label="Mature Mainstream" value="Mature Mainstream"/>
        <option label="Legacy" value="Legacy"/>
        <option label="Obsolete" value="Obsolete"/>
    </select>

    <text label="Date" name="date" size="30" required="true"/>

    <text label="Produced Assets" name="producedAssets" size="30" required="true"/>

    <text label="URI" name="uri" size="30" />

</component>
```

Figura 8 – Modelo de Template do SECOGov

Conforme mencionado anteriormente, os valores dos campos do formulário de um componente são armazenados no banco de dados em uma tabela denominada *t_docelement* e a ligação do componente aos seus *docelements* é realizada pela tabela *t_component_docelement*. Todo componente da plataforma possui uma lista de *docelements* vinculados a ele e, para exibir os valores, deve-se acessar a sua lista de *docelements* e retornar o valor, a partir do *id* do componente.

A lógica utilizada para resolver a questão foi a seguinte: é feito acesso ao banco de dados que retorna todos os componentes cadastrados no mesmo. Para cada componente retornado, busca-se no banco os seus respectivos *docelements* e armazena-se a informação em uma estrutura de dados MAP (um objeto que mapeia chaves para valores), na qual o valor da chave é o *id* do componente e os valores associados são os

docelements. Cada chave do MAP é processada iterativamente em cada valor de cada chave. Para cada valor, que no caso é um elemento do *docelement* do componente, verifica-se se o nome do campo deste elemento é "vendor", "nature" ou "technology" (nome dos campos no *template*), informações que devem ser exibidas na página de Listagem de Componentes.

Caso o elemento seja de um dos tipos descritos, o elemento é inserido em um novo MAP, onde a chave é o próprio componente e o valor dessa chave é o valor preenchido no formulário no ato de cadastrar o novo componente na plataforma. Dessa forma, para cada componente listado, itera-se sobre o MAP contendo o componente e o seu respectivo valor, imprimindo na tela apenas o valor do *vendor* ou *nature* ou *technology*. Para os componentes cadastrados na biblioteca, que não são da categoria SECOGov, nada é exibido.

Ainda na tela de listagem de componentes, verificou-se a existência de três funcionalidades que estão listadas nos requisitos: Detalhar, Editar e Excluir. Com exceção da primeira, que necessitou ser ajustada conforme exibido pela Figura 9, as outras duas funcionalidades já estavam funcionando corretamente na biblioteca. A funcionalidade Detalhar Componente não era capaz de exibir os campos do grupo SECOGov e, após o ajuste realizado (onde o trecho de código comentado era o anteriormente utilizado), passou a funcionar corretamente. O ajuste não se restringe apenas ao caso do *template* do grupo SECOGov, servindo também para qualquer outro *template* utilizado.

```
<!--<custom:inputValued name="docElem" disabled="true"/>-->
        <bean:write name="docElem" property="value"/>
```

Figura 9 – Ajuste realizado no código da funcionalidade Detalhar Componente

Na Brechó, estava disponibilizado um sistema de pesquisa de componentes, porém a lógica utilizada não atendia aos requisitos especificados neste trabalho. No sistema de busca, o componente cadastrado deveria ter o status de ACEITO, dado pelo administrador da biblioteca, e era *case sensitive*, ou seja, uma busca pelo nome "Componente" teria resultado diferente de uma busca pelo nome "componente". Dessa forma, para não alterar a lógica existente, decidiu-se criar um novo mecanismo de busca que contemplasse as necessidades do SECOGov. Esse mecanismo de busca está presente na página de Listagem de Componentes, possuindo, além do campo onde é inserida a palavra a ser buscada, um filtro do tipo *combobox*, que define se a palavra é o

nome do ativo de software, o nome do fornecedor, a tecnologia utilizada ou um usuário da biblioteca. No novo mecanismo, não é verificado se o *status* do componente está como aceito ou não.

Além disso, as buscas passaram a não ser *case insensitive*, ou seja, uma busca por "Componente" ou "componente" retorna sempre o mesmo resultado. Existe ainda o recurso de buscar por uma tecnologia, por exemplo, ao inserir apenas parte do seu nome. Isso é possível, pois no mecanismo de busca desenvolvido, ao invés de buscar no banco de dados a palavra digitada, buscam-se os elementos que possuem parcialmente ou em sua totalidade a palavra que foi digitada. Nesse caso, ao digitar "**era**" no campo de busca e definir o filtro como Tecnologia, obtém-se como resultados todos os ativos que tiverem como tecnologia o valor de "Mobile Op**era**ting System".

Para resolver a questão das versões, licenças e seus respectivos usuários, utilizou-se parte da estrutura disponibilizada na biblioteca e fez-se ajustes pontuais para atender aos requisitos do grupo SECOGov. Na tela de cadastro de nova licença, foi inserido o campo Quantidade Total e o campo Usuários, alterando, assim, a classe *Component* e *Package*, como mostra a Figura 10. No campo Quantidade Total, deve-se informar o número total de licenças disponíveis para a versão em questão, ao passo que, no campo Usuários, deve-se listar todos os usuários que utilizam as licenças, separando-os por ponto e vírgula (;). Caso não seja utilizado o ";", será considerado apenas como um usuário.

A biblioteca Brechó, antes da adaptação para contemplar o grupo SECOGov, dispunha de *plug-ins* para elaboração de gráficos. Há na biblioteca um conceito de *e-commerce*, no qual um componente pode ser negociado e, ao clicar no ícone da opção "Análise" na página Listar Componentes, é possível fazer uma análise de mercado do componente, sendo exibido um gráfico como resultado dessa análise.

Como um dos requisitos para o grupo SECOGov é a elaboração de mecanismos gráficos para um conjunto de análises a serem realizadas, ao invés de adaptar a página de análise de mercado do *e-commerce*, foi decidido criar uma nova página, de *layout* bem similar, na qual, além da limitação temporal, estaria também presente o tipo de análise a ser realizada, segundo a especificação dos requisitos. Dessa forma, existe na biblioteca duas possíveis análises de mercado: uma que atende ao caso dos componentes da categoria SECOGov e outra para os demais componentes.

Para realizar esse tipo de distinção e, ao clicar no ícone "Análise", ir para a análise de mercado correspondendo ao tipo de componente (SECOGov ou não), foi

necessário realizar três alterações, sendo a primeira em nível de banco de dados, a segunda em nível de página JSP e a terceira em nível de código-fonte. A alteração no banco de dados ocorreu na tabela *t_component*, adicionando uma nova coluna denominada *secogov*, do tipo *booleano*, assim como a classe *Component* também sofreu alteração (Figura 10). A intensão de adicionar essa nova coluna é possibilitar que um componente do tipo SECOGov seja facilmente identificado e diferenciado dos demais.

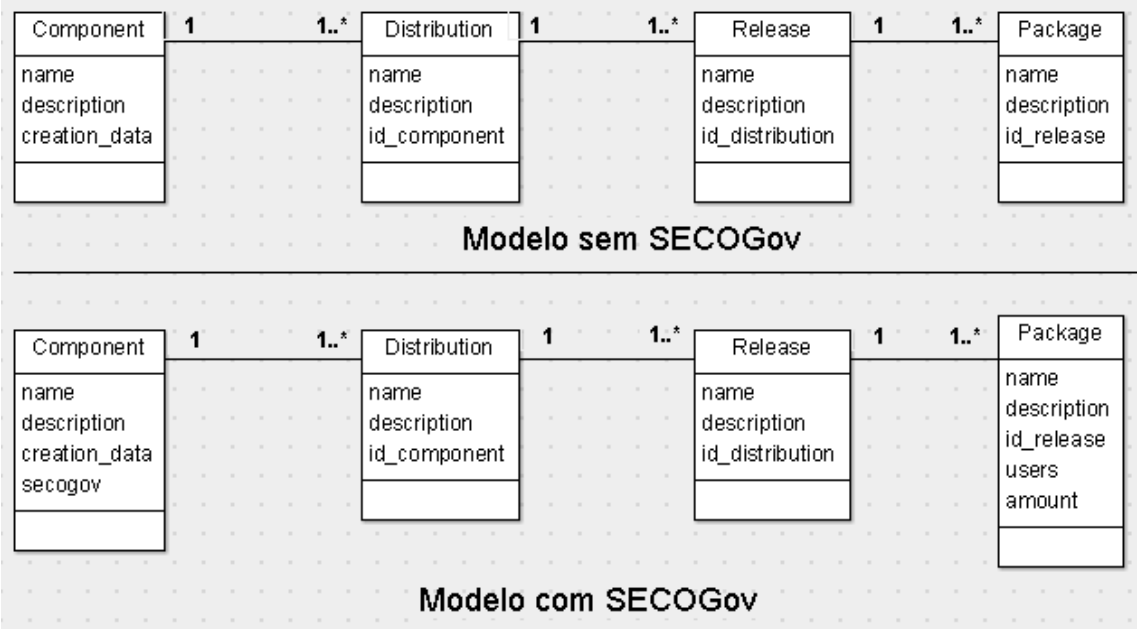

Figura 10 - Alteração realizada nas classes Component e Package

Em nível de página JSP, a alteração foi feita no campo "Análise". Antes de inserir o link para acessar a página de Análise, o componente é verificado a fim de analisar se o campo SECOGov do componente é verdadeiro ou falso. Caso seja verdadeiro, trata-se de uma ativo de software (SECOGov), logo, insere-se o link que direcionará para a análise de mercado que contemple os gráficos utilizados no grupo SECOGov. Caso contrário, é inserido o link que direcionará para a análise de mercado existente na biblioteca. Por fim, como tratam-se de regras de negócio totalmente diferentes, a lógica da geração de cada gráfico não poderia ser tratada por um mesmo método e/ou *action* do *framework* Struts. Nesse caso, um novo método foi criado (Figura 11), mantendo a mesma base lógica, porém, adaptando a lógica de formação dos gráficos para atender a demanda deste.

