Mestrado em Engenharia Informática
Dissertação/Estágio
Relatório de Estágio

csSECURE – Software Update Service

Tiago Luís Gomes Carregã
tlgc@student.dei.uc.pt

Orientadores:
Doutor Edmundo Monteiro, DEI, FCTUC
Eng.º Bernardo Patrão, Critical Software S.A.
Data: 12 de Julho de 2012

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

iTGROW Critical software

AGRADECIMENTOS

Terminada esta importante jornada da minha vida académica, não podia deixar de dirigir algumas palavras de agradecimento a todos os que, das mais diversas formas, contribuíram para o meu crescimento, tanto ao nível profissional como humano.

Ao Doutor Edmundo Monteiro, pelo acompanhamento e em especial por toda a experiência, saber e conhecimento transmitidos durante o período de estágio.

Ao Eng.º Bernardo Patrão, pela paciência inesgotável. Pela disponibilidade e acompanhamento constantes. Por toda a experiência, saber e conhecimento transmitidos. Pela preocupação. Pela amizade.

A toda a equipa do csSECURE, que se revelou fantástica, repleta de companheiros inigualáveis, sempre disponíveis para ajudar no que fosse necessário, quando fosse necessário.

À iTGROW e Critical Software pela oportunidade e condições concedidas. Tive o privilégio de trabalhar num ambiente fantástico, rodeado de pessoas e profissionais exemplares.

À Catarina Fonseca, pelo apoio, acompanhamento e atenção dispensada, pela energia positiva e pelo contributo para a minha integração na empresa.

Aos meus pais e irmão, pelo seu inesgotável apoio ao longo de todo o meu percurso académico. Pelo orgulho perante as conquistas. Pela força perante os obstáculos. Tornaram o sonho possível. Obrigado.

Por fim, um agradecimento muito especial à Laura, companheira incansável nos altos e baixos, nos bons e maus momentos. Por toda a força, ternura, carinho e confiança incondicionais, o meu sincero obrigado.

RESUMO

O principal objetivo deste projeto de estágio passa por desenvolver uma ferramenta de atualizações de *software*, que permita atualizar o csSECURE de forma totalmente automática, independente da iniciativa do utilizador.

Os principais requisitos desta ferramenta passam pelo seu funcionamento automático, seguro e transparente, reduzindo o nível de intrusão na utilização da máquina. Este conjunto de requisitos não se verifica em nenhuma das soluções presentes no mercado, abrindo assim espaço à investigação e desenvolvimento de um produto inovador.

De forma a atingir o objetivo acima enunciado, foi elaborado um estudo aprofundado do estado da arte na área das atualizações automáticas de *software*, seguido de uma fase de planeamento, especificação e desenvolvimento, concretizada no lançamento de uma primeira versão de avaliação do *csSECURE – Software Update Service*.

Como resultado e para além do *software* produzido, o presente relatório descreve os resultados de maior interesse das diversas fases do estágio, os principais desafios encontrados e as soluções propostas para os superar.

ÍNDICE

1 Introdução

1.1 Âmbito do estágio

O presente estágio foi realizado no âmbito da cadeira de Dissertação/Estágio do Mestrado em Engenharia Informática do Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra. O projeto de estágio teve início no dia 12 de Setembro de 2011 e terminou a 12 de Julho de 2012.

Todo o trabalho foi desenvolvido nas instalações da empresa Critical Software S.A., em Coimbra, no âmbito do projeto *csSECURE - Software Update Service* e inserido na área da segurança e atualizações automáticas de *software*.

A orientação do estágio esteve a cargo do Doutor Edmundo Monteiro, por parte da Universidade de Coimbra e do Eng.º Bernardo Patrão, da Critical Software S.A.

1.2 O produto csSECURE

A Critical Software investiu no desenvolvimento de um produto na área da segurança da informação – o csSECURE (*http://cssecure.net*). Este sistema é direcionado à proteção de informação não estruturada (e-mail, documentos do Microsoft Office, entre outros) em ambientes nos quais a garantia da confidencialidade da informação é um fator relevante. O csSECURE posiciona-se como uma solução de *Enterprise Rights Management* (ERM) e tem como base o conceito de segurança multinível. No seio de uma organização, a informação não estruturada é classificada segundo a política de segurança definida, ou seja, é-lhe atribuído um nível de confidencialidade. Paralelamente, os utilizadores na organização são credenciados para um ou mais níveis de confidencialidade, o que lhes confere direitos sobre os documentos classificados. A informação é protegida recorrendo a algoritmos de encriptação acompanhados de assinaturas digitais e o csSECURE garante a correta aplicação dos direitos, sempre que um utilizador solicita o acesso a determinado documento.

Na Figura 1 está ilustrada a situação de classificação de um *e-mail*, no Microsoft Outlook.

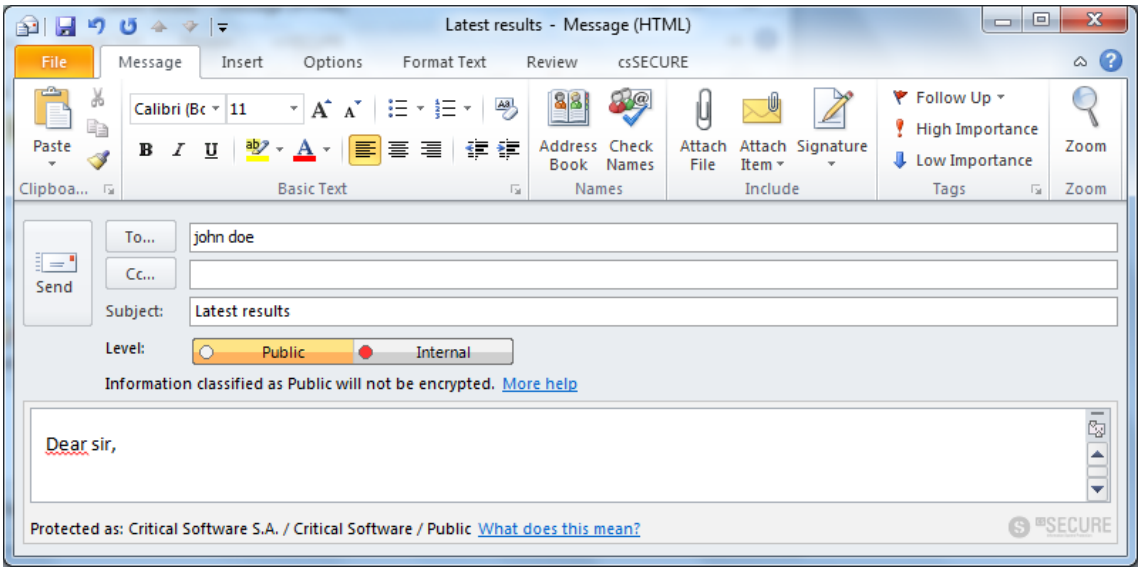


Figura 1 - Interface de classificação de confidencialidade para Microsoft Outlook

No csSECURE, a gestão dos direitos é feita de forma detalhada. Direitos como abertura ou edição, impressão, copiar, guardar, responder e encaminhar são geridos de forma independente. Esta gestão detalhada permite que a informação seja partilhada dentro da organização permitindo ao mesmo tempo evitar ações indesejadas que resultem em fuga ou perda de informação.

A Figura 2 ilustra uma vista em camadas da arquitetura de alto nível do csSECURE que, como se pode verificar, é fortemente baseada em tecnologias Microsoft.

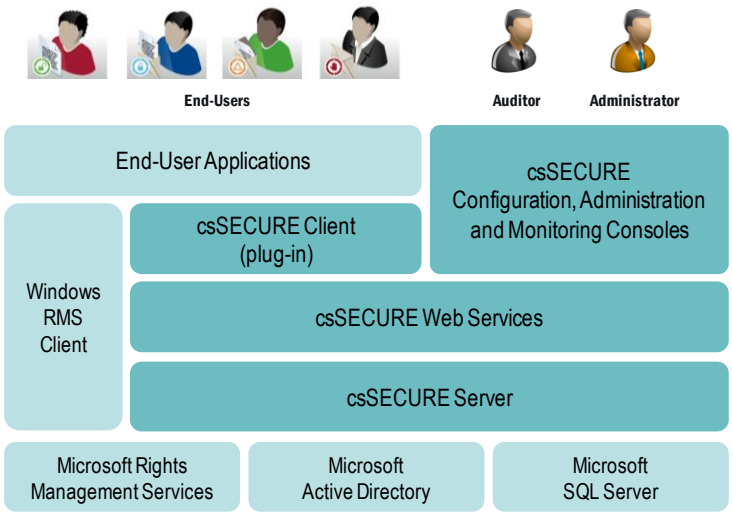


Figura 2 - Arquitetura de alto nível do csSECURE

O csSECURE implementa todos estes conceitos de *Enterprise Rights Management* recorrendo a tecnologias de criptografia, gestão e integração com ferramentas de autenticação e credenciação resultando numa integração estreita com as aplicações do utilizador final.

Sendo o csSECURE orientado a ambientes empresariais, o desenvolvimento foca-se nas aplicações mais comuns do utilizador final passando por *e-mail*, *webmail*, editores de texto, folhas de cálculo, dispositivos móveis, portais colaborativos, entre outros.

1.3 Problema

A atualização de *software* surge, nos dias de hoje, como um mecanismo primário de segurança que não pode, de forma alguma, ser descurado, sob pena de comprometer a segurança e funcionamento dos sistemas de informação.

Confiar exclusivamente em mecanismos de segurança externos como *firewalls* ou semelhantes já não é suficiente para garantir um elevado nível de segurança. Apesar da maturidade evidenciada, estes sistemas são falíveis e não garantem a completa blindagem de um sistema de informação que está à mercê de vários tipos de ataques, sejam de natureza técnica ou social. Como tal, é imperativo reduzir as falhas de segurança de *software* e assegurar que mesmo que os mecanismos de segurança externos falhem, o *software* interno de uma organização se encontra estável, atualizado e protegido.

Como muitas das falhas de segurança de determinados produtos são descobertas com o decorrer do tempo, estas necessitam então de ser corrigidas através de atualizações.

Não só numa perspetiva de segurança, mas também numa perspetiva funcional, manter *software* atualizado permite disponibilizar aos utilizadores de um produto as últimas funcionalidades lançadas ou até correções de falhas encontradas relacionadas com o funcionamento do mesmo.

No entanto, existem razões de naturezas diversas que impedem o *software* de ser atualizado com a necessária frequência. Por um lado, existe ainda alguma resistência por parte dos utilizadores que, por desleixo ou esquecimento, falham na simples tarefa de manter o seu *software* atualizado, por outro, os mecanismos de atualização utilizados são pobres e não asseguram que o *software* seja atualizado regularmente. Estes são talvez os dois maiores impedimentos no que toca às atualizações de *software*.

No caso concreto do csSECURE, o mecanismo de atualizações que vigora é baseado em GPO (*Group Policy Objects*), um recurso da Microsoft que permite, entre um largo

conjunto de outras funcionalidades, distribuir *software* dentro de um determinado domínio. Este mecanismo apresenta algumas falhas e não é de todo confiável no que toca à tarefa de manter o *software* atualizado.

A ideia central do mecanismo de distribuição de *software* por GPO é a de que todas as atualizações, uma vez publicadas, serão instaladas no próximo iniciar de sistema. Acontece que, no caso particular do csSECURE (distribuído para praticamente todas as máquinas pertencentes ao domínio interno da Critical Software), o comportamento real não corresponde ao esperado. Muitas das vezes, são precisos até três reinícios de sistema para que a atualização se faça o que, para determinados utilizadores, pode representar um grande período de tempo sem atualizar o seu *software*. Em alguns casos em particular a atualização não chega sequer a ser instalada, sem qualquer razão aparente, o que representa um perigo ainda mais significativo.

Sendo o csSECURE um produto na área da segurança, tão ligado às ferramentas de utilização diária de produção de documentação sensível e confidencial (ver secção 1.2), é da maior importância que este se mantenha atualizado.

1.4 Objetivos do Estágio

Para responder ao problema encontrado surge então o projeto *csSECURE – Software Update Service*, que tem como principais objetivos colmatar a falha que existe no mecanismo de atualizações do csSECURE (ver secção 1.2) e assegurar que o mesmo se mantém atualizado de forma segura, transparente e não invasiva.

Em suma, os principais objetivos do projeto *csSECURE – Software Update Service* são:

- Fornecer uma solução de atualizações automáticas para o produto csSECURE, de forma a garantir que o mesmo se encontra sempre atualizado em todas as máquinas onde está instalado.
- Implementar um sistema de atualizações seguro, transparente e não invasivo.

Para além destes objetivos, existem também alguns objetivos extra considerados que, caso sejam atingidos, representarão uma mais-valia para o projeto:

- Dotar a plataforma de um componente de publicação de atualizações no servidor, facilitando o processo de lançamento de versões de produto.
- Garantir que a plataforma de atualizações implementada, ao ser integrada no produto csSECURE, tem no mesmo um impacto igual ou superior a outras soluções de atualizações automáticas presentes no mercado.

- Garantir que a solução desenvolvida é genérica o suficiente para permitir atualizar outros produtos que não o csSECURE.

1.5 Estrutura do Relatório

A secção 1, Introdução, descreve o âmbito do estágio, introduz o produto cujas necessidades de atualização motivaram a existência do presente projeto, indicando o problema a resolver e os objetivos finais a atingir.

Na secção 2, Gestão de Projeto, é dada a conhecer a metodologia de desenvolvimento, a equipa de projeto e toda a calendarização do projeto de estágio, incluindo reuniões e planeamento.

A secção 3, Desenvolvimento, apresenta todas as etapas da fase de desenvolvimento do projeto, desde o estudo do estado da arte, à especificação de requisitos, arquitetura do sistema, desafios superados, testes de validação e segurança.

Na secção 4, Resultados, é detalhada toda a documentação produzida e *software* desenvolvido pelo estagiário. São também feitas algumas considerações relativamente ao trabalho futuro.

Por fim, as Conclusões, na secção 5, destacam os objetivos atingidos neste projeto de estágio e fazem um balanço dos conhecimentos e competências adquiridas pelo estagiário.

2 Gestão de Projeto

2.1 Metodologia de Desenvolvimento

A metodologia em vigor na equipa de desenvolvimento do produto csSECURE, onde o estagiário esteve integrado durante todo o período de estágio, é a metodologia de desenvolvimento ágil SCRUM, funcionando em *sprints* de duas semanas. Da mesma forma, a metodologia adotada para o desenvolvimento do presente projeto foi também a metodologia SCRUM, embora esta tenha decorrido no seu próprio âmbito, isto é, paralelamente ao trabalho desenvolvido no csSECURE. O estagiário familiarizou-se com todas as práticas de SCRUM adotadas na equipa e colocou as mesmas em prática no desenvolvimento do projeto de estágio curricular *csSECURE – Software Update Service*.

Desenvolver um produto significa, muitas vezes, ter uma massa difusa de potenciais clientes destinatários, com os quais geralmente não se fala diretamente, sendo ainda assim necessário incluir no produto tudo aquilo que os mesmos pretendem.

Desenvolver um produto significa também assumir riscos e apontar a fatores inovadores de mercado, que permitam diferenciar o produto da competição. O mundo e o mercado estão em constante movimento, em constante evolução, a competição melhora os seus produtos, os clientes alteram as suas necessidades de negócio, bem como as suas próprias necessidades. É necessária agilidade!

Como tal e respeitando a natureza desta metodologia, foi inicialmente elaborada uma primeira versão do *Product Backlog*, um documento "vivo" que contém todas as funcionalidades que se pretendem implementar no produto sob a forma de *user stories,* assim como outros requisitos não funcionais igualmente importantes. Este documento foi reavaliado a cada *sprint*, numa sessão própria para o efeito, a sessão de *Grooming*, onde as prioridades são revistas pelo *Product Owner* e as estimativas são reajustadas de acordo com o parecer de toda a equipa (apesar de este ser um projeto paralelo, nas sessões de *Grooming* do projeto *csSECURE – Software Update Service*, toda a equipa do csSECURE fez questão de estar sempre presente e contribuir para melhor reavaliar estimativas). Para além do *Product Backlog*, cada *sprint* possuiu também o seu próprio *Sprint Backlog*, que contém as *user stories* e respetivas tarefas para aquela determinada *sprint* em particular.

Ao longo do período de estágio, por *sprint*, decorreu sempre um conjunto de sessões definido pela metodologia adotada. Esse conjunto é apresentado de seguida:

- ***Planning*:** Reunião de planeamento que marca o início da *sprint*, onde são escolhidas as *user stories* de maior prioridade do *Product Backlog* (definidas pelo *Product Owner*) para serem executadas. Uma vez escolhidas as *user stories*, estas são divididas em tarefas de menor dimensão, sendo que cada tarefa é estimada em termos de tempo de execução (a estimativa é feita tendo em conta o *worst case*, *best case* e *most likely* para cada caso em particular). É nesta reunião que o *Sprint Backlog* fica definido.
- ***Daily Scrum Meetings*:** Reuniões diárias em equipa que decorrem no início do dia onde é exposto o trabalho executado no dia anterior, qual o trabalho que se irá executar no próprio dia e os objetivos com que cada elemento da equipa se compromete.
- ***Grooming*:** Sessões que decorrem uma vez a cada *sprint* para que o *Product Owner* reavale as prioridades e a equipa reajuste as estimativas das *user stories* do *Product Backlog* em termos de *story points* (medida representativa do esforço e complexidade das mesmas).
- ***Review*:** Reunião onde são apresentados resultados e o *Product Owner* avalia o *achievement* atingido pela equipa na *sprint* em questão.
- ***Retrospective*:** Reunião que marca o final da *sprint*, onde é analisado o que de melhor correu na *sprint* (*well done*), o que necessita de ser melhorado (*to improve*) e o que correu pior mas que não depende diretamente da equipa (*rants*).

Durante o tempo restante, processa-se a fase de desenvolvimento e testes, onde é executado todo o trabalho planeado no início da *sprint*.

A equipa do csSECURE sempre fez questão de acompanhar todas as sessões deste projeto de estágio em cada *sprint* (*Planning*, *Grooming*, *Review*, *Retrospective*, *Daily Scrum Meetings* e outras reuniões necessárias), permitindo ao estagiário sentir-se integrado em equipas de SCRUM de maior escala.

