Engenharia de Software **2011/2012**

Solução desenvolvida no projeto final

Trabalho elaborado pelo grupo A0403

Grupo Segurança:	Grupo Replicação:	Grupo Prototipagem:
João Loff	**Tiago Aguiar**	**Hugo Sequeira**
Nº 56960	Nº 64870	Nº 63925
Alexandre Almeida	**Viteche Ashvin**	**Edgar Santos**
Nº 64712	Nº 64878	Nº 64753

Introdução

Nesta fase foi necessário desenhar uma solução que estende-se o funcionamento local para um funcionamento distribuído com requisitos não funcionais que tornassem o projeto mais fiável e seguro.

Gestão de Projeto

O trabalho avaliado foi divido em três grupos de duas pessoas: um grupo ficou encarregado da segurança, outro encarregue de replicação de tolerância a falhas e outro encarregue da prototipagem para estes assuntos. Usando para isso uma filosofia de desenvolvimento baseado em metodologias ágeis: *XP* no desenvolvimento do projeto e sobre *SCRUM* na gestão projeto.

Na gestão de projecto, e segundo o *SCRUM*, ao longo do projeto tivemos reuniões semanais à segunda-feira, assinalando o inicio de cada sprint. Nesta reunião semanal definimos uma estimativa das tarefas a realizar ao longo desse sprint, através do *SCRUM POKER*, as tarefas a realizar naquele sprint, os grupos responsáveis pela mesma e também uma estimativa de horas de trabalho para o próximo sprint. Para além desta reuniões semanais, e seguindo a metodologia, realizou-se diariamente uma SCRUM meeting, onde cada equipa relatava as suas atividades diárias e o progresso, assim como a atualização das horas despendidas até ao momento naquela tarefa e no dia que passou. De referir que

Em relação ao desenvolvimento de software, foi usado *XP* e *Pair Programming*, onde todas as tarefas definidas no *SCRUM* tinham como objectivo serem executadas por grupos

definidos previamente. Para além disso, e tirando um período onde foram desenvolvidos protótipos específicos sobre os assuntos de distribuição da aplicação, foi usado uma metodologia orientada a testes, *Test Driven Developing*, explicitada mais adiante.

Desenho de Software

Para tal a *bridge* distribuída foi redesenhada para utilizar um novo protocolo nos seus comandos, desenhado na cadeira de Sistemas Distribuídos, que respeita-se os requisitos levantados.

Inicialmente para cada comando, implementamos a funcionalidade desejada de acordo como o tipo de comando. Nesta abordagem obtivemos muito repetição de código na *bridge* distribuída. Após uma vasta análise e refactorização decidimos resolver este problema através de um padrão de desenho explicado em seguida.

Implementando o padrão de desenho Strategy, fundamentalmente definimos o que é um comando e dividimos os comandos disponíveis em dois grandes sub-grupos: comandos de escrita e comandos de leitura. Com esta generalização obtivemos três grandes classes: *CommandStrategy.java*, *GetCommandStrategy.java*, *WriteCommandStrategy.java*.

Através de uma refactorização *bottom-up*, o código comum a todos os comandos ficou implementado em *CommandStrategy.java*, o código comum a cada tipo de comando ficou em *GetCommandStrategy.java* e *WriteCommandStrategy.java*, o código específico a cada comando ficou na sua estratégia. Com esta solução conseguimos abstrair mas também especificar código sem repetição de código extrema.

Outra refactorização ocorreu nos *handlers* às respostas aos pedidos *WebServices* em que o código comum de cada resposta, mais especificamente as exceções e a comando que invocou a resposta, ficou na classe superior. Com esta implementação, conseguimos poupar código, e através de uma interface assíncrona, que não devolvia nenhuma exceção, conseguimos iterar as respostas a questionar se alguma resposta gerou exceção, estendendo a funcionalidade existente.

Desenhamos também uma classe nova *VoteList.java*, que faz a gestão da votação de qual a resposta correta a devolver, através de votos, utilizando a classe *Vote.java*. Deste modo e através de métodos *Template* criamos uma barreira de abstração, que pode ser alterada no futuro caso seja preciso modificar o termo de votação ou escalar o sistema.

Para além dos comandos distribuídos, foi necessário desenhar uma solução que garantisse integridade e autenticidade

Nesta solução foi necessário implementar um servidor de Certificados, ao qual chamamos *Certificate Authority* (acrónimo CA) que iria servir a nossa aplicação de certificados válidos para as comunicações entre *Presentation* e *Application Server*. Esta entidade, foi desenvolvida segundo o mesmo desenho da arquitetura do software que foi introduzido e explicitado na aplicação Anacom.

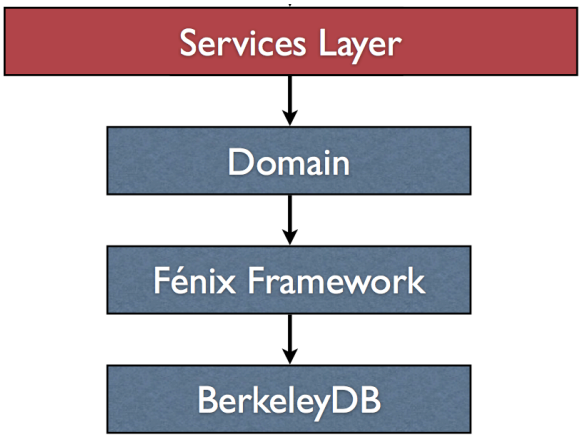

Figura 1. Desenho da Arquitectura da CA

Resumidamente, esta entidade possui uma base de dados com objetos relacionais em que usamos a *Berkeley DB*. Esta base de dados é gerida através da Fénix Framework, e ao conjunto das duas entidades chamamos de domínio. Em cima desta camada de domínio, existe a camada de serviços da CA, composta por: serviço para obter a chave pública da mesma; serviço para assinar um certificado; serviço para revogar um certificado; e serviço para obter os certificados revogados presentes na CA.

Para o caso da CA não é necessário existir uma camada de apresentação, logo, em comparação com o desenho arquitetural da Anacom, esta camada não existe, sendo que os clientes da mesma vão comunicar com ela diretamente através da camada de serviços.

Software Testing

Para o desenvolvimento de testes para a aplicação foram utilizadas duas abordagens distintas. Numa primeira, focámo-nos nos testes da aplicação localmente testando o funcionamento correto dos serviços, das suas propriedades, e dos seus erros esperados. Nesta parte usamos a *framework* de testes *JUnit*.

Para o desenvolvimento de testes para a parte distribuída, focámo-nos apenas nos testes de aspectos específicos da comunicação distribuída, como é o caso do algoritmo de quóruns e replicação, usando para isso a *framework EasyMock*[1], que possui um funcionamento semelhante á ferramenta *jMock* explicada nas aulas[2].

Um segundo grupo de testes, envolveu o desenvolvimento de *handlers* de testes, usados para simular a corrupção na transferência de certificados, atraso na comunicação dos mesmos, e restantes testes para a autenticidade e integridade da comunicação.

[1] (http://www.easymock.org/EasyMock3_1_Documentation.html) 15/05/12
[2] (http://www.jmock.org/easymock-comparison.html) 15/05/12

Diagrama de Classes