```
<td class="list" style="text-align:center">

    <logic:equal name="component" property="secoGov" value="true">
        <html:link forward="componentAnalisysSECOGov" paramId="id" paramProperty="id" paramName="component">
            <img src="../img/analisys.png" border="0" alt='<bean:message key="text.analisys"/>' />
        </html:link>
    </logic:equal>

    <logic:equal name="component" property="secoGov" value="false">
        <html:link forward="componentAnalisysSales" paramId="id" paramProperty="id" paramName="component">
            <img src="../img/analisys.png" border="0" alt='<bean:message key="text.analisys"/>' />
        </html:link>
    </logic:equal>
</td>
```

Figura 11 - Diferenciação de fluxo para acesso a área gráfica

Apesar de já possuir um *plug-in* para elaboração de gráficos, notou-se que a lógica utilizada para gerar os gráficos para o SECOGov não seria atendida com o *plug-in* até então utilizado. Procurou-se então um *plug-in* que atendesse aos modelos gráficos exigidos pelo grupo SECOGov e decidiu-se pela utilização do *plug-in* JFreeChart, para a elaboração dos gráficos de barra e pizza, e do PREFUSE, para a geração gráfico em grafo para exibir a formação do ECOS ao redor da tecnologia do ativo de software.

A montagem de cada gráfico começa com a recuperação dos *docelements* do ativo de software alvo da análise. Acessa-se então a tabela *t_docelement*, informando qual o *id* do ativo e obtendo, como retorno, a lista de *docelements* vinculada a este ativo. Em seguida, armazena-se essa informação em uma estrutura de dados de conjunto denominado SORTEDSET (conjunto que fornece total ordenação de seus elementos). Para iterar sobre os elementos do *docelement*, criou-se um MAP onde foi utilizado o ativo de software como chave e os seus *docelements* como valor. Além disso, traz-se também a listagem de todos os artefatos cadastrados na biblioteca, assim como todas as licenças e versões. A lógica utilizada para a geração de cada gráfico foi:

1. **Número de licenças utilizadas e não utilizadas dos ativos de software (componentes na Brechó) de um fornecedor**

Como esse gráfico é gerado a partir do cálculo do número de licenças utilizadas de um determinado fornecedor, deve-se inicialmente identificar de que fornecedor se trata. Para isso, itera-se sobre os *docelements* do componente a partir do qual acionou-se o mecanismo de análise gráfica para descobrir quem é o seu fornecedor. Tendo descoberto o fornecedor, itera-se sobre a lista que contém todos os componentes, que foi pré-carregada para descobrir aqueles que possuem o mesmo fornecedor selecionado anteriormente. Quando um componente possui o mesmo fornecedor, itera-se sobre as

lista que contém todas as licenças a fim de encontrar aquelas vinculadas ao componente. Encontradas as licenças do componente em análise, recupera-se o número total de licenças que a organização possui, bem como o número de usuários que a utiliza.

Dessa forma, garante-se que todos os componentes cadastrados na biblioteca e suas respectivas licenças serão analisados, inclusive o componente utilizado para descobrir qual fornecedor analisar. Por fim, soma-se o número total de licenças e o número total de usuários que as utilizam. O número total de licenças utilizadas do fornecedor corresponde ao número total de usuários que a utilizam em relação ao número total de licenças cadastradas, sendo o número total de licenças não utilizadas o seu complemento. Garante-se também que, caso uma licença tenha mais usuários do que o número disponível, o valor apresentado será graficamente maior do que 100%, indicando a necessidade de adquirir licenças ou diminuir o número de usuários.

2. **As licenças utilizadas e não utilizadas das versões do ativo de software (componentes na Brechó)**

Começa-se iterando a lista de todas as licenças presentes na biblioteca para encontrar as licenças que pertencem ao componente a partir do qual acionou-se o mecanismo de análise gráfica. Para cada licença pertencente a esse componente, soma-se o número total de licenças que a organização possui e armazena-se em um MAP a versão a que essa licença está vinculada, como *key*, assim como a quantidade de licenças que são utilizadas e as licenças que não são utilizadas, como *value*, separando-as por ";". Itera-se então sobre este MAP que foi populado, calculando, para cada versão, qual foi o percentual de licenças utilizadas ou não. Por essa abordagem, consegue-se dividir o gráfico de pizza em partes iguais pelo número total de versões que o componente possui, ou seja, caso um componente possua quatro versões vinculadas a ele, o gráfico será divido em quatro partes iguais e, dentro de cada uma das fatias, a análise de licenças utilizadas e não utilizadas para cada versão será realizada e exibida.

3. **A utilização de todos os ativos de software (componentes na Brechó) cadastrados que possuem o mesmo ativo produzido em comum**

Nesse gráfico, utilizam-se as informações presentes nos arquivos de *log* informados no campo URI do ativo de software. O ideal seria integrar a biblioteca aos ativos de software existentes dentro da organização e, por algum mecanismo de controle

interno, obter a frequência de utilização dos componentes. Para o escopo deste trabalho, entretanto, desconsiderou-se este mecanismo, provendo uma interface via campo URI como endereço para encontrar o arquivo de *log* do ativo de software, a ser gerado pela organização. O arquivo de *log* utilizado neste trabalho é fictício, elaborado de forma manual e aleatória, com o único objetivo de exemplificar o uso desse recurso.

A lógica para a formação desse arquivo é: cada linha de *log* contém a informação de quando o ativo de software foi utilizado no formato de data completa (ano, mês, dia do mês, dia da semana, hora, minuto e segundo) e qual usuário utilizou o ativo. Cada informação é separada pelo caractere "|" e, ao término de cada linha, o caractere ";" é inserido, informando o final da linha de *log*. Inicialmente, pensou-se em inserir ao final de cada linha a informação do *id* do componente ao qual o *log* pertence. No entanto, ao colocar na URI não só o endereço do local do arquivo, mas também o nome do arquivo vinculado ao ativo de software, a informação se tornou desnecessária.

Dessa forma, o primeiro passo é encontrar qual o ativo produzido pelo ativo de software. Para isso, itera-se sobre os *docelements* do componente a partir do qual aciona-se o mecanismo de análise gráfica e descobre-se sobre qual ativo produzido deve-se montar o gráfico de análise. De posse desta informação, itera-se sobre a listagem pré-carregada que possui todos os ativos de software cadastrados na biblioteca, armazenando em um MAP tanto do ativo de software, como *key*, como seus respectivos *docelements*, como *value*. Como esse gráfico é baseado na frequência temporal de utilização do ativo produzido, na página JSP de geração de gráficos, solicita-se o ano e período, por mês, que se deseja analisar a frequência de uso. Dessa forma, o próximo passo é calcular justamente o período, em meses, de quando está se analisando o arquivo de *log*.