Os principais intervenientes de uma equipa SCRUM são os seguintes:

- ***Product Owner*:** O proprietário, que representa a parte interessada. É ele quem define as prioridades de acordo com as oportunidades de negócio que surgem. É ele quem prioriza e gere todo o *Product Backlog* do produto.
- ***Scrum Master*:** O elemento que assegura a continuidade das boas práticas de SCRUM. Tem o *skillset* necessário para blindar, guiar e motivar a equipa e poder ajudar sempre que necessário.
- ***Team*:** Grupo de elementos que analisam, planeiam, implementam e testam as funcionalidades do produto de forma iterativa (*sprints*).

Todos estes elementos funcionam de forma consonante, em ciclos iterativos (*sprints*) de duas semanas. Na figura seguinte, está representado o processo global que é seguido utilizando esta metodologia.

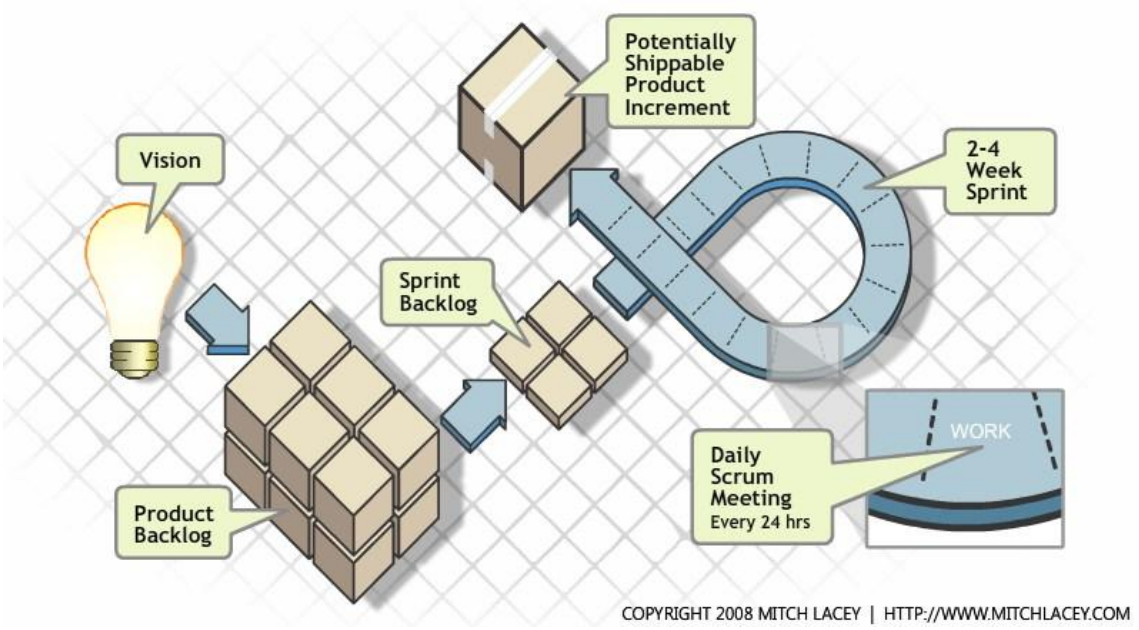


Figura 3 - Metodologia de desenvolvimento ágil SCRUM

Como se pode ver na Figura 3, toda a lógica de SCRUM tem início com uma visão, uma ideia de produto que ganha forma através de um *Product Backlog*. Este está em constante evolução sendo revisto, reavaliado e atualizado a cada *sprint* pelo *Product Owner*.

De acordo com Mitch Lacey [48], após a sua definição inicial, um *Product Backlog* é concretizado em ciclos de desenvolvimento (*sprints*) que normalmente duram duas ou quatro semanas, onde todo o trabalho (*Sprint Backlog*) é planeado e executado com o pressuposto de no final de cada ciclo entregar uma ou mais componentes funcionais do produto.

Ao longo de todo o ciclo de desenvolvimento existe diariamente uma sessão, a *Daily Scrum Meeting*, onde a equipa se reúne para avaliar o ponto da situação.

No final de cada ciclo, o *Product Owner* avalia então todo o trabalho realizado pela equipa na sessão de *Review*.

Esta metodologia foi adotada desde muito cedo no projeto sendo que, no total, foram concretizadas 18 *sprints*. O planeamento poderá ser consultado em maior detalhe na secção 2.3.3.

A descrição das práticas e ferramentas utilizadas nesta metodologia, *Product Backlog* e todos os *Sprint Backlogs* poderão ser consultados em maior detalhe, respetivamente, nos documentos anexos Metodologia de Desenvolvimento [1], Especificação de Requisitos [4] e *Sprint Backlogs* [2].

2.2 Equipa

Sendo o *csSECURE – Software Update Service* um projeto de estágio, este decorreu paralelamente ao trabalho desenvolvido no csSECURE. Assim sendo, a equipa que compõe este projeto é constituída apenas por dois elementos:

- Estagiário Tiago Carregã – Investigação e Desenvolvimento;
- Eng.º Bernardo Patrão – *Product Owner*, *Scrum Master* e responsável por SQA (*Software Quality Assurance*).

É de referir também que o estagiário esteve em contacto regular com o *Product Owner* do csSECURE, para que este pudesse também perceber até que ponto este projeto seria exequível e quando seria possível obter uma primeira versão de avaliação, para atualizar o csSECURE em máquinas pertencentes ao domínio da Critical Software.

2.3 Calendarização e Acompanhamento

2.3.1 Reuniões de projeto

Dada a metodologia utilizada, as reuniões de projeto foram bastante regulares, sendo realizadas semanalmente, por vezes mais do que uma vez, quando executadas também nas sessões de *Grooming*. Nestas reuniões eram avaliados os riscos do projeto, discutidos detalhes técnicos e possíveis abordagens alternativas, assim como eram reavaliadas as prioridades e estimativas das *user stories* em *Product Backlog*.

2.3.2 Reuniões de estágio

As reuniões de estágio, em conjunto com ambos os orientadores (Doutor Edmundo Monteiro e Eng.º Bernardo Patrão) decorreram pontualmente ao longo do projeto de estágio, sempre que foi necessário e em alturas estratégicas, de forma a dar a conhecer o desenvolvimento do projeto e para que este pudesse ser enriquecido com as sugestões e ideias daí resultantes. Alguns pontos de viragem no rumo do projeto surgiram destas reuniões.

2.3.3 Planeamento

O projeto *csSECURE – Software Update Service*, conta com um plano de execução dividido em *workpackages* que é executado, atualizado e reajustado ao longo do tempo. Cada *workpackage* é concretizada na implementação de funcionalidades e requisitos não funcionais. Dada a natureza da metodologia utilizada, o planeamento

está sujeito a alterações a cada *sprint*. Uma vez reavaliadas as estimativas e prioridades nas sessões de *Grooming*, surge então uma versão atualizada do *Product Backlog*. De seguida são apresentados o planeamento inicial e final, para se poder obter um termo de comparação e observar as principais alterações que surgiram com o decorrer do projeto, em termos de *workpackages*.

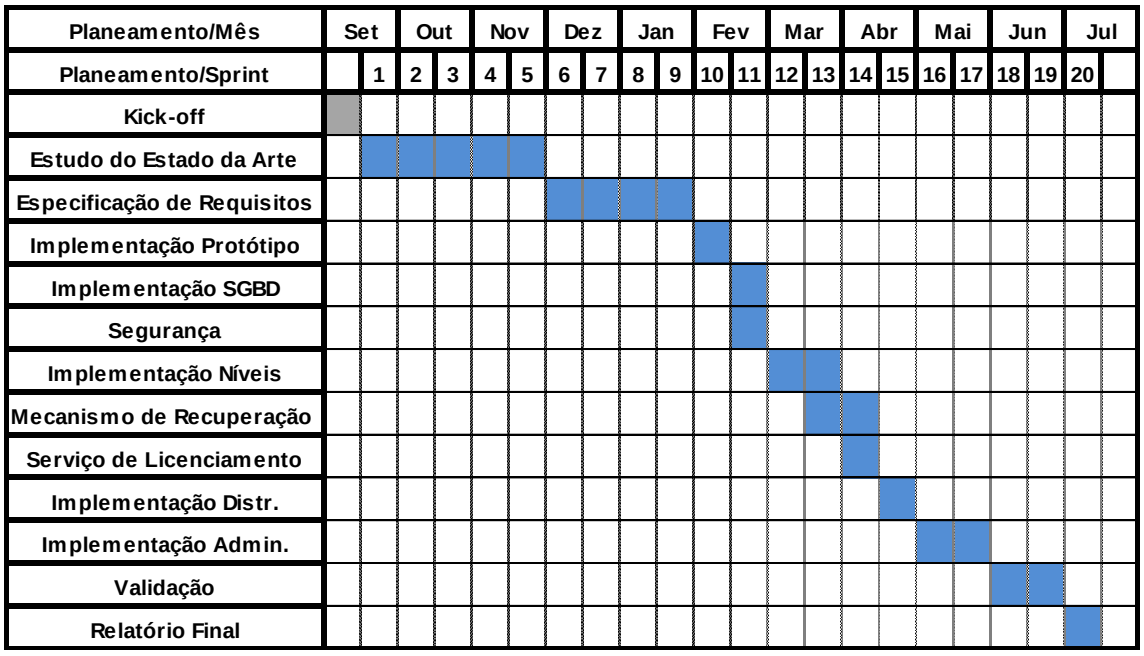


Figura 4 - Planeamento inicial do projeto de estágio

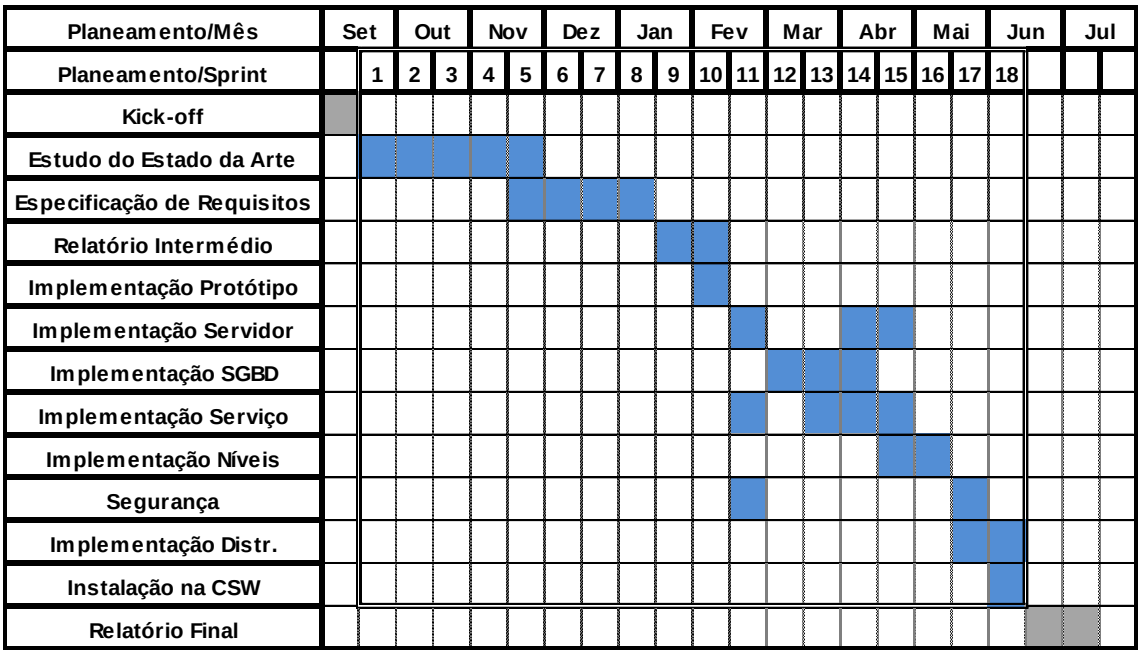


Figura 5 - Planeamento final do projeto de estágio

Como se pode observar, do planeamento inicial ao final surgiram algumas alterações relativamente à ordem e/ou inclusão de determinados *workpackages* bem como ao conjunto total de *sprints* previsto. Esta evolução é da responsabilidade do *Product Owner* e reflete o resultado da sua reavaliação do *Product Backlog* ao longo de todas as *sprints* do projeto, através das sessões de *Grooming*. No planeamento final, a área central realçada (*sprints* 1 a 18) define o período durante o qual o estágio decorreu em SCRUM, onde as fases de *kick-off* e documentação do relatório final de estágio não estão incluídas, sendo o respetivo trabalho planeado e realizado pelo estagiário.

***Workpackages*:** Unidades que representam diferentes fases do projeto. O planeamento final do *csSECURE – Software Update Service* é constituído por 13 *workpackages*, que de seguida são apresentados:

- ***Kick-off*:** Fase inicial do projeto onde o estagiário se familiariza com o problema a resolver.
- **Estudo do Estado da Arte:** Levantamento de soluções na área das atualizações automáticas de *software* já existentes no mercado, acompanhado de um estudo aprofundado daquelas que foram consideradas como as mais relevantes.
- **Especificação de Requisitos:** Recolha de uma lista de requisitos do projeto junto do cliente (Critical Software), concretizada numa primeira versão de *Product Backlog*. Ainda nesta fase são especificados a arquitetura e o desenho detalhado da solução.
- **Implementação Protótipo:** Implementação de um pequeno protótipo de uma solução de atualizações automáticas, já com cliente e servidor, com o cliente apenas a verificar periodicamente a existência de novas atualizações e, caso existam, forçando a instalação.
- **Relatório Intermédio:** Elaboração do relatório intermédio de estágio.
- **Implementação Servidor:** Implementação do Webservice e servidor Web HTTP e integração dos mesmos com o sistema de gestão de bases de dados.
- **Implementação SGBD:** Implementação de todo o sistema de gestão de bases de dados da plataforma, do lado do servidor, para permitir armazenar todos os dados necessários ao funcionamento do mesmo.
- **Implementação Serviço:** Implementação das duas componentes serviço e aplicação (o serviço corre no contexto do sistema, a aplicação corre no contexto do utilizador com componente gráfica) que, ao comunicarem em conjunto, formam a aplicação cliente.
- **Implementação Níveis:** Implementação de dois modos distintos de funcionamento, que variam de acordo com o nível de criticidade atribuído ao

pacote de atualização onde, caso existam recursos bloqueados a impedir a instalação, no modo intrusivo (para atualizações consideradas críticas) é pedido ao utilizador que termine e guarde todo o seu trabalho para que a instalação se faça e, no modo não-intrusivo, a plataforma simplesmente aguarda até que os recursos sejam desbloqueados sendo que, caso estes não o sejam ao fim de determinada janela temporal, a instalação é agendada para o reiniciar da máquina.

- **Segurança:** Implementação dos mecanismos de segurança necessários à plataforma como SSL/TLS, autenticação integrada e assinaturas digitais.
- **Implementação Distribuição:** Implementação da consola de distribuição, uma aplicação Web onde é possível publicar novas atualizações do produto.
- **Instalação na CSW:** Fase de integração do projeto na Critical Software, com instalação do servidor no sistema de informação da organização e instalação da aplicação cliente nas máquinas dos *beta-testers*.
- **Relatório Final:** Fase de elaboração do relatório final de estágio.

Os *workpackages* inicialmente previstos “Mecanismo de recuperação” e “Serviço de licenciamento” foram retirados do planeamento no decorrer do projeto visto terem deixado de fazer sentido no âmbito do projeto. O primeiro, “Mecanismo de recuperação”, referente à recuperação de erros de instalação, foi reavaliado quanto à sua prioridade e, visto a plataforma funcionar de forma automática e os instaladores serem criados num ambiente controlado, possuindo o seu próprio mecanismo de *rollback*, perdeu relevância, sendo relegado para trabalho futuro. O segundo, “Serviço de licenciamento”, foi mesmo excluído do projeto. Após uma reavaliação da necessidade deste componente na arquitetura da plataforma, concluiu-se que era redundante e apenas acrescentaria *overhead* desnecessário ao funcionamento da plataforma. Além disso, caso se pretendesse distribuir outro tipo de produto que não o csSECURE, seria necessário criar serviços de licenciamento para cada produto distinto, tornando assim a plataforma pouco convidativa uma vez que obrigaria a trabalho extra para que esta entrasse em funcionamento.

O *workpackage* “Implementação Admin.” que estava previsto no planeamento inicial e consiste na implementação da consola de administração, foi também relegado para trabalho futuro uma vez reavaliada a sua prioridade no decorrer do projeto. Como etapa inicial, foi decidido implementar a consola de distribuição para poder colocar o mais rapidamente possível a plataforma de atualização em funcionamento na Critical Software, distribuindo novas versões do csSECURE para todas as máquinas do domínio com o cliente csSECURE instalado.

***Milestones*:** Dado este ser um projeto de estágio, para além das reuniões definidas no âmbito da metodologia utilizada, foi ainda realizado um conjunto de reuniões formais de controlo com ambos os orientadores de estágio, onde o principal foco de avaliação foi o progresso do projeto.

- ***Kick-Off Meeting*:** Reunião que marcou o início do projeto. Foi definida uma primeira versão do plano e discutida a estratégia inicial de abordagem ao problema.
- **Reunião de Controlo 1:** Reunião cujo principal objetivo passou pela revisão conjunta das *user stories* e requisitos do projeto.
- **Reunião de Controlo 2:** Reunião que marcou a fase final do desenho detalhado, onde foi revista a arquitetura da aplicação.
- ***Project Close Meeting*:** Reunião que decorreu já na fase final do período de estágio, cujo objetivo passou por validar as ações de fecho do projeto e retirar conclusões sobre a fase de desenvolvimento, quer ao nível da metodologia como ao nível técnico. Foi também feito um balanço final e uma avaliação sobre o cumprimento ou não dos objetivos inicialmente propostos.

***Deliverables*:** Conjunto de componentes a entregar ao longo do projeto.

- **Relatório de Estudo do Estado da Arte:** Trata-se de um levantamento das soluções *open-source* e comerciais existentes na área das atualizações automáticas de *software*, bem como das tecnologias mais utilizadas.
- **Especificação de Requisitos:** Especificação detalhada de todo o *Product Backlog* e requisitos não funcionais do *csSECURE – Software Update Service*.
- **Especificação de Casos de Teste de Aceitação:** Documento que apresenta a especificação dos casos de teste que permitem validar *user stories* e requisitos não funcionais do sistema.
- ***Sprint Backlogs*:** Documento que contém um resumo detalhado de todas as *sprints* executadas ao longo do período de estágio no âmbito da metodologia SCRUM.
- **Plano de Desenvolvimento do *Software*:** Documento que apresenta uma justificação para a escolha da metodologia de desenvolvimento ágil SCRUM no csSECURE e respetivos projetos de estágio, assim como o processo que permite concretizá-la e quais as ferramentas utilizadas para o efeito.
- **Versão de Pré-Produção do *csSECURE - Software Update Service*:** Instaladores do servidor e serviço de atualizações, para poder instalar os mesmos, respetivamente, no servidor e máquinas dos utilizadores (*beta-testers*) da Critical Software.