Iterando sobre o MAP que foi produzido e que contém como chave o ativo de software e, como valor, os seus *docelements*, obtém-se todos os ativos de software que possuem o mesmo tipo de ativo produzido do componente a partir do qual acionou-se o mecanismo de análise gráfica. Quando encontrado o mesmo ativo produzido através do campo "produced assets" dos *docelements*, configura-se uma variável booleana que informa que esse componente faz parte da solução e continua-se iterando sobre os *docelements* em busca do endereço do *log* informando no campo URI.

De posse do endereço do arquivo de *log*, itera-se sobre cada linha desse arquivo em busca do período temporal definido. Para cada linha do arquivo pertencente a um mês *X*, pertencente ao período definido, é somada uma unidade ao valor total de vezes

que o ativo de software foi utilizado naquele mês *X*. Dessa forma, ao término da iteração sobre a lista de ativos de software cadastrados, tem-se todos os que utilizam o mesmo tipo de ativo produzido e, para cada mês do período definido, quantas vezes o ativo de software foi utilizado, sendo comparado o ativo produzido por cada ativo de software.

4. **ECOS, onde o nó central é a tecnologia utilizada e ao redor são dispostos os ativos de software (componentes na Brechó) que utilizam essa tecnologia**

Por se tratar da criação de um grafo, a geração de uma imagem se torna complicada, pois, para melhor visualização, é recomendado que os nós se adaptem às distâncias entre eles em tempo de execução, garantindo que a informação fique o mais visível possível. Como o *plug-in* JFreeChart que fora utilizado nos demais gráficos não oferecia suporte à criação de grafos, um outro *plug-in* chamado Prefuse foi utilizado. O *plug-in* mostra os grafos a partir de um arquivo .XML e, portanto, para que o grafo pudesse ser montado, configurou-se o arquivo .XML para que ficasse dentro da pasta *log* criada anteriormente para outros fins. Dando inicio à criação do grafo, o primeiro passo é identificar qual tecnologia do componente a partir do qual acionou-se o mecanismo de análise gráfica e armazenar tal informação no arquivo .XML como um vértice. Descoberta a informação, itera-se sobre a lista de todos os componentes cadastrados, verificando se a tecnologia utilizada no componente era a mesma inicialmente identificada. Para cada componente que possui a mesma tecnologia, acrescenta-se ao grafo um novo nó, ligando-o à tecnologia encontrada inicialmente por meio de um vértice. Ao término da leitura de toda lista de componente, tem-se a formação do grafo contendo, como informação central, a tecnologia em foco e, ao seu redor, conectado pelos vértices, os componentes que utilizam aquela tecnologia.

Capítulo 4. Exemplo de Uso

Para ilustrar as modificações realizadas na biblioteca Brechó visando adaptá-la à abordagem do grupo SECOGov, um exemplo de utilização dos recursos implementados é apresentado neste capítulo. Começa-se fazendo o *upload* do *template* do grupo SECOGov, cadastrado como uma categoria, e criando um ativo de software pertencente a essa categoria. Logo após, são cadastradas algumas licenças e, para cada ativo de software criado, associa-se um número total de licença que uma organização *X* possui e os seus respectivos usuários internos. Por fim, visualiza-se na forma gráfica o resultado de algumas informações disponíveis para um ativo de software do grupo SECOGov.

Ao acessar a biblioteca Brechó (Figura 12), a tela inicial disponibiliza um mecanismo de busca por componentes, além de apresentar alguns componentes em foco (i.e., componentes de destaque, que foram bem avaliados e/ou muito adquiridos, ou o contrário, na forma de promoção), assim como realizar um novo cadastro de usuário ou acessar a biblioteca como um usuário cadastrado (Santos *et al.*, 2010).

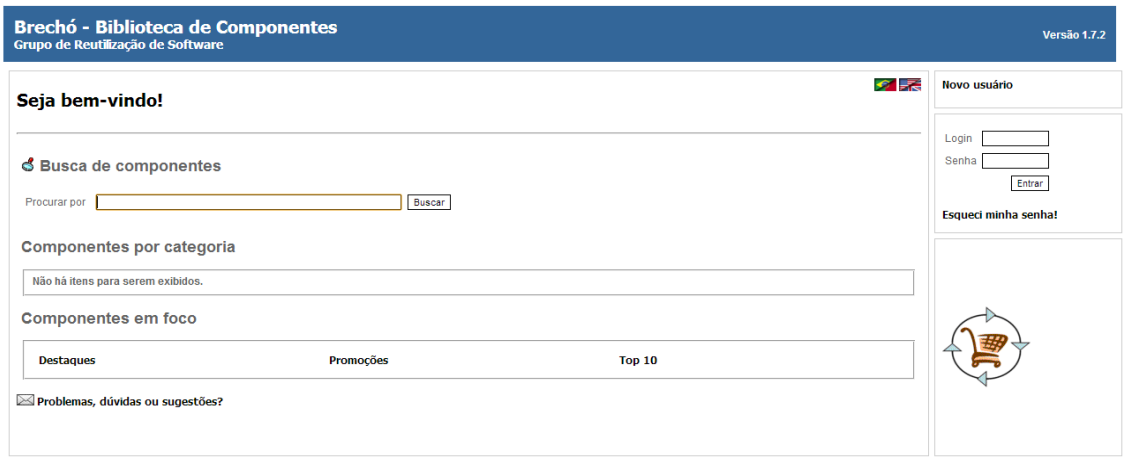

Figura 12 - Página inicial da biblioteca Brechó

Para acessar todos os recursos que a biblioteca oferece, utiliza-se o usuário "supervisor" na execução deste exemplo (Figura 13). Esse usuário é o administrador da biblioteca. Após realizar *login*, um menu lateral a direita da página inicial será disponibilizado. O primeira passo a ser feito consiste em cadastrar o *template* de formulário do grupo SECOGov e, para isso, deve-se acessar o menu "Instalar Formulário" (Figura 14). Em seguida, seleciona-se o arquivo .XML referente ao grupo SECOGov e confirma-se a instalação do *template*. No lado esquerdo da tela, uma

mensagem de instalação bem sucedida aparece na tela inicial, abaixo de “Seja bem-vindo!” (Figura 15) e acima de “Busca de componentes”.

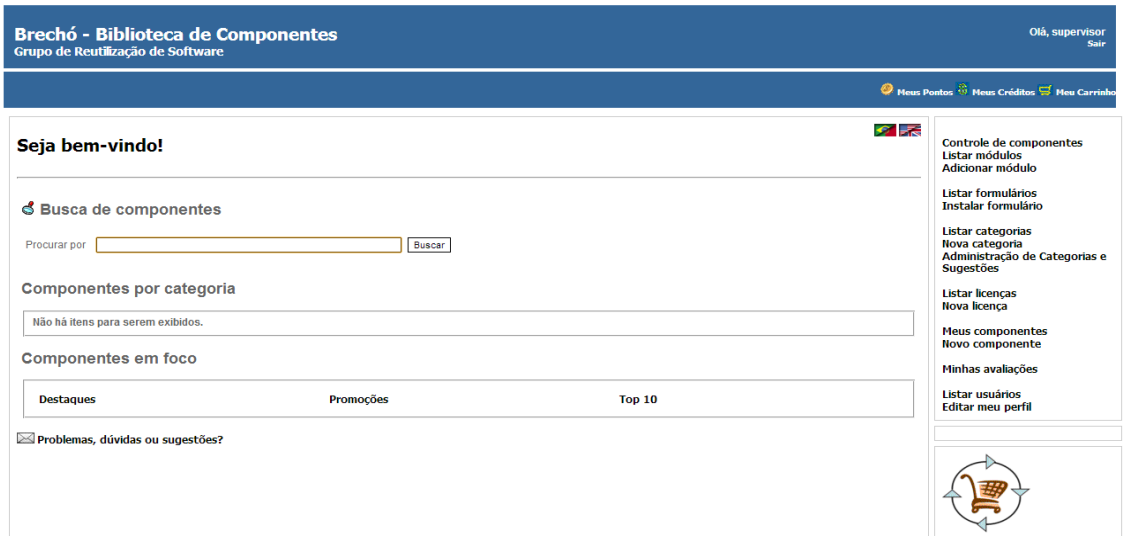

Figura 13 - Menu de acesso de usuário administrador

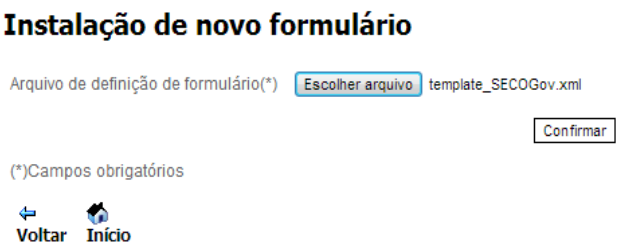

Figura 14 – Página de Instalação de um novo formulário

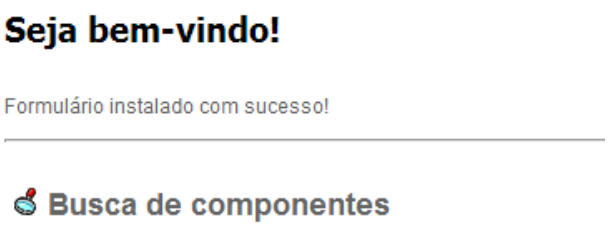

Figura 15 – Mensagem de instalação de formulário

Para ilustração, são instalados mais dois outros tipos de formulário, utilizados durante a apresentação do exemplo (Figura 16). Estes formulários não contemplam as informações necessárias para o grupo SECOGov, ou seja, são *templates* utilizados para componentes que usuários comuns (desenvolvedores que são produtores ou consumidores de componentes e serviços), que não são organizações consumidoras, utilizam para armazenar e negociar seus componentes, estando fora do escopo deste trabalho. A instalação destes formulários se torna necessária para demonstrar as diferenças existentes entre ambos.