- **Relatório Final de Estágio:** Relatório que compila a informação de maior relevância a registar (suportada pela documentação técnica em anexo) no âmbito do projeto *csSECURE – Software Update Service*.

3 Desenvolvimento

3.1 Estudo do Estado da Arte

3.1.1 Introdução

O objetivo desta primeira fase do projeto consistiu numa análise de mercado, identificando as soluções existentes na área das atualizações automáticas de *software*. Após um longo período de pesquisa, foram compilados os resultados que se seguem e que poderão ser consultados em maior detalhe no documento anexo de Estudo do Estado da Arte [3].

De seguida, são apresentados o critério de seleção e parâmetros de comparação utilizados na análise de mercado, assim como um pequeno resumo daquelas que foram as soluções identificadas.

Posteriormente são apresentadas as conclusões gerais retiradas deste estudo, acompanhadas de uma matriz comparativa onde é possível obter um panorama global das diferenças entre cada uma das soluções identificadas ao nível dos sistemas suportados, segurança e funcionalidades mais relevantes.

3.1.2 Critério de seleção e parâmetros de comparação

O csSECURE procura uma solução para se manter atualizado de forma automática, segura, transparente, não intrusiva e independente dos privilégios do utilizador da máquina. É necessário garantir que os pacotes de atualizações são de origem segura, que não foram comprometidos no processo de transferência para os clientes e que a instalação das mesmas não interfere com a normal utilização da máquina a não ser em caso de extrema necessidade (tais como atualizações críticas relacionadas com falhas de segurança), tudo isto independentemente de o utilizador possuir, ou não, direitos de administração.

Assim sendo, dentro da área das atualizações de *software*, o principal critério de seleção que filtrou as escolhas que de seguida serão identificadas consiste na capacidade da solução permitir processos de atualização automáticos, ou seja, apenas as soluções que permitem atualizações automáticas foram selecionadas.

Após esta seleção criteriosa, os parâmetros de comparação entre as soluções identificadas foram escolhidos pela proximidade com as características e funcionalidades pretendidas para a solução a implementar no âmbito do projeto *csSECURE – Software Update Service*:

- Relativamente aos sistemas operativos suportados e uma vez que o csSECURE se baseia fortemente em tecnologias Microsoft, apenas foram selecionadas soluções que suportam o sistema operativo Windows, utilizando apenas como parâmetro de comparação a versão do mesmo.

- Na área da segurança, são utilizados como parâmetros de comparação os mecanismos de segurança mais comuns em soluções de atualizações de *software* como os tipos de autenticação e encriptação utilizados.

- Ao nível das funcionalidades, foram selecionadas para efeitos de comparação aquelas que mais impacto terão numa solução de atualizações automáticas de *software*, como o tipo de atualização que é feito, a transparência de todo o processo, o nível de intrusão que uma simples atualização tem na utilização da máquina, a possibilidade de correr como serviço e permitir instalar atualizações em máquinas onde o utilizador não possui direitos de administração, entre outras.

3.1.3 Soluções open-source

As soluções *open-source*, isto é, de código livre, identificadas no âmbito das atualizações automáticas de *software* foram as seguintes:

- *Sharp AutoUpdater*,
- *NAppUpdate*,
- *wyUpdate & AutomaticUpdater Control*.

Todas estas soluções foram desenvolvidas na linguagem de programação C#, com o intuito de serem integradas em aplicações .NET, para assim permitir que as mesmas ofereçam a funcionalidade de atualizações automáticas.

Num breve resumo destas soluções, temos então a *Sharp AutoUpdater*, que permite adicionar um mecanismo de atualizações automáticas a aplicações .NET e está ainda num estado de desenvolvimento muito prematuro, a *NAppUpdate* que até ao momento e assim como a solução anterior apenas pode ser integrada em aplicações .NET (mais concretamente *Windows Forms* e *Windows Presentation Foundation*) apresentando um reduzido conjunto de funcionalidades e, finalmente, a *wyUpdate & AutomaticUpdater Control*, que das três, é aquela que apresenta um maior nível de maturidade, não só pelo facto de ser possível integrar em qualquer tipo de aplicação como pelo tipo de segurança que apresenta, entre outras funcionalidades.

Uma vez analisadas em maior detalhe as desvantagens de cada solução, podemos apontar como principal desvantagem o facto de nenhuma destas permitir a instalação

de atualizações em máquinas onde o utilizador não possui direitos de administração. Devido ao facto de o processo de atualização estar integrado na própria aplicação, este corre no contexto do utilizador que a iniciou, o que despoleta um pedido de elevação de privilégios (Windows *User Account Control*) durante uma tentativa de instalação.

No caso concreto do csSECURE, existem clientes cuja maioria dos utilizadores não possui direitos de administração sobre a própria máquina, quer por motivos de segurança ou qualquer outra política interna à organização.

Sendo o csSECURE um produto com uma forte presença no dia-a-dia de cada utilizador no interior de uma organização e dada a sua integração em ferramentas utilizadas diariamente, uma das principais preocupações é reduzir o nível de intrusão, de modo a não interferir com a normal utilização da máquina. Assim sendo, é esperado o mesmo tipo de cuidado e preocupação no que toca ao mecanismo de atualizações do próprio produto.

Qualquer uma das soluções de atualização acima identificadas não possui um mecanismo que permita instalar as atualizações sem interromper o trabalho do utilizador. Desta forma, para além da já referida restrição ao tipo de aplicação onde estas soluções podem ser integradas, seria muito pouco recomendável a integração de qualquer uma delas no csSECURE, uma vez que não preenchem os principais requisitos pretendidos.

3.1.4 Soluções comerciais

Quanto às soluções comerciais presentes no mercado na área das atualizações automáticas de *software*, foram identificadas as seguintes:

- *TrueUpdate*,
- *The Software Update Wizard*,
- *AutoUpdate Plus*,
- *AppLife Update*.

Destas quatro soluções, apenas a *AppLife Update* foi criada com o intuito de ser integrada em aplicações .NET, a *AppLife Update*.

Naturalmente que por todas estas soluções serem comerciais, estamos perante a grande desvantagem de depender de preços e licenciamentos que podem por si só motivar a sua exclusão. Ainda assim, vale a pena registar algumas daquelas que foram consideradas como as informações relevantes que surgiram no decorrer do estudo do estado da arte, neste sector de mercado em particular.

De um modo geral, todas estas soluções permitem adicionar a funcionalidade de atualizações automáticas a uma aplicação através da integração direta, onde a aplicação invoca a ferramenta de atualizações algures no código, sendo que no cenário mais comum a ferramenta é invocada a partir de algum botão de atualizações integrado na aplicação. Para automatizar o processo, teria de ser implementado na aplicação um mecanismo adicional que invocasse periodicamente a ferramenta de atualizações de forma automática. Esta é aliás uma característica comum nas soluções analisadas, que advém do facto de estas serem integradas diretamente nos produtos.

O csSECURE pretende uma solução que atualize o produto funcionando de forma independente e isolada, não estando diretamente integrada no mesmo.

De todas estas soluções e apesar de nenhuma reunir todas as condições para equacionar a sua integração com o csSECURE, houve uma em destaque pelas suas funcionalidades e pela arquitetura interna apresentada, que permite responder aos desafios impostos pelos requisitos pretendidos no processo de atualização do csSECURE, a *The Software Update Wizard*.

Para fazer face à problemática da instalação de atualizações em máquinas onde o utilizador não possui privilégios de administração, esta solução apresenta uma arquitetura interessante onde o serviço de atualizações é dividido em duas componentes, um serviço e uma aplicação, sendo que ambos são executados em contextos de segurança diferentes. O serviço executa no contexto de segurança do sistema e a aplicação executa no contexto do utilizador.

Os serviços que correm no contexto de segurança do sistema permitem instalar atualizações com elevação de privilégios, no entanto, não são autorizados a comunicar graficamente com o utilizador.

Já as aplicações a correr no contexto de segurança do utilizador, podem comunicar graficamente com o mesmo, no entanto sujeitam-se aos privilégios que este possui. Caso o utilizador não possua certos privilégios, não será possível executar uma instalação, impedindo assim o produto de ser atualizado.

Assim sendo, realçando o fator inovador desta arquitetura, o conjunto destes dois componentes permite executar a instalação no serviço no contexto de segurança do sistema (isto é, com permissões de administrador), ao mesmo tempo que é conseguida a comunicação com o utilizador graficamente através da aplicação que executa no contexto de segurança do mesmo. Estes dois componentes em conjunto compõem o serviço de atualizações do *csSECURE – Software Update Service.* Na figura seguinte

está representada a referida arquitetura de dois componentes, no contexto da solução *The Software Update Wizard*.

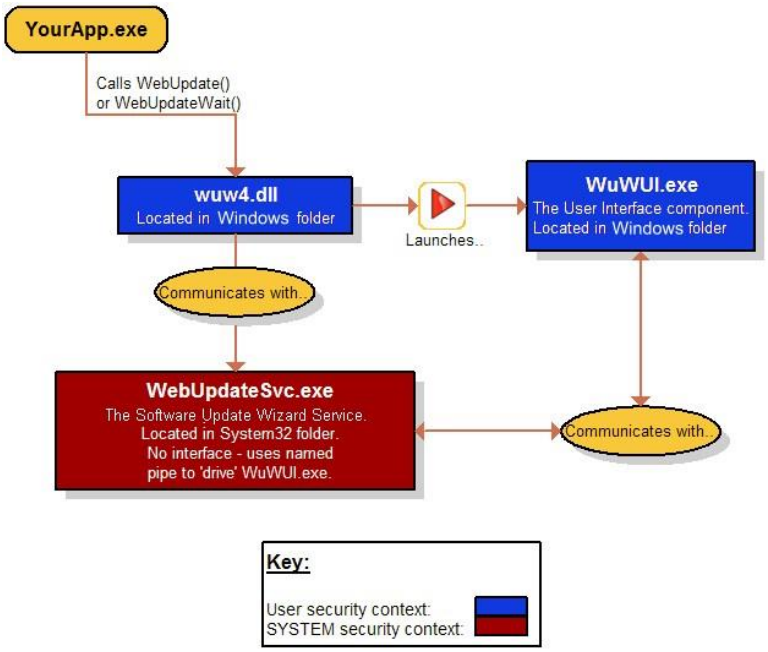


Figura 6 - Arquitetura da solução The Software Update Wizard

http://www.powerprogrammer.co.uk/helpfile/Software%20Update%20Wizard/architecture.htm

Como podemos ver, na Figura 6 existe uma separação de contexto onde serviço (*WebUpdateSvc.exe*) e aplicação (*WuWUI.exe*), apesar de separados, concretizam todas as etapas do processo de atualizações.

Ainda assim, com a *The Software Update Wizard* a perfilar-se como a mais atraente da gama de soluções comerciais analisadas, esta encontra-se sujeita a licenciamento limitado, renovável e de preço relativamente elevado o que desde logo a torna muito pouco apelativa.

Em suma, apesar de nenhuma destas soluções ser utilizada pelas demais razões referidas anteriormente, da análise deste sector de mercado e com a descoberta da arquitetura desta solução em particular surgiu um conjunto de ideias interessantes para o desenvolvimento de uma solução de raiz.

A análise detalhada de cada uma destas soluções poderá ser encontrada no documento em anexo de Estudo do Estado da Arte [3].

3.1.5 Conclusões

De seguida é apresentada uma matriz de características e funcionalidades, onde é possível analisar comparativamente e de forma global um resumo de todas as soluções

identificadas, com o intuito de agregar toda a informação relevante que permita clarificar as conclusões retiradas deste estudo.

		Soluções open-source			Soluções comerciais			
		NAppUpdate	Sharp AutoUpdater	wyUpdate + Automatic Updater Control	TrueUpdate	AutoUpdate Plus	The Software Update Wizard	AppLife Update
Sistema Operativo	MS Win 98ME				✓	✓	✓	
	MS Win NT 4.0				✓	✓		
	MS Win 2000	✓	✓	✓	✓	✓	✓	✓
	MS Win XP	✓	✓	✓		✓	✓	✓
	MS Win Server 2003			✓		✓	✓	
	MS Win Vista	✓	✓	✓	✓	✓	✓	✓
	MS Win Server 2008			✓		✓		
	MS Win 7	✓	✓	✓	✓	✓	✓	✓
Segurança	SSL/HTTPS			✓	✓	✓		
	Base64 encoding				✓			
	SHA256 checksums	✓						
	MD5 checksums				✓		✓	
	Autenticação HTTP				✓	✓	✓	✓
	Autenticação FTP				✓	✓	✓	✓
	Strong names	✓						
	Cert. X509		✓					
	Chaves assimétricas					✓		✓
	Encriptação Blowfish				✓			
	Encriptação RSA			✓				
	Encriptação não revelada						✓	
Sistemas	Integração exclusiva .NET	✓	✓					✓
	32 bits	✓	✓	✓	✓	✓	✓	✓
	64 bits	✓		✓		✓	✓	✓
	Com Windows UAC	✓	✓	✓	✓	✓	✓	✓
Funcionalidades	Atualiza Serviços			✓		✓		
	Atualiza Bases de Dados			✓				✓
	Exec. ficheiros após instalação			✓	✓	✓	✓	✓
	Permite escalonamento	✓		✓		✓	✓	✓
	Instala em modo silencioso			✓	✓	✓	✓	✓
	Não força término da aplicação							
	Instala sem permissões admin.					✓	✓	✓
	Executa em modo serviço					✓	✓	✓
	Não depende de scripting	✓	✓	✓		✓		✓
Linguagem de Desenvolvimento		C#	C#	C#	C/C++	-	C/C++/C#	-

Figura 7 - Matriz comparativa

Esta matriz encontra-se dividida em cinco secções, onde são analisados os sistemas operativos suportados, mecanismos de segurança utilizados, algumas das características e principais funcionalidades desejadas e a linguagem ou linguagens de desenvolvimento de cada uma das soluções previamente identificadas. As motivações para a escolha dos parâmetros de comparação entre as soluções identificadas e correspondentes entradas nesta matriz foram já descritas anteriormente na secção 3.1.2. Adicionalmente estão também presentes as linguagens de desenvolvimento de algumas das soluções identificadas (em duas das soluções não foi possível obter essa informação), para poder perceber quais são as tecnologias mais utilizadas na implementação de soluções deste tipo - uma componente igualmente importante no estudo do estado da arte.

Finalizando este estudo, podemos concluir que nenhuma das soluções identificadas no mercado serve por completo o conjunto de requisitos do csSECURE.

Quer por limitações nas funcionalidades, restrições no tipo de integração ou até preços de licenciamento elevados, nenhuma das soluções analisadas reuniu todas as condições pretendidas, o que motivou o desenvolvimento de uma solução de raiz para manter o csSECURE atualizado de forma automática.

Ainda assim, este estudo revelou-se bastante proveitoso ao permitir retirar uma boa ideia daqueles que são os pontos menos fortes das soluções presentes no mercado e sobretudo das características que não se pretendem incluir na implementação de uma solução desta natureza.

As principais ideias que ficaram e que serviram como base para a especificação de requisitos foram as seguintes:

- Pretende-se uma solução de atualizações automáticas isolada, que funcione de forma isolada e independente. Soluções de integração no produto são pouco atrativas. Desta forma e por uma questão de simplicidade para o utilizador, é mais simples apenas ter de instalar a plataforma de atualizações para manter o csSECURE atualizado.
- As atualizações deverão ser feitas de forma totalmente automática, sem dependerem da iniciativa ou intervenção do utilizador. O objetivo aqui é retirar ao utilizador a responsabilidade de manter o próprio produto atualizado, focando a sua atenção em outros pontos de maior interesse pessoal ou profissional.
- A solução deverá permitir instalar atualizações mesmo em máquinas onde o utilizador não possui privilégios de administração. Este será um requisito essencial para competir de forma mais agressiva com a concorrência. Como vimos na matriz representada na Figura 7, apenas três soluções comerciais apresentam esta funcionalidade.
- O processo de atualização deverá ser totalmente transparente, sem incomodar o utilizador que assim mantém o produto atualizado de forma regular sem interrupções na utilização da máquina, recebendo apenas uma notificação assim que o processo é concluído.
- A segurança terá de ser alvo de especial atenção, para que esta seja uma solução competitiva capaz de ultrapassar as soluções concorrentes nesse campo em particular. Com isto pretende-se dar a confiança e noção de segurança necessárias ao cliente para que este se sinta motivado a apostar nesta solução.

Após este estudo surgiu também uma ideia clara do modelo de arquitetura a implementar. Tendo como base a arquitetura da solução *The Software Update Wizard*, verificou-se que o modelo conjunto serviço-aplicação seria de facto a melhor abordagem para a solução a desenvolver, pelas vantagens que cada um dos dois componentes oferece. Por um lado o serviço, que executado no contexto do sistema permite instalar atualizações em máquinas onde o utilizador não possui privilégios de administração, por outro a possibilidade de comunicar com o utilizador através da aplicação (entre outras), nenhum deles podia ser descurado sob o risco de comprometer a qualidade e funcionalidade da plataforma de atualizações.

Assim sendo, de seguida é apresentada a especificação de requisitos do *csSECURE – Software Update Service*.

3.2 Especificação de Requisitos

3.2.1 Introdução

Dada a metodologia de desenvolvimento seguida (SCRUM), todas as funcionalidades pretendidas são inseridas num *Product Backlog* sob a forma de *user stories* (ver secção 3.2.3), podendo ser representadas visualmente em diagramas de casos de uso (ver secção 3.2.4) e fazendo-se acompanhar de uma listagem de requisitos não funcionais (ver secção 3.2.6) que definem outras características de natureza não funcional do sistema.

Esta especificação de requisitos tem como principais objetivos definir e detalhar o *csSECURE – Software Update Service* na sua totalidade. No entanto, no caso particular do *Product Backlog* (ver secção 3.2.3), devido à extensão do mesmo, apenas serão identificadas as *user stories* realizadas durante a fase de implementação do presente projeto. Os demais casos de uso e requisitos não funcionais serão identificados na sua totalidade, nas respetivas secções. O *Product Backlog* completo encontra-se especificado no documento anexo de Especificação de Requisitos [4].

De seguida é dada uma breve noção da arquitetura do sistema, para permitir uma melhor compreensão da posterior especificação de requisitos.

3.2.2 Breve noção da arquitetura do sistema

O *csSECURE – Software Update Service* funciona com base no modelo cliente-servidor, onde a ideia central é a de que o distribuidor (equipa de desenvolvimento) lança uma nova versão de produto e publica a mesma na plataforma de atualizações, neste caso um servidor central ao qual os clientes têm acesso (por clientes entenda-se clientes do csSECURE, organizações onde o csSECURE está instalado). Esses clientes, instalando *o csSECURE – Software Update Service* no próprio sistema de

informação, passarão também eles a ter um servidor de atualizações (gerido por um administrador), que assume dois papéis distintos: cliente, ao verificar periodicamente e de forma automática no servidor central se existem novas atualizações publicadas (transferindo as mesmas, caso existam) e servidor, onde os serviços de atualização instalados nas máquinas dos utilizadores acedem, para que possam manter o csSECURE atualizado.

Assim sendo, uma vez publicada uma atualização no servidor central pelo distribuidor, esta é automaticamente disponibilizada aos clientes do csSECURE, que assim terão no seu próprio servidor local uma nova versão disponível, a aguardar a aprovação do administrador, para que esta seja finalmente distribuída de forma automática pelas máquinas dos utilizadores.

Na figura seguinte é representada uma vista física, de alto nível, de toda a arquitetura do sistema *csSECURE – Software Update Service*.

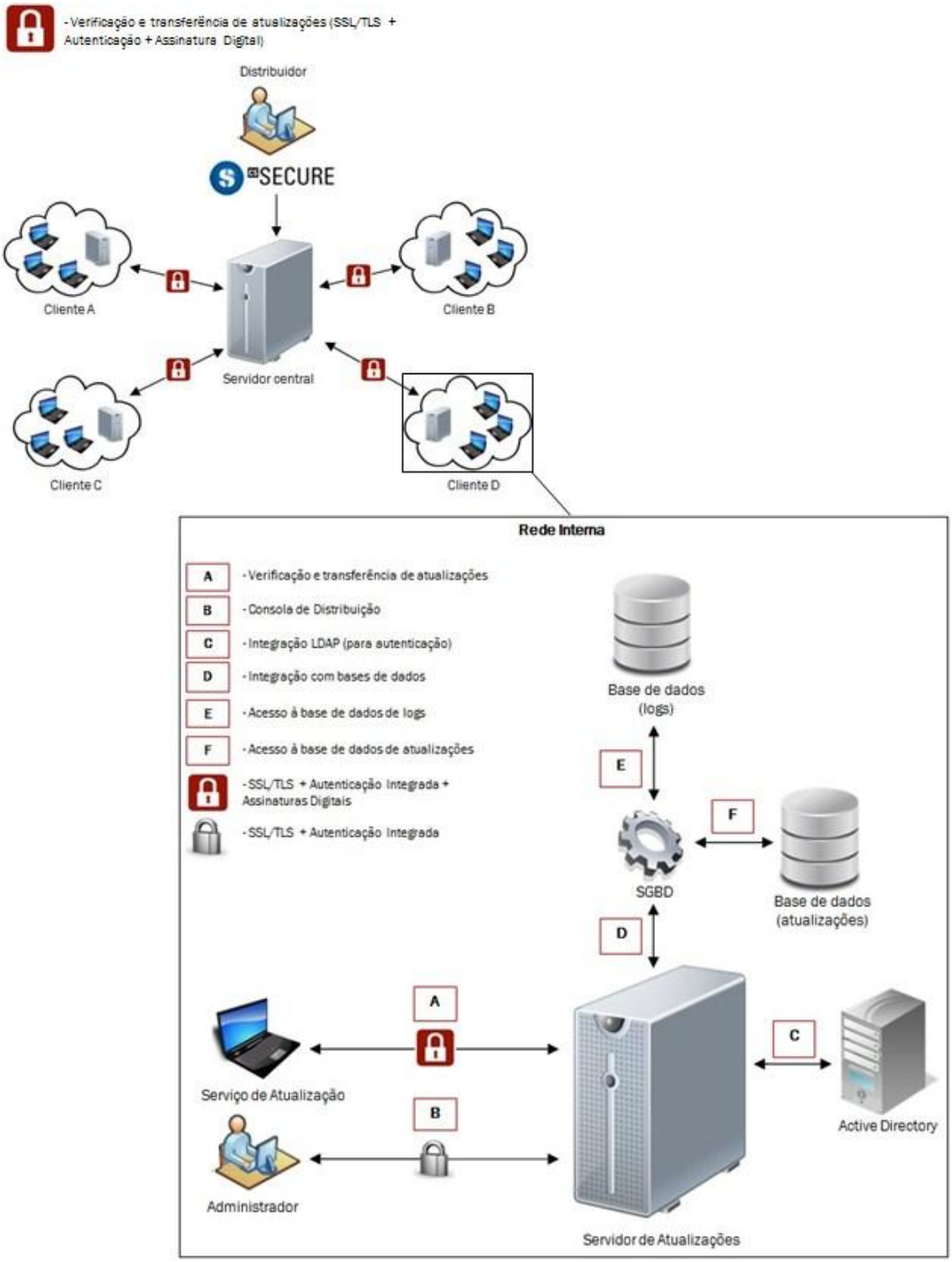


Figura 8 - Vista física de alto nível da arquitetura global do sistema

Como podemos ver, a Figura 8 representa a vista física de todo o sistema, desde o cenário inicial onde o distribuidor publica uma nova atualização no servidor central aos clientes que periodicamente verificam a existência de novas atualizações,

descarregando as mesmas para o servidor local, que assim ficam a aguardar a aprovação do administrador, para finalmente serem distribuídas pelas máquinas do domínio.

Existem três diferentes tipos de ator que interagem diretamente com o *csSECURE – Software Update Service*:

- Distribuidor
- Administrador
- Utilizador (*Workstation*)

O distribuidor, como já foi referido na descrição do funcionamento do sistema, representa a equipa de desenvolvimento do produto, isto é, quem lança as novas versões e as publica na plataforma para que estas sejam disponibilizadas aos clientes.

O administrador é o elemento responsável por gerir a plataforma nos clientes do csSECURE, validando as novas atualizações que são descarregadas para o servidor local, entre outras tarefas de administração.

Por utilizador entenda-se o utilizador final, aquele que vê o produto csSECURE ser atualizado na sua própria máquina (*workstation*).

Os serviços de atualizações instalados nas máquinas dos utilizadores poderão também ser vistos como um ator que interage com o sistema cliente, ou seja, toda a estrutura da plataforma de atualizações instalada no cliente.

Os detalhes funcionais e as várias visões da arquitetura do sistema são descritos de forma mais aprofundada na secção 3.3. De seguida, é então apresentada uma parte do *Product Backlog* do *csSECURE – Software Update Service*.

3.2.3 Product Backlog

Um *Product Backlog* é um "documento vivo", em constante renovação ao longo do tempo e em cada *sprint*, que basicamente é constituído por *user stories*. As *user stor*ies traduzem funcionalidades do sistema e tipicamente são definidas da seguinte forma:

Um <utilizador> pretende obter uma <funcionalidade> para alcançar um <resultado>.

De acordo com Mitch Lacey [48], uma *user story* deve ser simples e clara, pequena o suficiente para ser exequível numa *sprint*, deve identificar uma funcionalidade específica do sistema, poder ser testada e acima de tudo deve adicionar valor ao produto, sendo facilmente identificada pelo *Product Owner*.

Associada às *user stories* existe uma medida de referência universal que permite ao *Product Owner* e a toda a equipa de desenvolvimento ter uma noção de alto nível sobre o esforço envolvido nas mesmas: *story points*. Estes *story points* refletem o esforço necessário em termos de tempo e de complexidade para a execução da *user story* e são estimados nas sessões de *Grooming* (sessões que decorrem uma vez por *sprint* onde o *Product Owner* reavalia prioridades e a equipa estima o esforço necessário para as *user stories* em *backlog*, em termos de *story points*).

Para uma boa perceção do *Product Backlog* que de seguida será apresentado, é importante explicitar e detalhar claramente aqueles que são os quatro tipos de utilizador distintos identificados ao longo das *user stories*:

- *Product Owner*: O dono de produto, responsável pelo negócio.
- Utilizador: O utilizador final da máquina onde o serviço de atualizações está instalado.
- Distribuidor: A equipa de desenvolvimento que lança as versões de produto.
- *Developer*: O programador que implementa a solução.

De seguida são então apresentadas as *user stories* realizadas durante a fase de implementação deste projeto, pela ordem em que foram realizadas e acompanhadas dos respetivos *story points*.

Código	User story	Story points
#E04.US01	Como Product Owner, quero ter um protótipo do Software Update Service funcional, para prova de conceito.	13
#E03.US01	Como utilizador, quero que a plataforma instale uma nova atualização (MSI), para atualizar o meu produto.	8
#E03.US02	Como utilizador, quero ter a garantia de que a verificação de novas atualizações é segura, para me sentir seguro ao usar uma plataforma automática (SSL).	5
#E01.US01	Como distribuidor, quero armazenar dados sobre as atualizações de forma persistente, para que o servidor mantenha o estado dos pacotes de atualização.	10
#E04.US02	Como Product Owner, quero robustecer o Software Update Service, para obter um funcionamento consistente.	8
#E01.US02	Como distribuidor, quero que a plataforma gere respostas aos pedidos dos clientes de forma automática, para que a informação sobre as atualizações lhes seja disponibilizada.	5
#E01.US03	Como distribuidor, quero poder publicar atualizações na plataforma, para as disponibilizar aos clientes.	5

#E01.US04	Como distribuidor, quero poder definir a instrução de instalação de ficheiros .msi e respetiva ordem, para instruir o sistema cliente na instalação.	5
#E04.US03	Como developer, quero simular o processo de deteção de recursos bloqueados de um instalador MSI, para que a deteção seja feita antes da execução do instalador.	8
#E03.US03	Como utilizador, quero que a plataforma não interrompa o meu fluxo de trabalho para instalar uma atualização, para não ter de interromper tarefas vitais.	13
#E03.US04	Como utilizador, quero ter a garantia da proveniência dos pacotes de atualização, para me sentir seguro quanto à origem dos mesmos.	8
#E01.US05	Como distribuidor, quero poder listar as versões instaladas nos clientes, para poder avaliar o desempenho da plataforma em termos de distribuição do csSECURE nas máquinas cliente.	8
#E01.US06	Como distribuidor, quero as consolas de distribuição de acordo com os standards gráficos da Critical Software e do csSECURE, para proceder à utilização das mesmas.	8
#E04.US04	Como Product Owner, quero poder ter uma primeira versão de release do Software Update Service, para o poder testar nos beta-testers da Critical Software.	8

Tabela 1 - User stories realizadas do Product Backlog

Cada *user story* presente na Tabela 1 pressupõe a implementação de componentes em falta para a conclusão da mesma. De acordo com Mitch Lacey [47], esta é uma prática comum em SCRUM que permite a definição de *user stories* de forma isolada.

Assim, durante a fase de implementação, surgiram *user stories* onde acabaram por ser implementadas múltiplas funcionalidades. Para melhor representar as funcionalidades implementadas em cada uma destas *user stories* foi criada uma tabela de tarefas planeadas, associadas a cada uma das *user stories* apresentadas, que poderá ser consultada em anexo no documento Especificação de Requisitos [4], assim como uma matriz de rastreabilidade entre *user stories* e casos de uso, que poderá ser consultada na secção 3.2.5.

Estas *user stories*, apresentadas na Tabela 1, foram escolhidas de acordo com as prioridades definidas pelo *Product Owner* para que fosse possível lançar o mais depressa possível uma versão do *csSECURE – Software Update Service* na Critical Software, garantindo assim a devida atualização do produto nas máquinas dos trabalhadores da empresa. No final de cada *user story* está especificada uma tarefa de "testes". Na secção 3.5 poderá ser consultada informação mais detalhada acerca da forma como os testes foram realizados.

Assim sendo, esta é uma versão reduzida do *Product Backlog* que define apenas as funcionalidades que ficaram implementadas, a especificação completa do mesmo poderá ser consultada no documento em anexo de Especificação de Requisitos [4].

3.2.4 Casos de uso

Nesta secção pretendem-se representar visualmente as funcionalidades do sistema de forma mais clara e evidente, recorrendo a diagramas de casos de uso. Algumas das *user stories* presentes no *Product Backlog* da Tabela 1 são identificadas como casos de uso, dado o caráter funcional das mesmas, no entanto, uma vez que o *Product Backlog* da Tabela 1 representa apenas as *user stories* realizadas, existem casos de uso (como por exemplo os de administrador) que apenas poderão ser mapeados nas respetivas *user stories* consultando o *Product Backlog* completo, que se encontra no documento anexo de Especificação de Requisitos [4].

Os casos de uso que de facto foram implementados foram-no de forma agrupada, isto é, pegando no exemplo da *user story* #US07 do *Product Backlog*, nesta foi implementada a funcionalidade de publicação de atualizações por parte do distribuidor, assim como praticamente todas as outras subjacentes que se encontram representadas no respetivo diagrama de casos de uso da Figura 9.

O principal objetivo desta secção é portanto o de permitir obter uma perspetiva global da forma como cada ator interage com o sistema.

De seguida serão apresentados três diagramas distintos, correspondentes aos casos de uso nas vistas de distribuidor, administrador e serviço de atualizações, assim como uma breve referência à interação do utilizador final com o sistema, que é bastante reduzida.

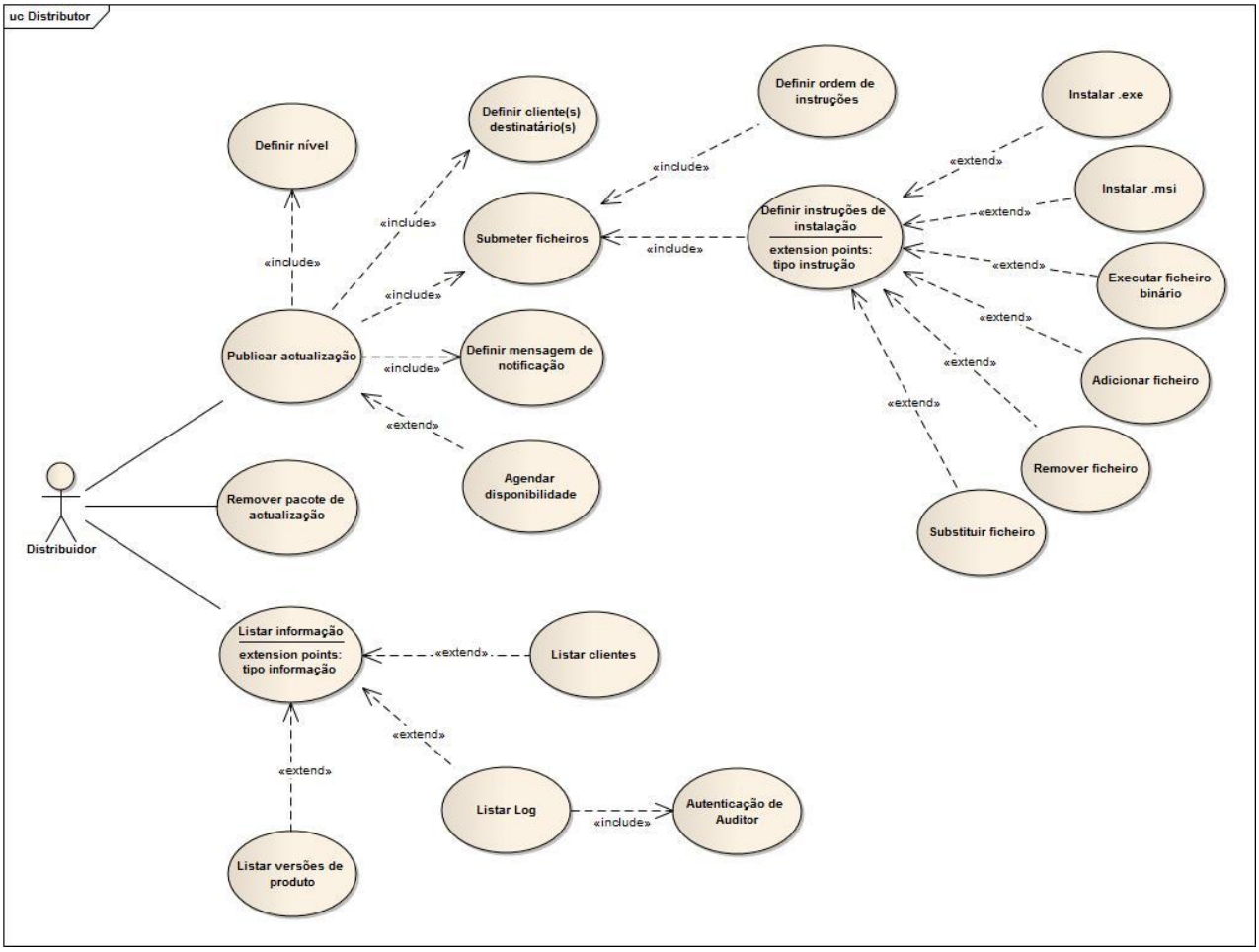


Figura 9 – Diagrama de casos de uso do ator "distribuidor"

A interação do distribuidor com o sistema é feita ao nível do servidor central, através das consolas de distribuição, onde o mesmo pode publicar novas atualizações, remover atualizações e listar informação como as versões ativas na plataforma e as versões a vigorar nos clientes.

Ao publicar uma atualização, o distribuidor apenas tem de submeter os ficheiros de instalação, definir os clientes destinatários e respetivas mensagens de notificação (*release notes*).

Uma vez publicada uma atualização, esta é automaticamente transferida e armazenada no servidor local do cliente, aguardando aprovação do administrador. No diagrama seguinte são identificados os casos de uso de administrador, onde para além da aprovação de atualizações estão também representadas outras funcionalidades.

Efetuando a credenciação de auditoria, é também possível consultar os registos de acesso ao servidor e de distribuição do produto.