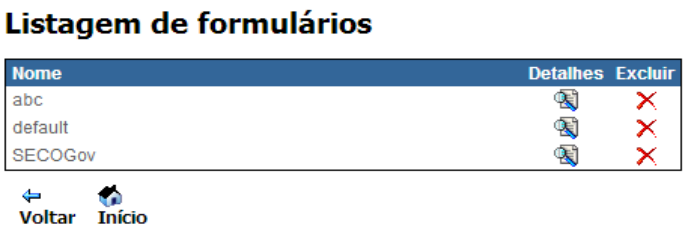

Figura 16 – Listagem de formulários cadastrados

Após o término da instalação dos formulários, pode-se realizar o cadastro da categoria "SECOGov", pelo menu lateral à direita, na opção "Nova categoria". Neste momento, deve-se dar um nome à nova categoria, informando, na sua descrição, a que tipo de componente ou ativo de software essa categoria estará associada, além de escolher o tipo de formulário que os componentes desta categoria devem responder (Figura 17). Ao clicar em "Prosseguir", prossegue-se para uma tela na qual as hierarquias (ou caminhos) onde a categoria deve ser classificada, de forma semelhante ao que acontece em árvores de diretórios nos sistemas operacionais (Figura 18). Ela pode ser do tipo Supercategoria, sendo uma categoria "raiz" dentro do sistema, ou uma Subcategoria, ou seja, estar dentro de uma ou mais Supercategorias. No exemplo, todas as categorias serão Supercategorias.

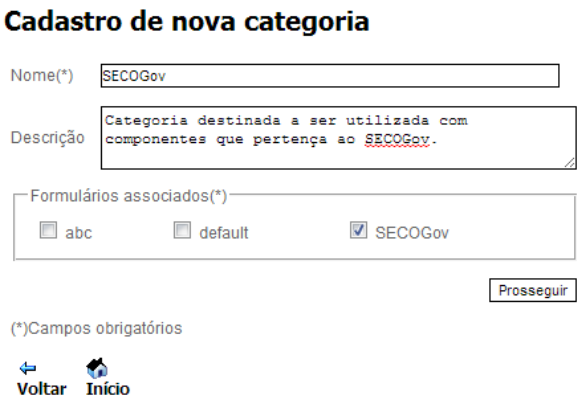

Figura 17 - Cadastro de nova categoria

Figura 18 - Hierarquia da nova categoria SECOGov

Assim como ocorreu para os formulários, duas novas categorias foram criadas, cada uma associada a um formulário instalado, para que se possa exemplificar as funcionalidades do grupo SECOGov. Após o cadastro das três categorias, na tela inicial, elas podem ser visualizadas na seção "Componentes por Categoria" (Figura 19). Parte-se agora para cadastrar componentes, alguns deles pertencentes à categoria SECOGov (ativos de software) e outros, às demais categorias (Figura 20). Um conjunto de ativos de software foi listado para ilustrar as funcionalidades da biblioteca, não tendo nenhum tipo de vínculo comercial com os fornecedores e produtos que serão mencionados.

Figura 19 - Listagem das categorias cadastradas

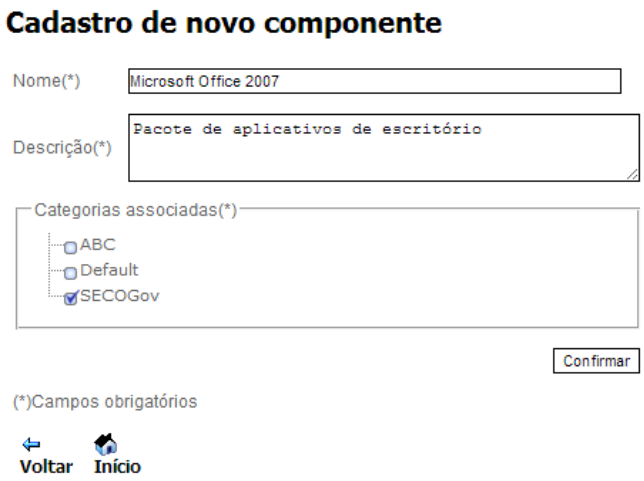

Figura 20 - Cadastro de novo componente

Ao clicar em "Confirmar", a biblioteca direciona o usuário para a tela de preenchimento dos formulários associados às categorias selecionadas (Figura 21). Após finalizar esta atividade, a biblioteca redireciona o usuário para a uma tela na qual é avaliado o nível de satisfação das categorias existentes [Raposo, 2007]. Foram cadastrados também os seguintes ativos de software (Figura 22):

- *Microsoft Office 2010*
- *Mozilla Thunderbird*
- *OpenOffice Writer 3*
- *Linux Ubuntu*
- *ABC 1*
- *Default 1*
- *ABC 2*

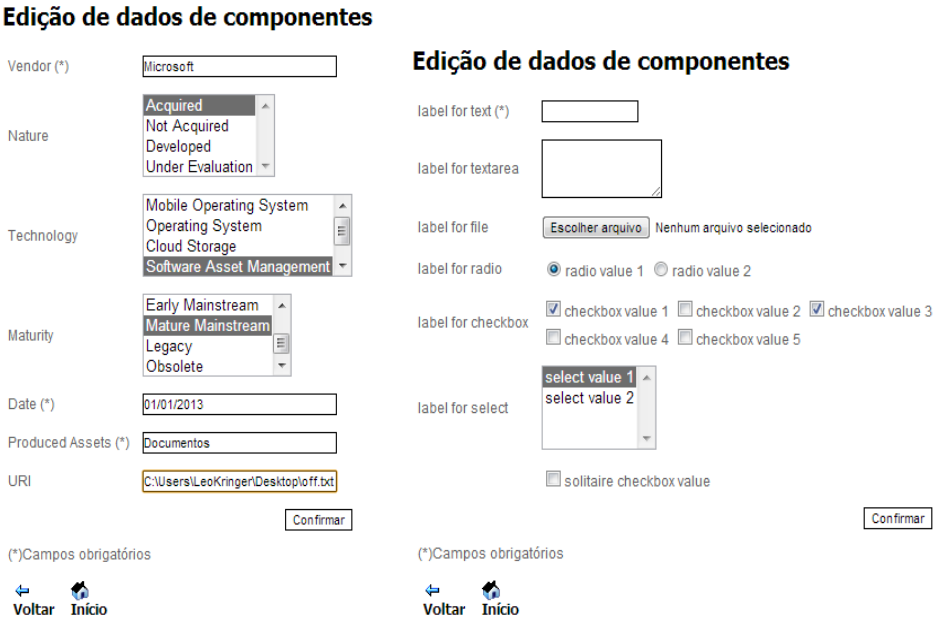

Figura 21 – Formulário SECOGov (à esquerda) e formulário ABC (à direita)

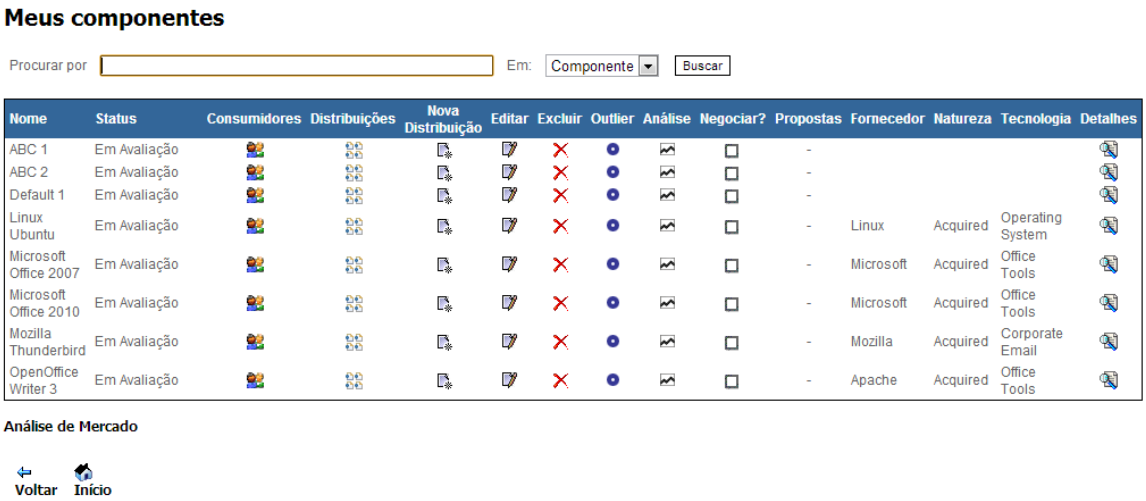

Figura 22 - Componentes cadastrados

Vale destacar que, para os componentes que não são do grupo SECOGov, os campos referentes a estes ficam sem preenchimento. Para os ativos de software do grupo SECOGov, cadastra-se as licenças associadas, assim como a quantidade de cada uma delas e respectivos usuários. Para isso, aciona-se o ícone da coluna "Distribuições" e, em seguida, o ícone "Nova Release" (Figura 23). Isso permite cadastrar uma nova versão (*release*) na qual estarão vinculadas as licenças. Para alguns ativos de software do grupo SECOGov, são cadastradas mais de uma versão e mais de uma licença. Obrigatoriamente, tem-se que associar um artefato (arquivo) do tipo *src* (*source*, ou código-fonte) e outro do tipo *bin* (*binary*, ou executável), além de definir um valor (crédito) para cada um deles, embora o preço não precise ser o mesmo. Esta funcionalidade faz parte de outro trabalho desenvolvido anteriormente na biblioteca