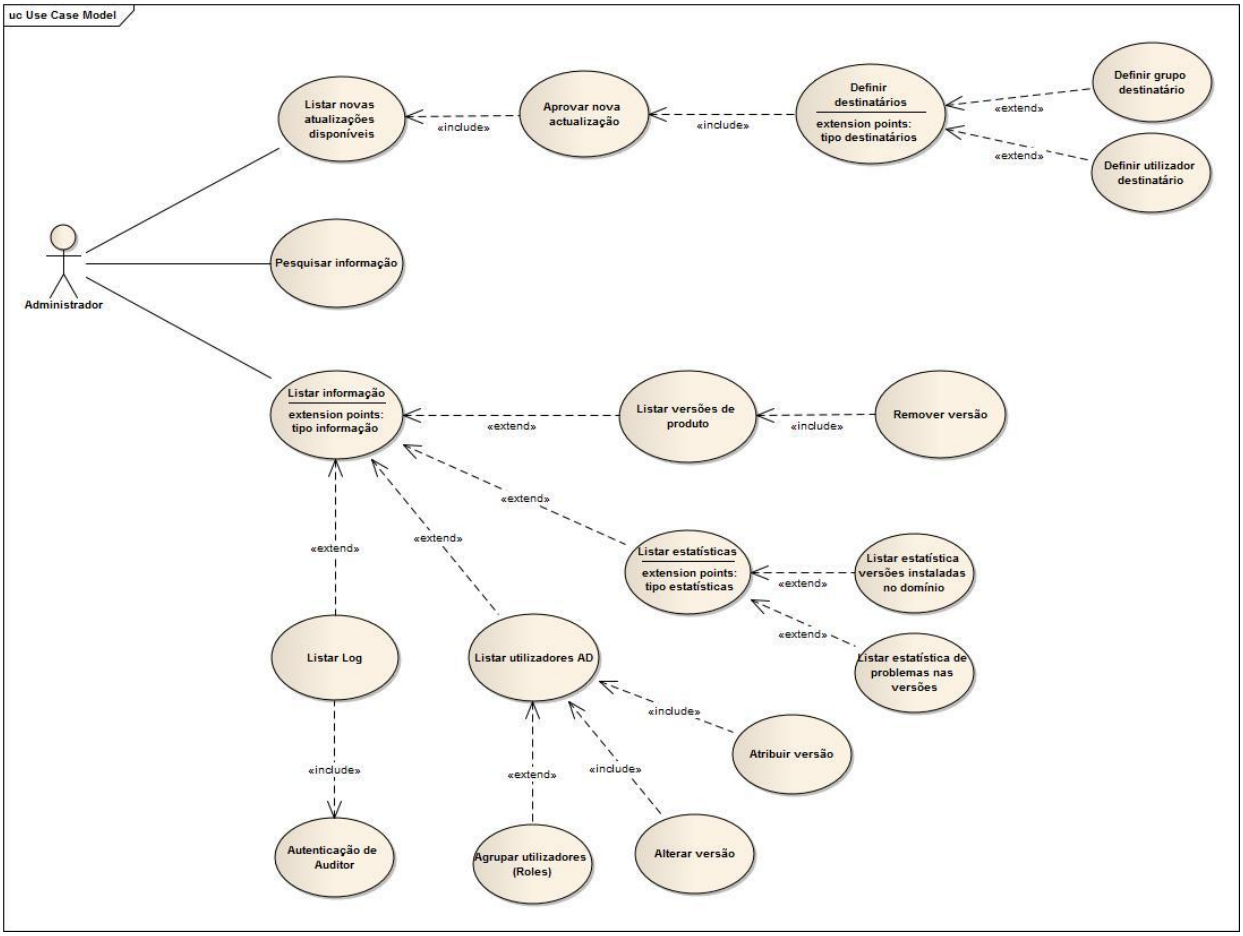


Figura 10 - Diagrama de casos de uso do ator "administrador"

O administrador interage com o sistema ao nível do servidor local, exercendo funções de gestão e controlo sobre as novas atualizações e a sua distribuição pelas máquinas dos utilizadores do domínio.

Uma vez transferida (de forma automática) uma nova atualização para o servidor local, o administrador é notificado sobre a existência da mesma, que se encontra em estado pendente, a aguardar aprovação. O administrador, que tem acesso às *release notes* (notas associadas à versão de produto) e nível de criticidade da versão de produto (ver secção 3.3.3), poderá aprovar a nova atualização quando assim o entender, bastando para isso definir os destinatários da mesma que poderá ser apenas um utilizador ou um grupo de utilizadores da *Active Directory*.

Para além de aprovar as novas atualizações através das consolas de administração o administrador poderá também pesquisar e listar informação como as estatísticas de atualizações bem e mal sucedidas e quais as versões de produto ativas no sistema. Através da integração com *Active Directory*, o administrador poderá também listar

todos os utilizadores do domínio, onde poderá alterar a versão de produto atribuída a cada utilizador ou grupo de utilizadores.

Apesar de não se tratar de um ator humano, o serviço de atualizações também pode ser visto como um elemento cujas funcionalidades podem ser representadas através de casos de uso, tal como está representado na Figura 11.

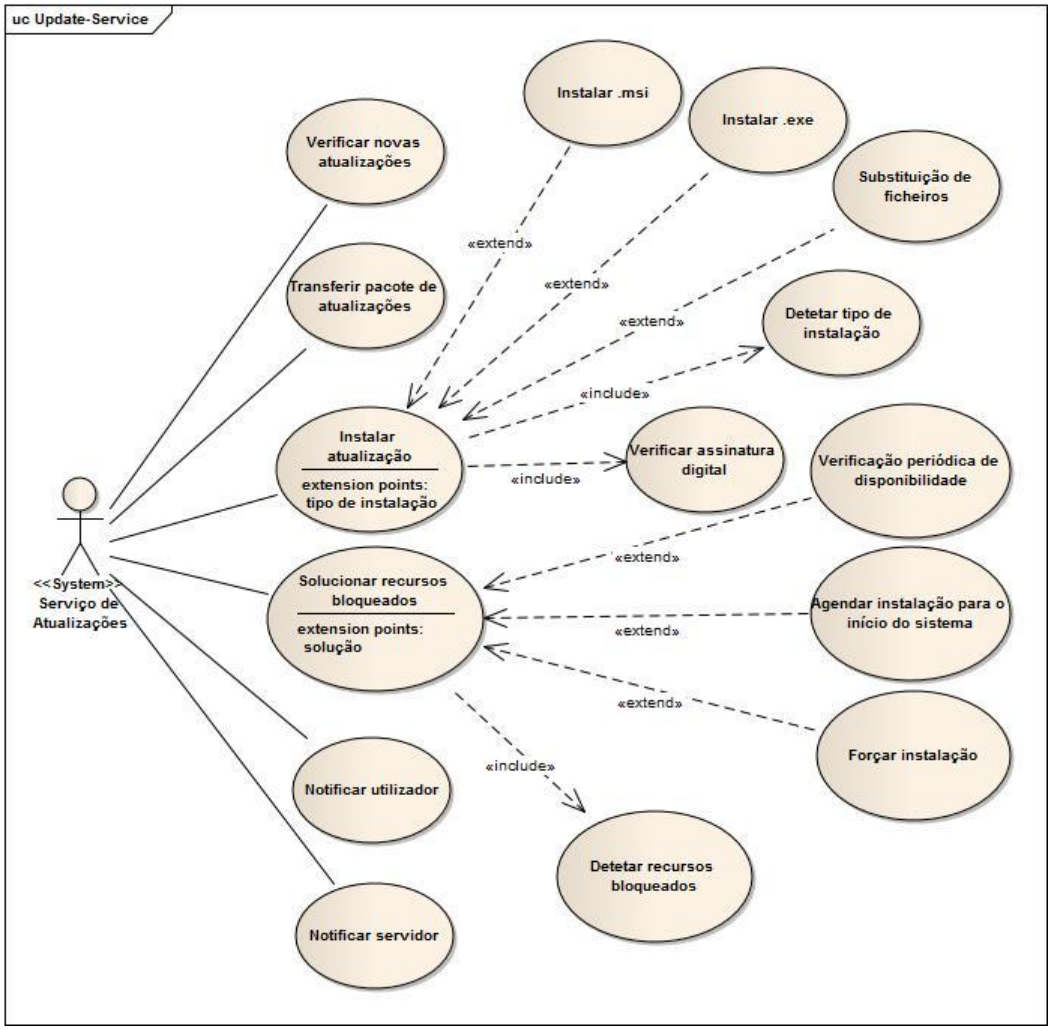


Figura 11 – Diagrama de casos de uso do ator "Serviço de Atualizações"

O serviço de atualizações apresenta como funcionalidades a verificação e transferência de pacotes de atualização, a instalação dos mesmos mediante a verificação da respetiva assinatura digital, a notificação de ambos utilizador e servidor e a deteção de recursos bloqueados acompanhada da respetiva solução. A solução para a existência de recursos bloqueados poderá ser uma de três: o mecanismo de verificação periódica (verificando se já os recursos já foram libertos), o agendamento da instalação para o início de sistema (se o limite de tentativas for excedido) ou o forçar da instalação,

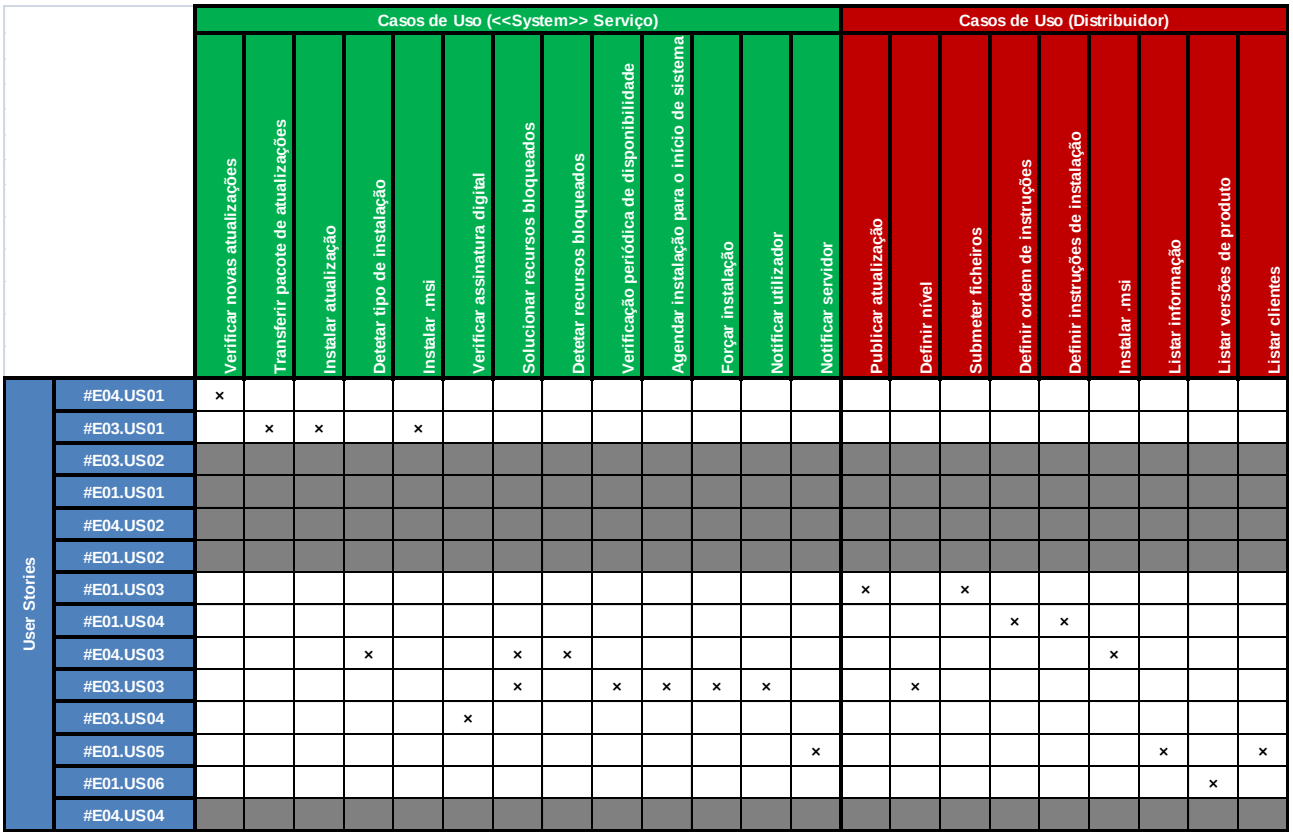
sendo que cada uma destas soluções depende do nível de criticidade do pacote de atualização (os níveis de criticidade são descritos em detalhe na secção 3.3.3).

Finalmente, a interação do utilizador final com o sistema é bastante reduzida, uma vez que o comportamento do serviço de atualizações é totalmente automático. Ainda assim, é possível que surja uma atualização crítica cuja instalação se encontra impedida de iniciar e o utilizador terá de confirmar a continuidade do processo, uma vez que esta situação envolve a interrupção de determinados processos em execução. Da mesma forma, no caso de uma atualização não-crítica ser agendada para o reiniciar de sistema, é dada a possibilidade ao utilizador de reiniciar no momento da notificação. À exceção destas duas situações em particular, o utilizador final não interage de outra forma com o sistema, pelo que não se justifica a representação dos seus casos de uso num diagrama.

De seguida é apresentada a ligação entre as *user stories* implementadas e respetivos casos de uso através de uma matriz de rastreabilidade, para melhor clarificar quais foram exatamente as funcionalidades implementadas em cada uma delas.

3.2.5 Matriz de rastreabilidade

Os casos de uso implementados nas *user stories* realizadas durante a fase de implementação refletem funcionalidades dos atores serviço de atualizações e distribuidor. Na Figura 12 é então apresentada a rastreabilidade entre as *user stories* implementadas (ver Tabela 1) e os respetivos casos de uso. A matriz de rastreabilidade completa (de todos os casos de uso e *user stories* do *Product Backlog*) poderá ser consultada no documento anexo de Especificação de Requisitos [4].



		Casos de Uso (<<System>> Serviço)													Casos de Uso (Distribuidor)								
		Verificar novas atualizações	Transferir pacote de atualizações	Instalar atualização	Detetar tipo de instalação	Instalar .msi	Verificar assinatura digital	Solucionar recursos bloqueados	Detetar recursos bloqueados	Verificação periódica de disponibilidade	Agendar instalação para o início de sistema	Forçar instalação	Notificar utilizador	Notificar servidor	Publicar atualização	Definir nível	Submeter ficheiros	Definir ordem de instruções	Definir instruções de instalação	Instalar .msi	Listar informação	Listar versões de produto	Listar clientes
User Stories	#E04.US01	×																					
	#E03.US01		×	×		×																	
	#E03.US02																						
	#E01.US01																						
	#E04.US02																						
	#E01.US02																						
	#E01.US03														×		×						
	#E01.US04																	×	×				
	#E04.US03				×			×	×											×			
	#E03.US03							×		×	×	×	×			×							
	#E03.US04						×																
	#E01.US05													×							×		×
	#E01.US06																					×	
	#E04.US04																						

Figura 12 - Matriz de rastreabilidade

Como podemos ver na Figura 12, existem algumas *user stories* marcadas a cinzento. Esta marca serve para indicar aquelas *user stories* que não representam funcionalidades concretas, representáveis por casos de uso. Analisando o resto da matriz, podem ser identificadas várias *user stories* onde foram implementados múltiplos casos de uso, isto é, múltiplas funcionalidades. Tal facto deveu-se ao acelerar da fase de implementação, que desta forma prosseguiu com a implementação de várias funcionalidades de forma inesperada, quando por vezes existiam *user stories* próprias para o efeito, que posteriormente acabaram por ser excluídas do *Product Backlog* por terem sido concluídas. Como consequência deste facto surgiram então *user stories* que demoraram mais do que uma *sprint* a executar, como poderá ser consultado em maior detalhe no documento anexo de *Sprint Backlogs* [2].

3.2.6 Requisitos não funcionais

Nesta secção serão apresentados os requisitos não funcionais do sistema que, precisamente devido à sua natureza não funcional, não são definidos, pelo menos diretamente, através de *user stories*. No entanto, alguns dos requisitos não funcionais que de seguida são apresentados, nomeadamente ao nível da segurança, encontram-se ligados a *user stories* do *Product Backlog* que de forma indireta assumem a sua implementação.

Código	Âmbito	Requisito não funcional
#RNF01	[Escalabilidade]	O servidor deve limitar o número de pedidos de clientes em situações de sobrecarga
#RNF02	[Desempenho]	O servidor central deve permitir a submissão de pacotes de atualização de tamanho menor ou igual a 50MB
#RNF03	[Segurança]	Todos os acessos e operações no servidor devem ser registados
#RNF04	[Segurança]	Os acessos ao servidor devem ser controlados
#RNF05	[Segurança]	A comunicação cliente-servidor deve manter-se confidencial em todas as fases do processo de atualizações
#RNF06	[Segurança]	A integridade e autenticidade dos pacotes de atualização devem ser asseguradas
#RNF07	[Segurança]	A plataforma deve impedir o não-repúdio de informação
#RNF08	[Segurança]	A plataforma deve poder ser auditada
#RNF09	[Robustez]	O serviço de atualizações deve ter a capacidade de recuperar as quebras de comunicação com a aplicação que comunica com o utilizador
#RNF10	[Transparência]	O processo de verificação e instalação deve ser totalmente transparente
#RNF11	[Compatibilidade]	O serviço de atualizações deve suportar instaladores MSI
#RNF12	[Compatibilidade]	A plataforma deve ser compatível com ambientes Windows (XP ou superior)
#RNF13	[Compatibilidade]	A plataforma deve ser compatível com ambos os sistemas x86 e x64
#RNF14	[Compatibilidade]	A plataforma deve ser integrável com sistemas Active Directory
#RNF15	[Compatibilidade]	A plataforma deve ser capaz de armazenar dados em bases de dados Microsoft SQL Server
#RNF16	[Compatibilidade]	A plataforma deve ser integrável com servidores Web Microsoft Internet Information Services (6.0 ou superior)
#RNF17	[Confiabilidade]	O processo de atualização não deverá permitir instalações corrompidas
#RNF18	[Interface]	As consolas de distribuição e administração devem seguir os padrões definidos pelo designer gráfico da Critical Software
#RNF19	[Interface]	As consolas de distribuição e administração devem apresentar uma baixa curva de aprendizagem

Tabela 2 - Requisitos não funcionais

Como pode ser visto na Tabela 2, o *csSECURE – Software Update Service* apresenta requisitos não funcionais ao nível da escalabilidade, desempenho, segurança, robustez, transparência, compatibilidade e da sua interface gráfica. Estes requisitos, cuja definição e implementação pode ser consultada em maior detalhe no documento anexo de Especificação de Requisitos [4], são considerados essenciais para a viabilidade do projeto, conferindo ao mesmo um elevado grau de competitividade no mercado.