Brechó [Marinho, 2008]. Para cada ativo de software, a Figura 24 mostra o esquema de versões e licenças que foi cadastrado na biblioteca, incluindo seus respectivos usuários e quantidade total de licenças.

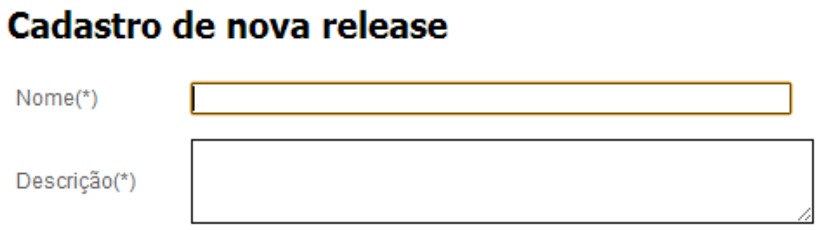

Figura 23 - Cadastro de uma nova versão

Componente	Versão	Total Licença	Usuários	Total Usuários
Office 2007	REL 1	10	u1;u2;u3;u4;u5;u6	6
Office 2010	REL 2	10	u1;u2;u3;u4	4
Thunderbird	REL 1	2	u1;u2;u3;u4	4
Writer 3	REL 1	4	u1	1
	REL 2	3	u1;u2;u3;u4	4
	REL 3	5	u1;u2;u3	3
Ubuntu	REL 1	n/a	n/a	n/a

Figura 24 – Esquema de versões e licenças por ativo de software

Uma vez cadastradas as versões e licenças que cada ativo de software possui, assim como os seus usuários, pode-se usufruir do recurso de gerência de ativos de software por meio de análises baseadas em gráficos. O primeiro gráfico disponível realiza uma análise das licenças cadastradas para cada ativo de software que possui um fornecedor em comum, gerando, como resultado, o percentual de licenças que está em uso. Para obter esta informação, no exemplo, deve-se acessar o mecanismo de análise gráfica a partir do ativo de software *Microsoft Office 2007*, cujo fornecedor é "Microsoft". Da listagem de ativos de software, repare que os ativos que possuem como fornecedor "Microsoft" são *Microsoft Office 2007* e *Microsoft Office 2010*.

Da tabela de versões e licenças, obtém-se a seguinte informação: o ativo de software *Microsoft Office 2007* possui dez licenças, sendo apenas seis delas utilizadas, o que resultaria em um percentual de 60% de licenças utilizadas e de 40% de não utilizadas. Para o ativo de software *Microsoft Office 2010*, existem também dez licenças, porém apenas quatro delas são utilizadas, gerando um percentual de 40% de licenças utilizadas e de 60% de não utilizadas. Ao gerar o gráfico de análise do número

de licenças utilizadas e não utilizadas do fornecedor “Microsoft”, espera-se receber, então, a informação de que 50% das licenças são utilizadas e 50% não são utilizadas, como mostra a Figura 25.

Figura 25 – Analise de licenças em uso do fornecedor Microsoft, entre os meses de janeiro e agosto de 2013

Caso seja necessário analisar o uso das licenças de um ativo de software especificamente, deve-se buscar tal informação no relatório da análise gráfica sobre as versões do ativo de software. Dessa forma, para o caso do ativo de software *Microsoft Office 2007*, por exemplo, para saber exatamente quantas licenças de cada versão estão em uso, deve-se acessar o mecanismo de análise gráfica a partir do componente em questão e filtrar a análise por versão, o que gera o resultado mostrado na Figura 26.

No exemplo, começa-se pelo caso em que, para um ativo de software e para uma versão deste, tem-se um número de usuário maior do que o número de licenças, ou seja, tem-se um número *X* de licenças menor do que o número de usuários que efetivamente a utiliza. Nesse caso, é desejavel que, na exibição do gráfico, o percentual de utilizaçao seja superior a 100%, indicando que aquele ativo de software possui um número de licenças inferior ao número de usuários. Consultando o esquema de versões e licenças de cada componente (Figura 24), nota-se que o ativo de software *Mozilla Thunderbird* se enquadra nesse caso. Ao acessar, então, o mecanismo de análise gráfica a partir deste ativo de software e a biblioteca gerar o relatório com o filtro “versão”, espera-se obter um gráfico com o percentual de uso superior a 100%, uma vez que há apenas duas licenças cadastradas para um número total de quatro usuários, como indica a Figura 27.