3.3 Arquitetura do Sistema

3.3.1 Introdução

De forma a garantir que todas as *user stories* e requisitos são abrangidos pela solução a desenvolver e para orientar a respetiva fase de implementação de forma organizada, foi especificada a sua arquitetura e desenho detalhado.

Nesta secção são apresentadas diversas perspetivas da arquitetura do *csSECURE – Software Update Service* e é especificada a arquitetura de pré-produção, que ilustra como a plataforma está implementada neste momento na Critical Software. De seguida, são apresentadas as tecnologias escolhidas para a implementação desta plataforma de atualizações, bem como as razões que motivaram a escolha das mesmas.

3.3.2 Arquitetura de alto nível

A vista física da arquitetura de alto nível de todo o sistema está representada na Figura 8, presente na secção 3.2.2.

Nas secções seguintes são apresentadas algumas vistas da arquitetura do *csSECURE – Software Update Service*, para melhor ilustrar o funcionamento do mesmo sob diferentes perspetivas.

3.3.3 Outras vistas de arquitetura

Sendo este um sistema que funciona no modelo cliente-servidor, uma das vistas mais relevantes é a vista de componentes cliente-servidor da arquitetura, pelo que esta será a primeira a ser representada, através da Figura 13. Por servidor entenda-se o servidor central, da Critical Software, onde são publicadas as atualizações. Por cliente entenda-se os clientes do csSECURE que adquirem tanto o produto como a plataforma de atualizações.

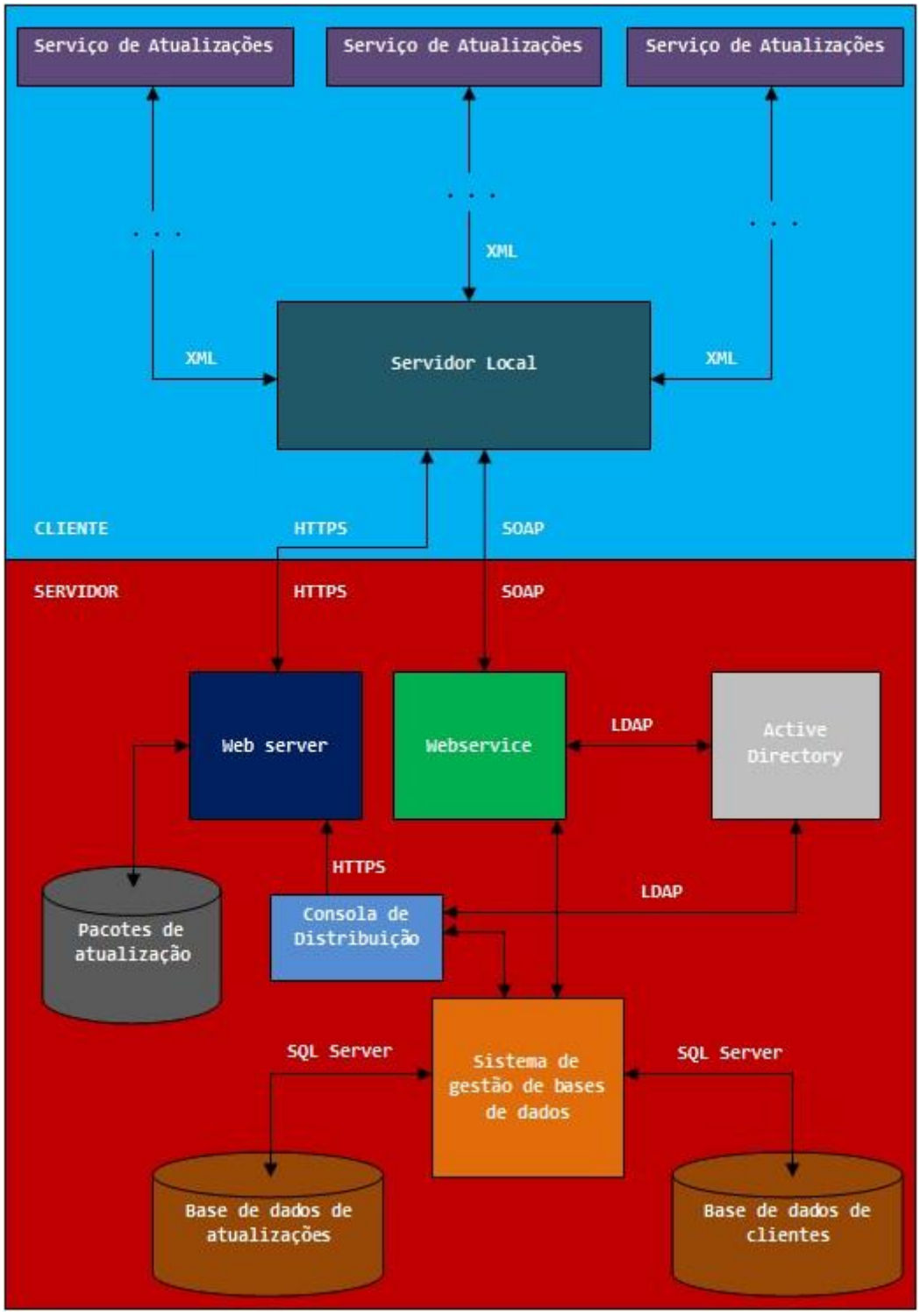


Figura 13 - Vista de componentes da arquitetura (cliente-servidor)

Como indicado, na Figura 13 está representada a vista de componentes cliente-servidor da arquitetura do *csSECURE – Software Update Service*. A região inferior, denominada "Servidor" representa o servidor central, onde a equipa do csSECURE publicará as atualizações para os seus clientes fora do domínio da Critical Software.

As novas atualizações são portanto publicadas através da consola de distribuição e são armazenadas no servidor Web, sendo que toda a informação associada a esse pacote de atualizações é também registada na base de dados. A partir desse momento, os clientes poderão descarregar a nova versão do csSECURE para o seu domínio (no servidor local), para posteriormente ser instalada nas máquinas dos utilizadores.

O mecanismo de autenticação dos clientes do csECURE no servidor é conseguido através de validação de credenciais criadas para o efeito na base de dados de clientes do servidor. Essas credenciais são fornecidas ao cliente num processo que pode ser visto como um *enrollment* ou registo de clientes externos ao domínio do servidor central.

Esses mesmos clientes são representados numa perspetiva de alto nível na região superior do diagrama, denominada de “Cliente”. Internamente, estes clientes também funcionam no modelo cliente-servidor, onde os serviços de atualizações instalados nas máquinas dos utilizadores assumem o papel de cliente e o servidor local assume o papel de servidor. A vista interna do *csSECURE – Software Update Service* instalado num cliente do csSECURE é apresentada de forma mais detalhada na Figura 14.

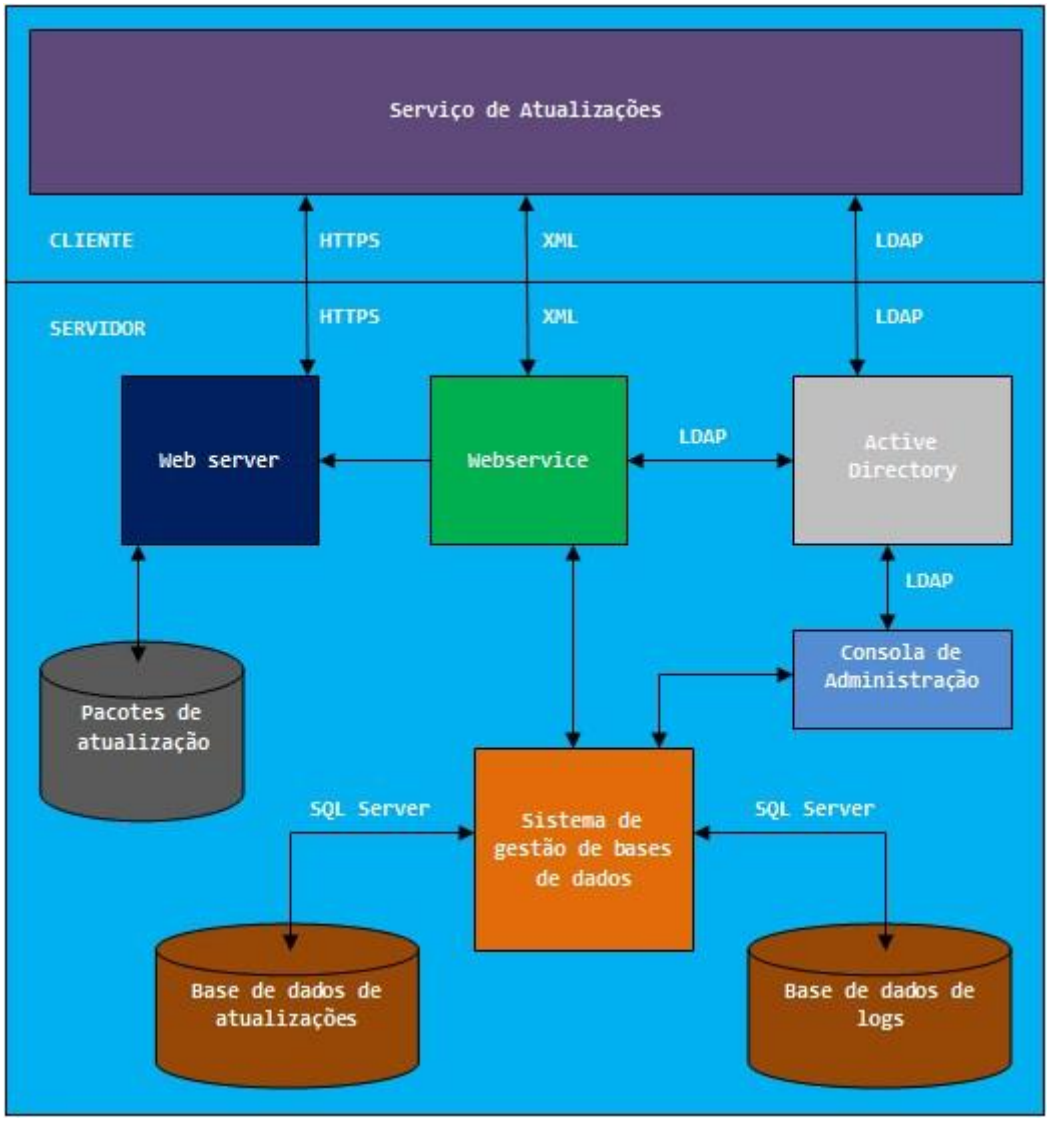


Figura 14 - Vista interna de componentes do csSECURE - Software Update Service instalado num cliente do csSECURE (cliente-servidor)

Em cada cliente onde o *csSECURE – Software Update Service* estiver instalado, existirá uma estrutura composta por Webservice, servidor Web HTTP e Sistema de Gestão de Bases de Dados, a estrutura base necessária para manter o csSECURE atualizado. Em cada empresa ou organização cliente, o Webservice do servidor local estará em contacto com o Webservice do servidor central (tal como na Figura 13) para de forma periódica verificar a existência de novas atualizações e, em caso necessário, fazer a sua transferência, registando a informação necessária na base de dados e o pacote de atualização no servidor Web. Uma vez armazenado, o novo pacote de atualização apenas ficará disponível para os serviços de atualização das máquinas dos utilizadores quando o administrador de sistema o aprovar, através da consola de administração.

Relativamente ao serviço de atualizações, este acede periodicamente ao Webservice do servidor local do domínio (com autenticação integrada) verificando a existência de novas atualizações aprovadas pelo administrador. A vista conceptual da sua arquitetura encontra-se representada na Figura 15, onde se podem observar os dois componentes internos que o compõem, um serviço e uma aplicação.

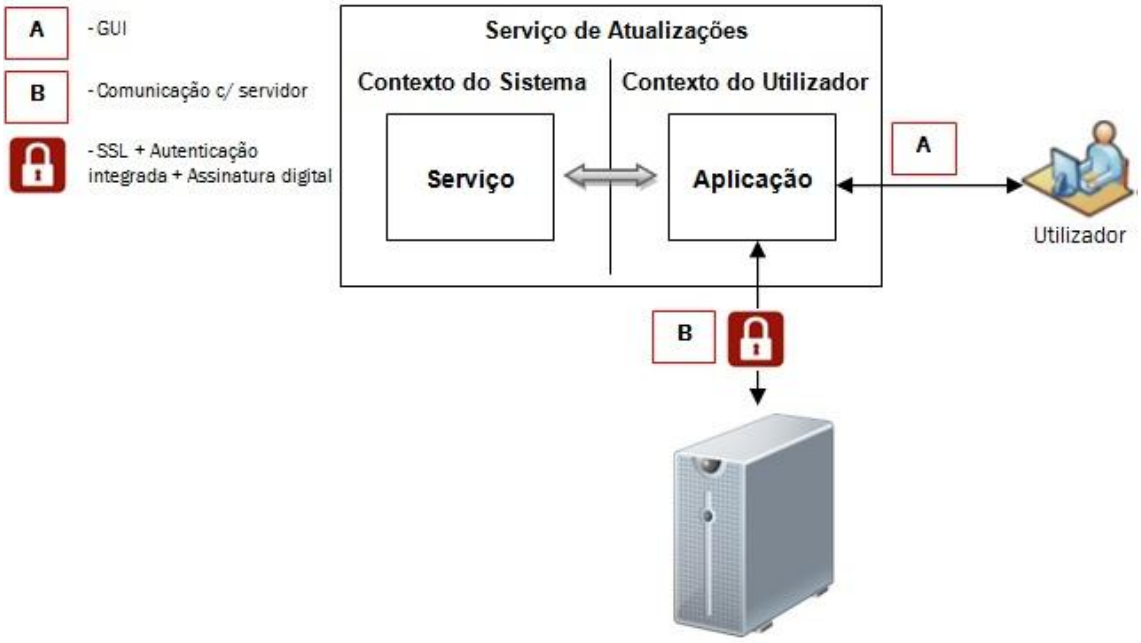


Figura 15 – Vista conceptual da arquitetura do serviço de atualizações

Qualquer notificação seja ela de instalação agendada para o reiniciar de sistema ou simplesmente de atualização bem-sucedida é mostrada ao utilizador através da aplicação - esta encontra-se a ser executada no contexto de segurança do utilizador e, portanto, com os mesmos privilégios que este possui sobre a máquina. É também a aplicação que contacta diretamente o servidor para as verificações e transferências dos pacotes de atualização, pelo simples facto de permitir utilizar as credenciais do utilizador para efeitos de autenticação. Ao mesmo tempo o serviço, que se encontra a ser executado no contexto de segurança do sistema e, portanto, com privilégios de administrador, trata de todas as operações relacionadas com a instalação de novas versões de produto.

Importa aqui salientar o impacto que o mecanismo de níveis de criticidade tem sobre o serviço de atualizações. Caso não exista qualquer impedimento à instalação de uma nova atualização, esta é executada de forma automática e transparente, notificando apenas o utilizador após a sua conclusão. No entanto, se existirem recursos necessários à instalação bloqueados, é necessário que o serviço de atualizações lide adequadamente com a situação. Para tal, existem dois níveis de atualização, o nível

"Crítico" e o nível "Não-Crítico". De seguida é detalhado o comportamento do serviço de atualizações numa situação de recursos necessários à instalação bloqueados, de acordo com o nível de criticidade de um pacote de atualização.

- **Nível Crítico:** Este nível existe precisamente para as atualizações "críticas", isto é, extremamente urgentes como sejam, por exemplo, correções de falhas de segurança. Desta forma, caso existam recursos bloqueados a impedir a instalação, o utilizador é informado de que uma instalação crítica está prestes a ser instalada, tendo este apenas que guardar o seu trabalho e clicar num botão. Automaticamente os processos que estão a trancar os recursos necessários são terminados, a instalação é feita e os mesmos processos que foram terminados pelo serviço de atualizações são novamente iniciados, por uma questão de conforto para o utilizador, que não terá assim de voltar a iniciar todas as aplicações terminadas. Visto esta ser uma atualização crítica, a opção de o utilizador fechar a notificação de instalação encontra-se desativada.
- **Nível Não-Crítico:** O nível "Não-Crítico" será talvez o mais comum, pois nem todas as atualizações são urgentes. Neste caso, se existirem recursos bloqueados, o serviço de atualizações entra em modo de espera e periodicamente vai tentando de novo. Caso os recursos sejam desbloqueados durante uma destas tentativas, a instalação prossegue normalmente. Caso os recursos não sejam desbloqueados ao fim de um número limitado de tentativas, a instalação é agendada para o próximo arranque do sistema, sendo o utilizador notificado deste facto. Este tipo de instalação é conseguido recorrendo ao *Task Manager* - uma ferramenta nativa do Windows que permite programar tarefas. Numa situação de instalação no arranque do sistema, para garantir que esta é executada antes de qualquer outra tarefa, a prioridade da tarefa de instalação em questão é alterada para que esta se sobreponha a todas as outras tarefas que possam vir a consumir recursos necessários à tarefa de instalação. Desta forma, a instalação poderá ser feita e o utilizador será notificado assim que o sistema operativo iniciar.

De seguida é apresentada uma vista funcional da arquitetura do serviço de atualizações, para melhor esquematizar as principais etapas do funcionamento interno do mesmo.

Figura 16 - Vista funcional da arquitetura do serviço de atualizações

Como se pode ver o serviço de atualizações contacta periodicamente o Webservice do servidor local, presente na estrutura do *csSECURE – Software Update Service* do cliente, que lhe envia as informações necessárias para determinar se existe uma nova versão de produto. A resposta do Webservice inclui a seguinte informação:

- **Identificador de produto:** Código único que identifica o produto a atualizar de forma unívoca perante serviço de atualizações e plataforma.
- **Product code:** Código único que identifica uma determinada versão de produto de forma unívoca.
- **Upgrade code:** Código que identifica todas as versões de um determinado produto.
- **Versão:** Versão de produto disponível na plataforma

- **Nível de criticidade:** Nível de criticidade do pacote de atualização que pode tomar os valores “Crítico” ou “Não-Crítico”.
- **URL de transferência:** Caminho que o serviço de atualizações deverá utilizar para descarregar o pacote de transferências, caso seja necessário.

Uma vez recebida a resposta do servidor, o serviço de atualizações verifica se o produto se encontra instalado e, em caso afirmativo, verifica também qual a versão de instalação do mesmo. Estas verificações são feitas com base no *upgrade code*, um código que identifica todas as versões de um determinado produto, isto é, todas as novas versões do csSECURE (apesar de cada uma ter o seu *product code* distinto) têm o mesmo *upgrade code*, permitindo assim identificar o produto de forma inequívoca.

Verificando a versão instalada e comparando com a versão disponível no servidor, o serviço de atualizações sabe, a partir desse momento, se necessita ou não de descarregar o pacote de atualizações. Caso exista de facto uma nova versão, então o pacote de atualizações é descarregado e a sua assinatura digital verificada. De seguida, caso todos os passos até aqui tenham sido concluídos com sucesso, a base de dados interna do instalador é pesquisada para determinar os recursos necessários à instalação e qual a diretoria de instalação do produto. Sabendo a diretoria de instalação e quais os recursos necessários, é feita uma verificação da disponibilidade dos últimos, para determinar se a instalação pode ser iniciada, ou se, por outro lado, existe algum impedimento por parte de algum outro processo.

Caso não exista nenhum impedimento, a instalação é iniciada e concluída de forma automática e transparente. Caso existam recursos necessários bloqueados, então é avaliado o nível de criticidade, para determinar o caminho a seguir.

Na secção 3.4 poderá ser consultada informação mais detalhada sobre a forma como são identificados os recursos necessários e a sua disponibilidade, assim como a diretoria de instalação.

3.3.4 Desenho detalhado

Como estrutura de suporte de dados do *csSECURE – Software Update Service* existem três bases de dados distintas, uma base de dados de atualizações, uma de clientes e uma de instalações. Estas bases de dados são responsáveis por armazenar informações associadas aos pacotes de atualização, informações associadas aos clientes para efeitos de autenticação e informações associadas às instalações realizadas pelos serviços de autenticação instalados nas máquinas dos utilizadores, respetivamente.

A base de dados de atualizações está presente na estrutura de ambos cliente e servidor, a base de dados de clientes na estrutura do servidor e a base de dados de instalações está presente na estrutura do cliente (esta separação das bases de dados encontra-se representada na Figura 13).

De seguida é apresentado o diagrama físico da base de dados de atualizações, visto ser esta a base de dados de maior relevância em todo o sistema.

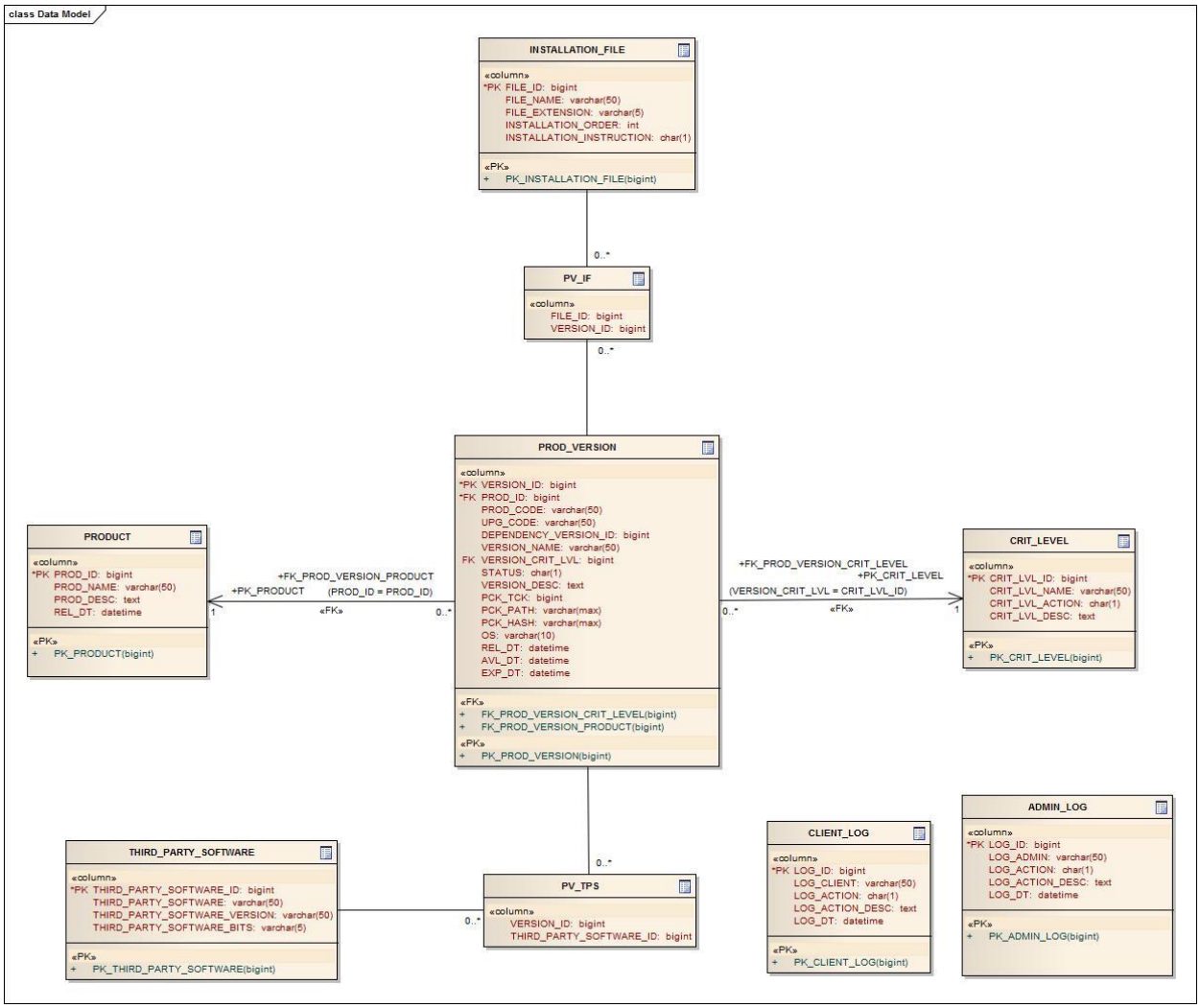


Figura 17 - Diagrama físico da base de dados de atualizações

Esta base de dados armazena toda a informação associada ao produto, versões de produto e respetivos ficheiros de instalação, local de transferência, nível de criticidade, entre outros. Com este modelo de base de dados está aberta a hipótese de estender o funcionamento desta plataforma para atualizações de outros produtos que não o csSECURE.

Todo o *software* desenvolvido, em ambos cliente e servidor, apresenta uma estrutura em camadas, conforme está representado na Figura 18.

Figura 18 - Estrutura em camadas do software desenvolvido

Esta é uma prática comum no desenvolvimento de *software* que tem como principal objetivo impor uma estrutura modular e independente [51], onde são separadas as camadas *Business Logic Layer* (BLL), *Business Objects Layer* (BO) e *Data Access Layer* (DAL). De seguida são descritas em maior detalhe cada uma destas camadas:

- **BLL:** ou *Business Logic Layer* é a camada responsável por manter toda a lógica da aplicação, isto é, separa algoritmos e operações de alto nível das camadas de mais baixo nível. Tipicamente, a camada BLL faz a ligação entre a camada de dados e a interface gráfica.
- **BO:** ou *Business Objects* é a camada responsável por estruturar a informação vinda da camada de dados em classes e objetos representativos.
- **DAL:** ou *Data Access Layer* é a camada de dados, responsável por todos os acessos e operações ao nível das bases de dados.

Dando um exemplo do funcionamento de uma estrutura deste tipo temos a situação em que a camada DAL acede a uma base de dados e retira informação, representada através de objetos da camada BO, para que estes sejam utilizados na camada BLL. Da mesma forma, a camada BLL pode também fornecer dados à camada DAL, sendo que estes são representados através da camada BO. No fundo a camada BO permite que a informação circule entre as camadas DAL e BLL de forma coerente.

3.3.5 Arquitetura de pré-produção

Sendo este um projeto de estágio, o período de implementação foi algo limitado, pelo que não foram implementados todos os detalhes, funcionalidades e mecanismos especificados inicialmente, mas sim aqueles que permitiam no prazo disponível implementar uma solução funcional pronta a responder rapidamente às necessidades do csSECURE.

Assim sendo, foi implementada uma versão ligeiramente modificada da arquitetura (a arquitetura de pré-produção) onde a distribuição do produto é feita apenas para o interior da organização. Neste cenário, são publicadas atualizações do csSECURE no servidor através das consolas de distribuição para que os serviços de atualização instalados em cada máquina possam periodicamente verificar a existência de novas atualizações e, caso necessário, descarregar e instalar as mesmas.

Focando-nos na vista física da arquitetura de alto-nível que ficou implementada, podemos então identificar na mesma os seguintes módulos funcionais:

- **Servidor de atualizações** (onde as atualizações são publicadas)
- **Sistema de gestão de bases de dados** (estrutura de suporte de dados)
- **Serviço de atualizações** (presente nas *workstations* de cada utilizador)

Na figura seguinte está representada a vista física da arquitetura de alto-nível do sistema implementado.

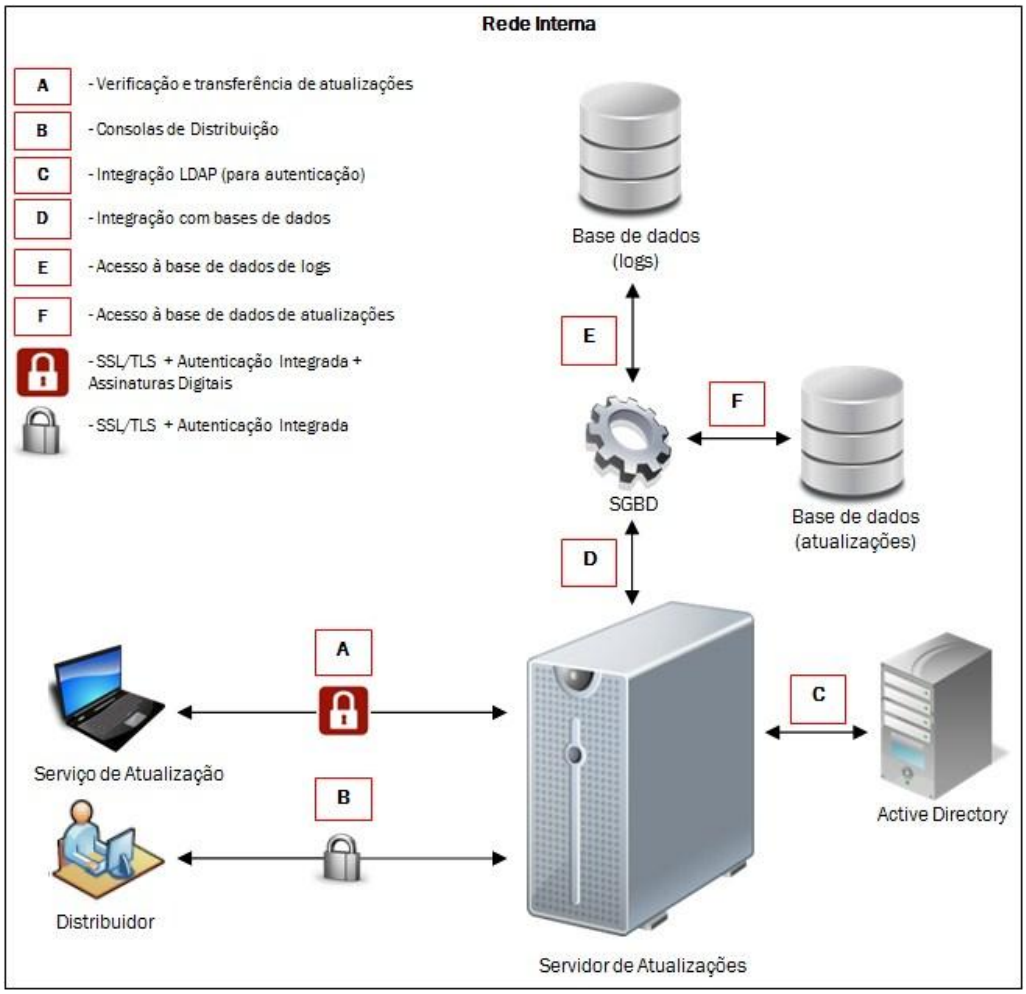


Figura 19 – Vista física da arquitetura de alto nível (pré-produção)

Como pode ser visto na Figura 19, o distribuidor pode publicar atualizações no servidor através da consola de distribuição, bem como aceder a informação relativa ao estado de instalação do produto nas máquinas dos utilizadores, que é disponibilizada pelo módulo sistema de gestão de bases de dados (SGBD). O serviço de atualizações (presente nas máquinas ou estações de trabalho dos utilizadores), por sua vez, acede periodicamente ao servidor para verificar a existência de novas atualizações. Este acesso é controlado por autenticação integrada e protegido por SSL (*Secure Socket Layer*, ver secção 3.6). Para além da integração com o módulo SGBD, está ainda representada na arquitetura a integração LDAP (*Lightweight Directory Access Protocol*) com *Active Directory*. Esta integração existe para que a autenticação de distribuidor e administrador se faça de forma automática sem ser necessário sobrecarregar o servidor com essa responsabilidade (o que acabaria por se tornar redundante, uma vez que com *Active Directory*, um elemento essencial da estrutura base do csSECURE, a autenticação é feita de forma bastante simples).

Abstraindo-nos agora da vista física de alto-nível da arquitetura e focando-nos na vista funcional da mesma, podemos observar em maior detalhe a forma como serviço de atualizações e distribuidor interagem com os componentes que compõem o servidor. De seguida, na Figura 20, está representado o funcionamento da plataforma numa perspetiva de alto nível.

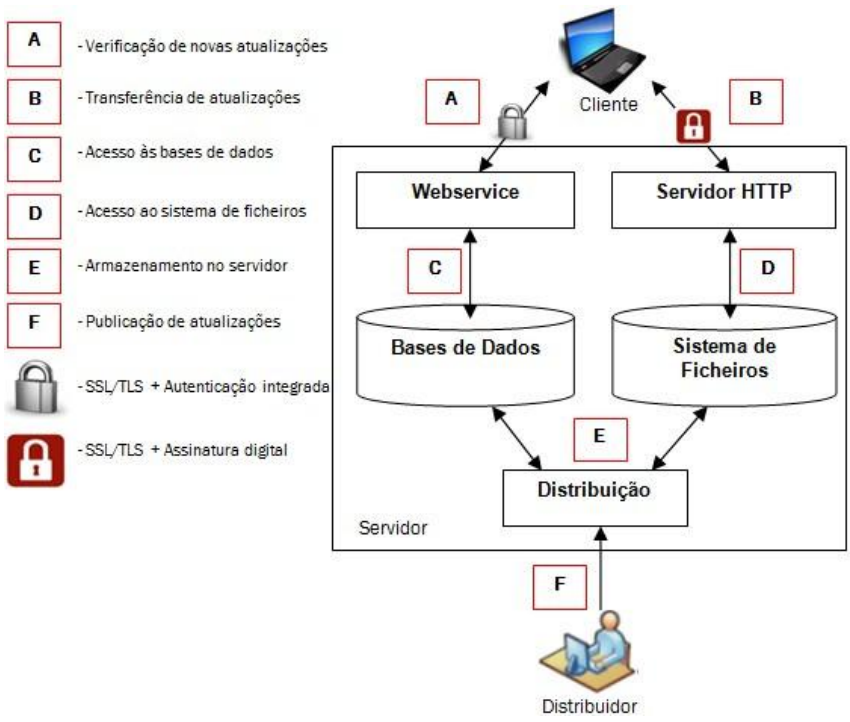


Figura 20 – Vista funcional da arquitetura de alto nível (pré-produção)

Numa breve descrição do funcionamento do sistema tal como está representado na Figura 20, o distribuidor, através das consolas de distribuição (via Web Browser), publica novas atualizações no servidor onde os dados são armazenados de forma automática no momento da submissão, tanto na base de dados (informação associada às versões de produto) como no sistema de ficheiros do servidor (pacotes de atualização). Posteriormente, os clientes contactarão o Webservice (que ligado à base de dados fornece a informação necessária ao serviço de atualizações) para verificar se existem novas atualizações e o servidor Web HTTP (que tem acesso ao sistema de ficheiros onde estão os pacotes de atualização) para transferir as mesmas.

O serviço de atualizações foi implementado exatamente como foi especificado, sem sofrer qualquer alteração. Assim sendo, a sua vista da arquitetura encontra-se já descrita na secção 3.3.3 e não será portanto aqui representada novamente.

3.3.6 Tecnologias

O *csSECURE – Software Update Service* tira proveito das tecnologias utilizadas pelo csSECURE, tendo como principal objetivo não obrigar o cliente a fazer qualquer tipo de instalação adicional. O csSECURE é um produto de determinadas características, está instalado no cliente, portanto será atualizado através das tecnologias já utilizadas.

Este projeto é então suportado pelas seguintes tecnologias:

- **Microsoft Windows:** O sistema operativo utilizado para instalar a plataforma de atualizações (Windows XP, Windows Vista, Windows 7). A escolha do sistema operativo base foi feita de acordo com os requisitos do csSECURE e respetivas tecnologias utilizadas, fortemente ligadas a sistemas Microsoft.
- **Microsoft Active Directory:** O serviço base que permite gerir e controlar utilizadores e grupos de utilizadores de um domínio, assim como implementar facilmente autenticação integrada em ambientes Microsoft Windows, evitando assim sobrecarregar os servidores com serviços de autenticação e respetiva estrutura de suporte de dados. Esta tecnologia é também bastante utilizada no produto csSECURE, facilitando assim a sua integração no *csSECURE – Software Update Service*.
- **Microsoft SQL Server:** Tecnologia de armazenamento de dados nos servidores, bastante comum em sistemas de informação onde predominam tecnologias Microsoft, os mesmos onde o csSECURE está instalado. Assim sendo e uma vez que este é o motor base do csSECURE, a sua utilização não representa nenhum custo adicional para o cliente que pretender instalar o *csSECURE – Software Update Service*.