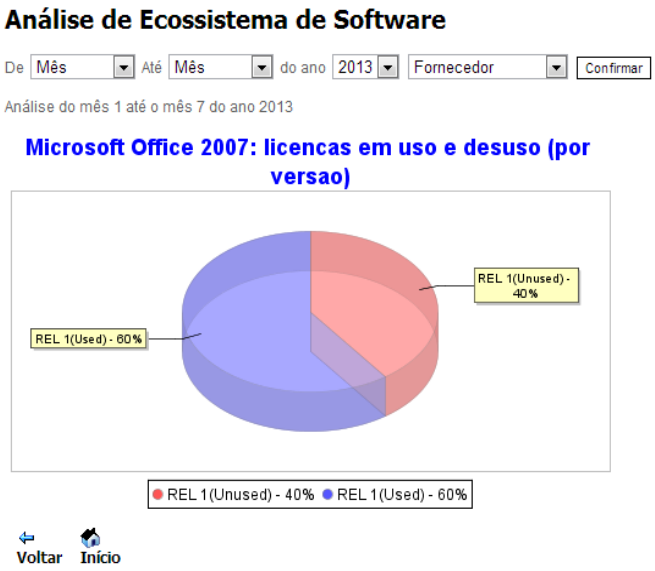

Figura 26 – Licenças em uso do ativo de software *Microsoft Office 2007*

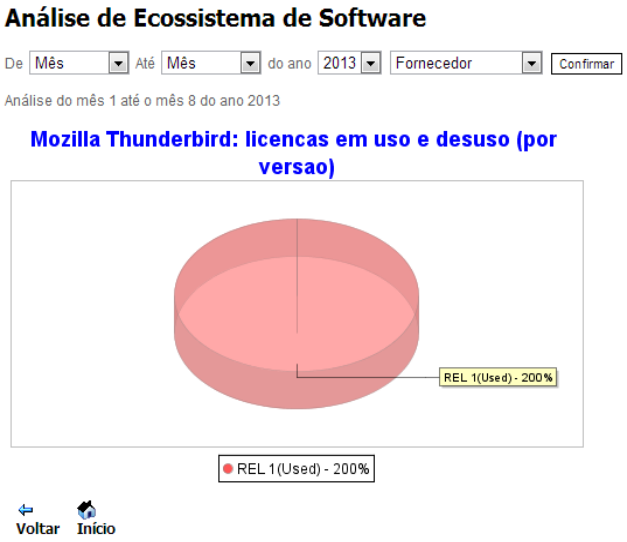

Figura 27 – Licenças em uso do ativo de software *Mozilla Thunderbird*

Note que o percentual de "licenças em desuso" não aparece, uma vez que não existe esse dado. Para o caso de um ativo de software com diversas versões disponibilizadas, cada uma contendo um dado número de licenças, nota-se que, ao analisar o esquema de versões e licenças (Figura 24), o Apache OpenOffice Writer 3 se enquadra nessa situação. Ao realizar uma breve análise, observa-se que, para algumas licenças, o uso supera o número disponível, ao passo que, em outros casos, o número de usuários é inferior. Considerando a análise de uso de cada versão disponível, espera-se que o gráfico gerado seja dividido pelo número de versões disponíveis para o ativo de software em questão, apresentando quanto de cada versão está ou não em uso. No exemplo, como o ativo de software dispõe de três versões, o gráfico deverá ser

particionado em três partes iguais, apresentando, em cada quadrante, a análise sobre a versão disponibilizada deste ativo, como exibe a Figura 28. Note as peculiaridades na análise deste gráfico para o ativo de software em questão:

1. O gráfico foi particionado exatamente pelo número de versões disponíveis (Figura 24), possibilitando uma análise geral sobre o ativo de software em relação às versões e licenças disponíveis;
2. Em cada partição, foi realizada a análise de uma versão, exibindo quanto de cada licença, por versão, está em uso (e quanto não está). A exceção é a versão na qual há um número maior de usuários do que licenças disponíveis, que apresenta apenas o percentual de licenças em uso superior a 100%.

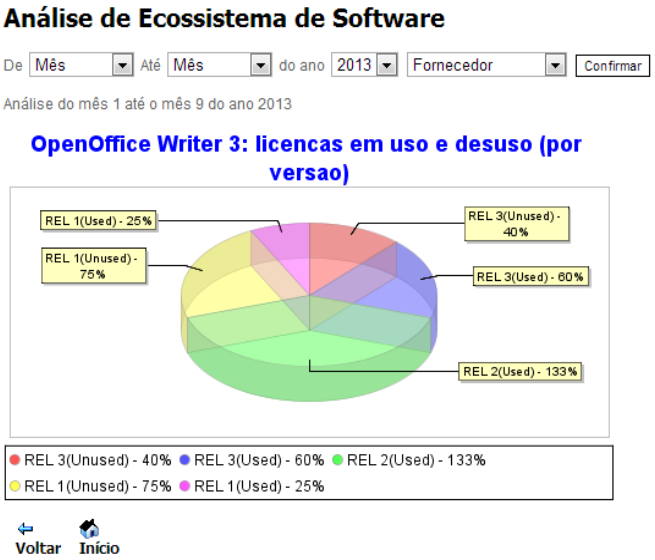

Figura 28 - Licenças em uso do ativo de software OpenOffice Writer 3

Para a geração do gráfico que mostra o relatório de utilização dos ativos de software em um período de tempo, os arquivos de *log* criados randomicamente no ato de cadastro do ativo de software foram utilizados. Além das informações temporais contida nos arquivos de *log*, para a formação do gráfico, foi necessário saber qual o ativo produzido pela utilização de um ativo de software. Como exemplo, são apresentados o relatório gerado para dois casos:

- Relatório de utilização de um ativo de software que gera um ativo produzido chamado "Documentos" (que é gerado quando os ativos de software *Microsoft Office 2007* e *Microsoft Office 2010* são utilizados) (Figura 29);
- Relatório de utilização do ativo de software que gera um ativo produzido chamado "e-mail" (Figura 30).

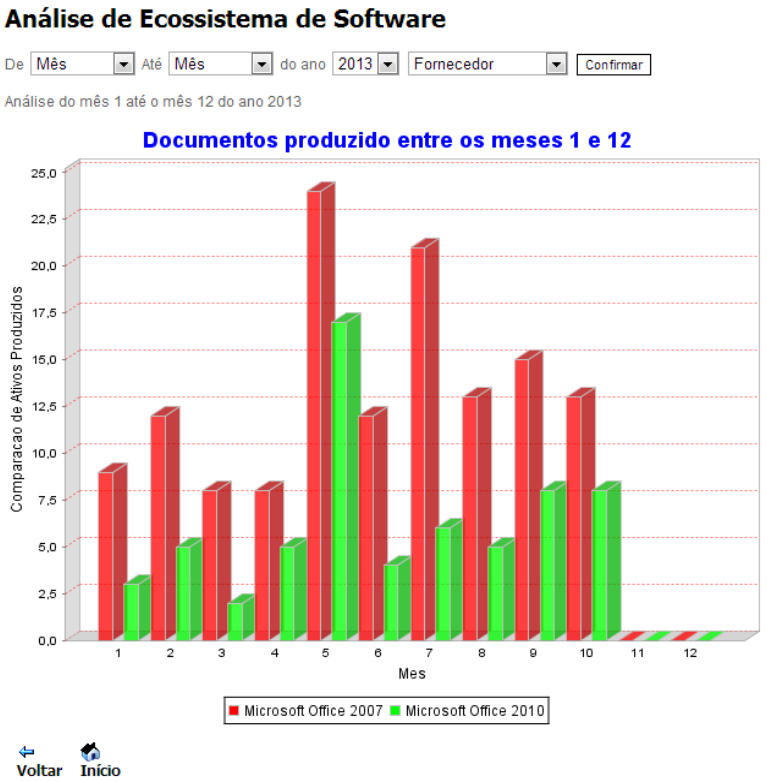

Figura 29 – Produtividade dos ativos de software do ativo produzido "Documentos"

Por fim, o último relatório a ser gerado é um grafo que mostra os ativos de software que utilizam uma mesma tecnologia, trabalhando a ideia de como um ECOS pode surgir e ser formado a partir de uma tecnologia de software central ou plataforma. Ao clicar em um dos três componentes que utilizam a tecnologia Software Asset Management, o grafo de ECOS é gerado, como mostra a Figura 31.

Além dos relatórios, outro recurso é disponibilizado nesta adaptação da Brechó para o grupo SECOGov: o sistema de busca que foi criado especialmente para contemplar as características de uma gerência de ativos de software. Na tela "Meus componentes", há o campo de busca no qual pode-se filtrar a consulta por *Componente*, *Fornecedor*, *Tecnologia* ou *Usuário* dos componentes (ativos de software). A seguir, um exemplo de uso de cada um dos filtros. No primeiro caso, filtra-se a busca por componente pelo texto "1", ou seja, ao digitar "1" no campo de texto e selecionar a opção "componente" no filtro, deseja-se que todos os componentes (ativos de software ou não), que tenham ao menos "1" no nome, sejam exibidos no resultado (Figura 32).

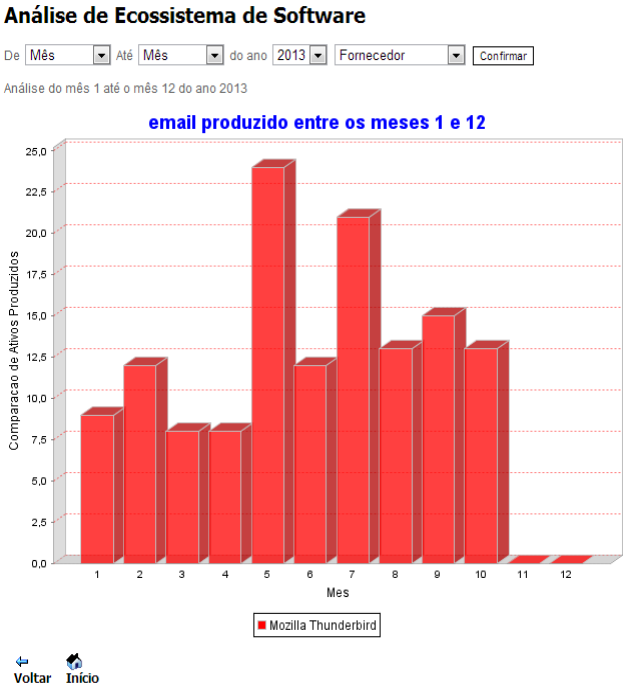

Figura 30 – Produtividade do ativo de software do ativo produzido "e-mail"

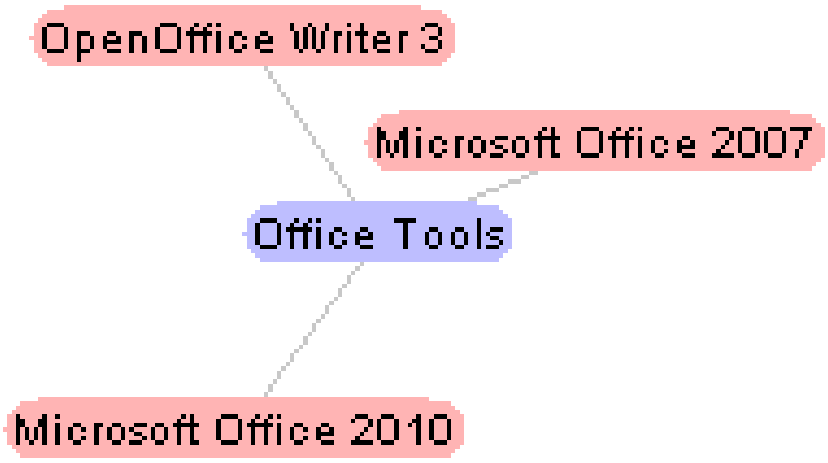

Figura 31 – Grafo de ECOS da tecnologia utilizada por um conjunto de ativos de software

No segundo caso, filtra-se a busca por fornecedor pelo texto "oft", ou seja, ao digitar "oft" no campo de texto e selecionar a opção "fornecedor" no filtro, deseja-se que todos os componentes (ativos de software ou não), que tenham fornecedor com nome que contenha a sequencia "oft", sejam exibidos no resultado (Figura 33). No terceiro caso, filtra-se a busca por tecnologia pelo texto "ss", ou seja, ao digitar "ss" no campo de texto e selecionar a opção "tecnologia" no filtro, deseja-se que todos os componentes (ativos de software ou não), que tenham pelo menos "ss" no nome, sejam exibidos no resultado (Figura 34). No último caso, filtra-se a busca por usuário pelo

texto “u6”, ou seja, ao digitar “u6” no campo de texto e selecionar a opção “usuário” no filtro, deseja-se que todos os componentes (ativos de software ou não), que tenham um usuário “u6”, sejam exibidos no resultado (Figura 35). Com os resultados exemplificados, contemplou-se todos os requisitos que foram inicialmente listados no Capítulo 3 para que a extensão da biblioteca Brechó passasse a atender às necessidades iniciais para que a gerência de ativos de software fosse realizada.

Figura 32 – Busca por componente com filtro com valor “1”

Figura 33 – Busca de fornecedor com filtro com valor “oft”

Figura 34 - Busca de tecnologia com filtro com valor “ss”

Figura 35 – Busca de usuário com filtro com valor “u6”

Capítulo 5. Conclusão

Com o avanço tecnológico da Computação, as funcionalidades presentes nos sistemas requeridos pelas organizações aumentaram, assim como a complexidade no seu desenvolvimento e manutenção. Segundo Werlang & Oliveira (2006), para resolução de problemas complexos e grandes, uma boa prática da Engenharia de Software consiste na utilização da técnica de dividir problemas grandes em menores a fim de reduzir a complexidade e facilitar o entendimento do problema. O processo de divisão do problema permite elaborar solução a partir do conceito de modularização, como acontece na ESBC na forma dos componente e serviços de software [Santos, 2013]. Muitos dos problemas possuem o mesmo domínio de aplicação envolvido e, por consequência, um conjunto de soluções similares [Werlang & Oliveira, 2006].

Dessa forma, para a reutilização de componentes e serviços, é necessário armazená-los previamente de alguma forma. Além disso, similarmente ao que aconteceu na indústria de hardware, a indústria de software também tentou gerar um mercado em torno de componentes e serviços de software, sem grande êxito [Santos *et al.*, 2010], embora isso venha mudando com o conceito de aplicações [Bosch, 2009]. Estas iniciativas levaram ao surgimento dos ECOS, redes de organizações que se relacionam a partir de ferramentas, sistemas e serviços disponibilizados ao redor de uma tecnologia de software central [Hanssen & Dyba, 2012].

O surgimento dessa rede gerou benefícios e problemas, tanto para as organizações fornecedoras como para as consumidoras de tecnologia de software. Para os fornecedores, a carteira de clientes aumentou, porém, o rigor sobre o controle de licenças vendidas teve que aumentar, dado que geralmente a maior parte dos lucros da venda de produtos de software vem justamente da sua permissão de uso. Por outro lado, para as organizações consumidoras, o controle das licenças existentes para cada ativo de software é crucial do ponto de vista financeiro, visto que um dos maiores custos para o setor de TI das organizações está justamente nas licenças dos ativos de software, no caso, componentes, serviços e aplicações que agregam valor à organização.

Frente a esse problema, o objetivo deste trabalho de final de curso foi adaptar uma biblioteca de componentes e serviços para desempenhar a gerência de ativos de software de uma organização consumidora na forma de uma ferramenta de governança, analisando a frequência de uso de cada ativo de software e o controle de licenças

vinculadas a cada versão existente. Apesar do sucesso na expansão das funcionalidades da biblioteca, melhorias ainda podem ser realizadas como, por exemplo, criar um cadastro prévio de colaboradores da organização, para que estes possam mais facilmente associados às licenças. O mesmo raciocínio serviria para os fornecedores, ou seja, ao deixar o preenchimento do campo Fornecedor como texto livre, erros podem ser inseridos no ato da digitação, o que ocasionaria em uma inconsistência na geração de relatórios e outros.

Como limitações, destaca-se a falta de um papel específico que represente a organização na biblioteca, na qual um conjunto de usuários estaria vinculado e teria acesso a todos os ativos de software cadastrados para essa organização. Como não há esse papel, outra restrição está no fato de quem deve cadastrar os ativos de software da organização. Como os ativos estão vinculados a um usuário, seria interessante que quem cadastrasse os ativos de software fosse o responsável por gerenciar e tomar decisões sobre estes, e não um usuário interno à organização. Outra limitação está relacionada ao cadastro do fornecedor não estar separada do cadastro do ativo de software. Como o campo fornecedor deve ser preenchido na forma de texto livre, no ato do cadastro de um ativo, um mesmo fornecedor pode ser redigido de forma diferente, ocasionando impactos no mecanismo de busca e análise gráfica da ferramenta. Exemplo: Maicrosoft (como se fala em português) e Microsoft (como se escreve em inglês).

Como trabalhos futuros, pretende-se realizar dois estudos experimentais: (i) utilizar a ferramenta no dia a dia de uma organização que não possui nenhum tipo de suporte tecnológico à gerência de ativos de software para avaliar os impactos que a abordagem pode ocasionar num primeiro momento nas organizações; (ii) utilizar a ferramenta no dia a dia de uma organização que já utilizou anteriormente algum tipo de sistema de gerência de ativos de software, adquirindo experiência sobre os pontos fortes e fracos que a ferramenta possui hoje, visando melhorá-la; e (iii) evoluir a adaptação da biblioteca Brechó para contemplar a visão externa de ECOS, i.e., incluir relatórios de institutos de pesquisa e de recomendação de tecnologia de ECOS para a organização consumidora, a fim enriquecer a visão interna, baseada na gestão dos ativos de software.

Referências Bibliográficas

Albert, B.; Santos, R.; Werner, C. (2013) "Software Ecosystems Governance to Enable IT Architecture Based on Software Asset Management". In: Proceedings of the 7th IEEE International Conference on Digital Ecosystems and Technologies – Complex Environment Engineering, Menlo Park, CA, USA, pp. 55-60.

Almeida, E. S.; Bianchini, C. P.; Prado, A. F.; Trevelin, L. C. (2003) "Um Padrão para o Desenvolvimento de Software Baseado em Componentes Distribuídos". Proceedings of the Second Latin American Conference on Pattern Languages of Programming, Itaipava, RJ, Brazil, pp. 175-190.

Antunes, L.; Santos, R.; Werner, C. (2013) "Impactos de Ecossistemas de Software no Processo de Aquisição". Anais do IX Workshop Anual do MPS, Campinas, SP, Brasil, pp. 96-106.

Bosch, J. (2009) "On Software Product Lines, Customer-Configured Products and Ecosystems". In: Anais do VIII Simpósio Brasileiro de Qualidade de Software, Ouro Preto, MG, Brasil, pp. 6-7.

BRAGA, R., 2000, Busca e Recuperação de Componentes em Ambientes de Reutilização de Software. Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

Brechó (2013) "Biblioteca Brechó". Disponível em http://reuse.cos.ufrj.br/brecho, acessado em 25/11/13.

Brown, A. W.; Wallnau, K. C. (1996) "Component-Based Software Engineering: Selected Papers from the Software Engineering Institute". Wiley - IEEE Computer Society Press, pp. 150.

Campbell, P.; Ahmed, F. (2010) "A Three-dimensional View of Software Ecosystems". In: Proceedings of the 4th European Conference on Software Architecture, 2nd International Workshop on Software Ecosystems, Copenhagen, Denmark, pp. 81-84.

Castor, E. M.; (2006) "Fábrica de Software: Passado, Presente e Futuro". Monografia de Pós-Graduação em Tecnologia da Informação, UNIBRATEC, Recife, PE, Brasil.

Coutinho, A.; Rangel, A. (2010) "Gerenciamento de Ativos de Software". KPGM no Brasil.

Feijó, R. H. B. (2007) "Uma Arquitetura de Software Baseada em Componentes para Visualização de Informações Industriais". Dissertação de Mestrado. Universidade Federal do Rio Grande do Norte, Natal, RN, Brasil.

Filho, R. C. S.; Katsurayama, A. E.; Santos, G.; Murta, L.; Rocha, A. R. (2008) “A Experiência na Implantação do Processo de Gerência de Reutilização no Laboratório de Engenharia de Software da COPPE/UFRJ”. ProQuality (UFLA), v. 4, p. 21-26.

Hanssen, G.; Dyba, T. (2012) “Theoretical Foundations of Software Ecosystems”. In: Proceedings of the 4th International Workshop on Software Ecosystems, 3rd International Conference on Software Business, Boston, MA, USA, pp. 6-17.

Jansen, S.; Boucharas, V.; Brinkkemper, S. (2009) “Formalizing Software Ecosystems Modeling”. In: Proceedings of the 1st International Workshop on Open Component Ecosystems, ACM/SIGSOFT Symposium on the Foundations of Software Engineering, Amsterdam, The Netherlands, pp. 41-50.

Kon, F.; Lago, N.; Sabino, V. (2011) “Licenças de Software Livre, descrição sistematização, compatibilidade e incompatibilidades”. Centro de Competência em Software Livre, IME-USP - SERPRO, São Paulo, SP, Brasil.

Manikas, K.; Hansen, K. (2013) “Software Ecosystems – A Systematic Literature Review”. Journal of Systems and Software, v. 86, n. 5 (May), pp. 1294-1306.

Marinho, A. S. (2008) “Uma Abordagem para Viabilização de Serviços e Tarifação de Componentes de Software”. Monografia de Graduação. Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil.

Marinho, A.; Werner, C.; Murta, L. (2009) “Infraestrutura de Serviços em uma Biblioteca de Componentes”. Anais do III Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software, Natal, RN, Brasil, pp. 18-25.

McGregor, J. (2012) “Ecosystem Modeling and Analysis”. In: Proceedings of 16th International Software Product Line Conference, Salvador, BA, Brasil, p. 268.

Microsoft (2013) “SAM – Software Asset Management”. Disponível em http://www.microsoft.com/pt-br/sam/, acessado em 27/11/13.

Marinho, A.; Santos, R.P.; Murta, L.; Werner, C. (2009) “Infraestrutura de Tarifação de Componentes e Serviços na Biblioteca Brechó”. Anais do III Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software, Natal, RN, Brasil, pp. 2-9.

Oliveira, F. C.; Paula, L. L. (2009) “Engenharia de Software Baseada em Componentes: Uma Abordagem Prática em ambientes Web”. Monografia de Graduação. Universidade de Brasília, Brasília, DF, Brasil.

Popp, K. (2012) “Leveraging Open Source Licenses and Open Source Communities in Hybrid Commercial Open Source Business Models”. In: Proceedings of the 4th

International Workshop on Software Ecosystems, 3rd International Conference on Software Business, Boston, MA, USA, pp. 33-40.

Rangel, A. (2013) "A Gestão de Ativos de Software (SAM – Sotware Asset Management) veio para ficar?". Disponível em http://www.tiespecialistas.com.br/2013/03/a-gestao-de-ativos-de-software-sam-software-asset-management-veio-para-ficar/#.UfUa142siSo, acessado em 27/11/13.

Raposo, R. (2007) "Brechó-ABC: Abordagem Integrada para Avaliação, Busca e Categorização de Componentes de Software". Monografia de Graduação. Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil.

Sabino, V.; Kon, F. (2009) "Licenças de Software Livre: História e Características". Relatório Técnico RT-MAC-IME-USP 2009-01. Centro de Competência em Software Livre, DCC/IME, Universidade de São Paulo, São Paulo, SP, Brasil.

Santos, R.P. (2013) "Engenharia e Gerenciamento de Ecossistemas de Software". Exame de Qualificação de Doutorado. COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

Santos, R. P.; Werner, C. M. L. (2011) "A Proposal for Software Ecosystems Engineering". In: Proceedings of the 3rd International Workshop on Software Ecosystems, Brussels, Belgium. CEUR Workshop Proceedings. Frankfurt: CEUR-WS, 2011. v. 746. pp. 40-51.

Santos, R.; Werner, C. (2012) "ReuseECOS: An Approach to Support Global Software Development through Software Ecosystems". In: Proceedings of the 7th International Conference on Global Software Engineering, 6th Workshop on Distributed Software Development, Porto Alegre, RS, Brasil, pp.60-65.

Santos, R. P.; Werner, C. M. L.; Silva, M. A. (2010) "Brechó-VCM: A Value-Based Approach for Component Markets". International Transactions on Systems Science and Applications (Print), v. 6, n. 2/3 (August), pp. 179-199.

SOFTEX (2011a) "Guia de Aquisição". Outubro, 2011.

SOFTEX (2011b) "Guia de Implementação – Parte 2: Fundamentação para Implementação do Nível F do MR-MPS". Novembro, 2011.

SOFTEX (2011c) "Guia de Implementação – Parte 3: Fundamentação para Implementação do Nível C do MR-MPS". Julho, 2011.

Spagnoli, L. A.; Becker, K. (2003) "Um estudo sobre o desenvolvimento baseado em componente". Relatório Técnico n. 26 (Maio), PUCRS, Porto Alegre, RS, Brasil.

Werner, C.; Murta, L.; Marinho, A.; Santos, R.; Silva, M. (2009) “Towards a Component and Service Marketplace with Brechó Library”, In: Proceedings of the IADIS International Conference WWW/Internet 2009, Rome, Italy, pp. 567-574.

Werner, C., Murta, L., Lopes, M., Dantas, A., Lopes, L. G., Fernandes, P., Prudêncio, J. G., Marinho, A., Raposo, R. (2007) “Brechó: Catálogo de Componentes e Serviços de Software”, In: Anais da XIV Sessão de Ferramentas do XXI Simpósio Brasileiro de Engenharia de Software, João Pessoa, PB, Brasil, pp. 24-30.

Williams, C.; O’Connor, S. (2011) “Four best Practices for Software Asset Management”. BCM Software Industry Insights.

Yu, E; Deng, S (2011) “Understanding Software Ecosystems: A Strategic Modeling Approach”. In: Proceedings of the Third International Workshop on Software Ecosystems, 2nd International Conference on Software Business, Brussels, Belgium, pp.65-76.