- **Visual Studio .NET (C#):** Tecnologia de desenvolvimento do serviço de atualizações e Webservice. Uma vez definidas todas as tecnologias de suporte deste projeto, esta foi a escolha mais razoável devido à compatibilidade e vasto suporte existentes. Após a análise feita no estudo do estado da arte, onde foram identificadas diversas soluções deste sector de mercado desenvolvidas nesta tecnologia, verificou-se uma grande tendência para a escolha da mesma devido à maturidade evidenciada e facilidade de integração. Acresce também o facto de esta ser uma tecnologia bastante utilizada no desenvolvimento do csSECURE, pelo que a familiaridade da mesma ajudou a reforçar a escolha inicial.
- **ASP .NET & JavaScript (Ext JS):** Tecnologia de desenvolvimento das consolas de distribuição e de administração. As consolas de administração e monitorização do csSECURE foram desenvolvidas da mesma forma, pelo que a adoção da mesma tecnologia de desenvolvimento facilita uma possível integração das consolas do *csSECURE - Software Update Service* nas consolas já existentes do csSECURE, evitando assim sobrecarregar o cliente com diferentes consolas de gestão distintas.
- **XML:** Tecnologia eleita para a troca de mensagens entre cliente e servidor, assim como para a definição dos ficheiros de configuração dos pacotes de atualização. Esta tecnologia foi escolhida devido à sua maturidade e universalidade, que lhe conferem um estatuto de enorme utilidade em sistemas desta natureza.

3.4 Implementação e Desafios Superados

Uma vez desenhada em detalhe a arquitetura da solução a desenvolver, seguiu-se a fase de implementação. Durante esta fase, para que a plataforma de atualizações apresentasse o comportamento transparente e automático que era pretendido, surgiram alguns desafios de cariz técnico bastante aliciantes, aos quais foi necessário dar resposta.

De seguida é apresentado um pequeno resumo dos principais desafios encontrados e de como estes foram superados.

- **Permissões de administração:** Esta plataforma permite instalar atualizações mesmo em máquinas onde o utilizador não possui permissões de administração. Como o csSECURE tem como clientes maioritariamente organizações cuja política de segurança não permite ceder privilégios de administração ao utilizador comum este é um requisito essencial. Caso contrário, não seria possível disseminar e instalar o produto de forma automática, sem a intervenção de um administrador. A ideia inicial para o

desenvolvimento da presente solução era a implementação de um simples serviço de atualizações automáticas, uma vez que os serviços podem ser executados no contexto de segurança do sistema com elevação de privilégios e, portanto, consegue instalar atualizações que o normal utilizador sem privilégios não consegue. Acontece que, por uma questão de segurança, os serviços Windows não devem comunicar diretamente com o utilizador [21] e já desde o sistema operativo Windows Vista que tal não é sequer permitido. Contrariamente a um serviço Windows, uma aplicação tem permissões para comunicar com o utilizador (por exemplo, para efeitos de notificação) e até permite usar as credenciais do mesmo para efeitos de autenticação. No entanto, uma aplicação é executada no contexto de segurança do utilizador e, portanto, apenas com os privilégios que este possui. Assim sendo, temos dois componentes distintos, cada um com as suas vantagens e desvantagens, mas nenhum por si só permite resolver o problema. Através da análise das conclusões retiradas do estudo de estado da arte (ver secção 3.1.5), surgiu a ideia de juntar os dois que assim, apesar de serem executados em contextos de segurança diferentes (o serviço no contexto de segurança do sistema e a aplicação no contexto de segurança do utilizador) comunicam entre si, permitindo instalar atualizações sem problemas de permissões (os pedidos de elevação de privilégios que tipicamente surgem nas instalações) e interagir com o utilizador quando necessário. A comunicação entre os componentes é feita por *named pipe* e o conjunto dos dois compõe o serviço de atualizações da plataforma, possibilitando assim um comportamento transparente e automático na instalação de novas atualizações.

- **Níveis de criticidade:** Um ponto inovador desta plataforma que não existe em nenhuma das soluções identificadas no mercado é a utilização de níveis de criticidade, que permitem distinguir as atualizações no que toca à urgência da sua instalação. Com o *csSECURE – Software Update Service*, qualquer instalação, caso não exista qualquer dos impedimentos anteriormente identificados, é sempre instalada de forma automática e silenciosa. No entanto, o bloqueio de recursos necessários às atualizações é um cenário bastante comum no dia-a-dia da utilização de qualquer *workstation* e, como tal, necessita de um mecanismo de resposta, adaptado às necessidades de cada pacote de atualização. No *csSECURE – Software Update Service*, para reduzir o nível de intrusão do processo de instalação de uma atualização “Não-Crítica” caso existam recursos bloqueados (situação que tipicamente necessita da intervenção do utilizador), o serviço de atualizações aguarda silenciosamente até que os mesmos estejam de novo disponíveis. Após esta espera silenciosa

e atingido o limite de tentativas, caso não seja possível iniciar a instalação, esta é então agendada para o arranque do sistema e o utilizador é notificado deste facto. Este mecanismo de espera serve qualquer tipo de atualização considerada como “normal” ou “não urgente”, mas e se existirem atualizações críticas que, por exemplo, pretendem colmatar falhas de segurança e, como tal, necessitam de ser instaladas assim que possível? Será que este tipo de atualizações poderá ser tratado de igual forma no cliente caso existam recursos bloqueados? É nesta situação que é revelada a verdadeira importância dos níveis de criticidade. Com os dois níveis de criticidade acima mencionados distinguem-se, respetivamente, as atualizações que não são urgentes e, como tal, poderão aguardar até obter os recursos disponíveis, das que são realmente urgentes e necessitam de ser instaladas o quanto antes, sendo que nesse caso o utilizador é informado da situação, tendo então de guardar o seu trabalho e terminar as aplicações que estão a bloquear a instalação. Desta forma o serviço de atualizações saberá como lidar com as atualizações, num misto de preocupação com a comodidade do utilizador e a urgência de instalação das mesmas.

- **Deteção de recursos bloqueados:** Ao longo deste documento, por diversas vezes tem sido mencionada a situação em que existem “recursos bloqueados que impedem a instalação das atualizações”. Esta deteção é da responsabilidade do serviço de atualizações, sendo que os recursos necessários à instalação terão de ser descobertos e verificados antes de dar inicio à instalação. Como tal, foi necessário simular a técnica utilizada no mecanismo de deteção de recursos bloqueados do próprio instalador para conseguir identificar os mesmos, antes de este ser executado. Após um período de investigação onde foram estudados a estrutura e funcionamento deste tipo de instaladores, o mecanismo por detrás da deteção de processos a bloquear recursos e a API (*Application Programming Interface*) de sistemas Windows, foi possível recriar este processo e assim fazer a deteção de forma antecipada. Um instalador MSI contém uma base de dados interna com toda a informação necessária à sua execução [40]. Ao explorar essa mesma base de dados programaticamente, é possível identificar quais os recursos que serão necessários à instalação, para posteriormente avaliar a respetiva disponibilidade. Na Figura 21 está representada, através da ferramenta ORCA [52], a tabela “File” da base de dados de um instalador do csSECURE, que contém todos os ficheiros de que a instalação irá necessitar.

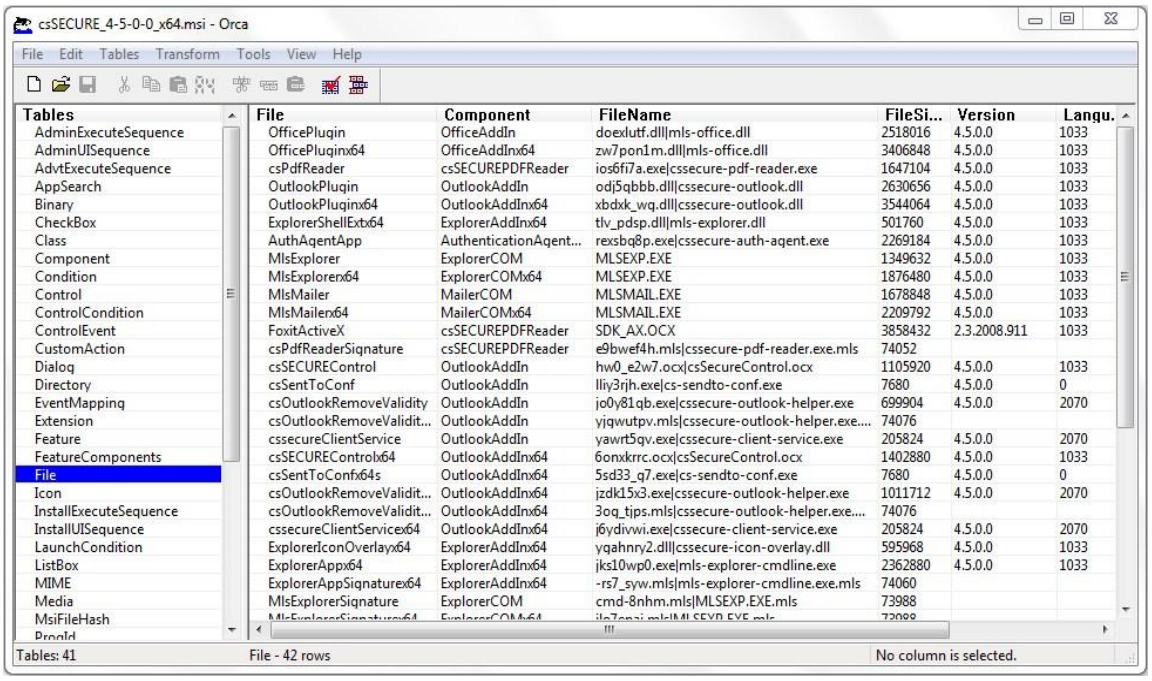


Figura 21 - Tabela "File" da base de dados de um instalador csSECURE

Sabendo quais os ficheiros necessários à instalação, é possível verificar a disponibilidade dos mesmos, no entanto, para isso é necessário saber a diretoria de instalação do csSECURE. Para descobrir a diretoria de instalação do csSECURE, é necessário consultar uma chave de registo específica que é inserida no momento da instalação. Assim sendo, acedendo à base de dados do instalador (armazenado pelo Windows para desinstalação e referente à atual instalação), mais concretamente à tabela "Registry", é possível determinar essa chave de registo e assim obter o [INSTALL_LOCATION], valor que contém diretoria de instalação do produto. Na Figura 22 está representada a tabela do instalador que contém a chave de registo onde é possível consultar a diretoria de instalação do produto. Caso este mecanismo de busca da diretoria de instalação falhe por alguma razão, existe um mecanismo secundário de pesquisa recursiva no sistema de ficheiros que permite também descobrir a diretoria de instalação.

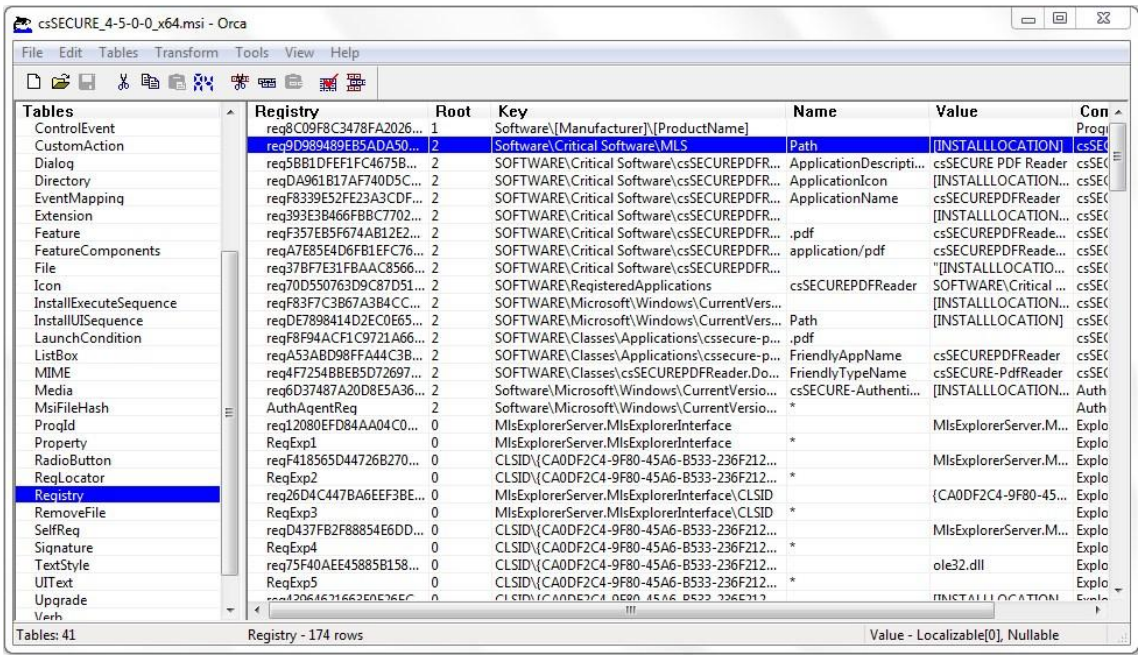


Figura 22 - Tabela "Registry" da base de dados de um instalador csSECURE

Uma vez descoberta a diretoria de instalação é então avaliada a disponibilidade dos ficheiros necessários à instalação, para determinar a viabilidade da mesma, naquele momento. Esta avaliação é feita recorrendo à tecnologia *Restart Manager* [43], que foi criada com o intuito de reduzir o número de reinícios de sistema nas instalações em ambientes Windows. Esta tecnologia permite determinar a disponibilidade dos recursos identificados previamente como necessários à instalação, assim como detetar quais os processos a consumir os mesmos, para que estes possam então ser libertos com o conhecimento do utilizador.

- **Transparência:** Outro ponto de particular interesse nesta plataforma de atualizações e que não se encontra na maioria das soluções no mercado é a transparência de todo o processo de atualizações que apenas necessita da intervenção do utilizador na situação específica em que uma atualização crítica deteta recursos a bloquear a instalação. À parte dessa situação em particular, o utilizador nunca é interrompido para que uma instalação seja executada. Todo o processo se passa de forma totalmente transparente, notificando o utilizador no final da atualização estar concluída. Esta funcionalidade é conseguida executando os instaladores em modo silencioso, gerindo de forma silenciosa qualquer impedimento que surja no momento da instalação.

O facto de esta solução funcionar de forma isolada, sem necessitar de configurações e não obrigar a qualquer tipo de integração no produto a instalar, é por si só uma grande mais-valia, pois oferece outro tipo de liberdade de utilização, contrariamente à maioria

das soluções presentes no mercado. Basta instalar o serviço de atualizações na máquina para o produto ser, a partir desse momento, automaticamente atualizado.

A plataforma foi ainda testada com outro produto que não o csSECURE, o 7Zip, também este com atualizações fornecidas através de instaladores MSI e os resultados foram bastante satisfatórios. Apesar de a plataforma ter sido planeada e pensada no âmbito do csSECURE, esta foi desenvolvida de forma modular e genérica para futuramente vir a ser integrada noutros produtos. Para já, os indicadores são positivos uma vez que o 7Zip, um produto de características bastante distintas das do csSECURE, foi também atualizado através desta plataforma.

3.5 Testes

Dada a metodologia de desenvolvimento ágil utilizada (SCRUM), em cada *sprint* do período de desenvolvimento do projeto está implícita uma fase de testes que permitem validar o grau de finalização das *user stories* e requisitos não funcionais implementados.

Assim sendo, em cada *sprint* foram realizados testes unitários e testes de regressão, com o objetivo de confirmar se as funcionalidades pretendidas foram devidamente implementadas e se todo o trabalho testado e validado previamente continuava funcional, após as novas alterações.

Para além dos referidos testes unitários e de regressão, foram também especificados casos de teste de aceitação, para que a validação de *user stories* e requisitos não funcionais fosse feita de forma objetiva.

Os casos de teste de aceitação poderão ser consultados no documento anexo de Especificação de Casos de Teste de Aceitação [5].

3.6 Segurança

3.6.1 Introdução

Neste capítulo, pretende-se abordar a problemática da segurança, analisando a forma como esta está diretamente relacionada com as atualizações automáticas de *software*.

Ao desenvolver o *csSECURE – Software Update Service*, surgiram algumas questões de segurança com as quais foi necessário lidar, nomeadamente garantir que os pacotes de atualização são de origem fidedigna e que os mesmos não foram corrompidos algures durante o processo de transferência, assim como em relação ao acesso controlado e autorizado de clientes ao servidor, evitando assim uma porta aberta a visitantes indesejados.

Assim sendo, de seguida é apresentado um breve resumo dos principais problemas de segurança que afetam sistemas de atualizações automáticas cliente-servidor como o do presente projeto, bem como da solução proposta para os resolver.

3.6.2 Ameaças de segurança

Sendo este um sistema automático, baseado na arquitetura cliente-servidor, onde os pacotes de atualização são transferidos para os clientes a partir do servidor, estamos perante uma situação de exposição onde os pedidos de resolução de DNS podem ser comprometidos e os dados transferidos intercetados algures durante a comunicação. Estamos portanto a falar dos ataques "DNS *spoofing*" e "*man-in-the-middle*".

- **DNS *spoofing*:** Neste tipo de ataques, os pedidos de resolução DNS são intercetados e modificados no momento em que o cliente pretende aceder ao servidor. Nesse momento o cliente deixará de estar a contactar o servidor fidedigno e terá a sua ligação reencaminhada para onde quer que o atacante a tenha definido, como por exemplo um servidor algures na Web que poderá simular o servidor real, fornecendo pacotes de atualizações com código malicioso.
- ***Man-in-the-middle*:** Este ataque, cujo nome é bastante elucidativo, resume-se à interceção de informação por parte de um interveniente externo que se faz passar por ambas as partes de uma comunicação, sendo que nenhuma delas se apercebe que não está a comunicar diretamente com a outra, podendo este interveniente intercetar ou alterar a informação da maneira que entender. Exemplificando este ataque, mais concretamente no âmbito de uma troca de informação entre cliente e servidor da plataforma *csSECURE – Software Update Service*, um atacante poderia intercetar a comunicação para a transferência de um pacote de atualizações, fazendo-se passar pelo servidor para o cliente e fornecendo-lhe o seu próprio pacote modificado, que poderá conter código malicioso para atingir determinado fim. O cliente, não se apercebendo de que não está a comunicar diretamente com o servidor, recebe o pacote de atualizações e estará assim a instalar na sua máquina *software* indesejado.

Para além destes ataques, que comprometem a integridade e autenticidade da informação, existem ainda outras questões de segurança que é necessário endereçar, como o não-repúdio, a confidencialidade e o acesso controlado aos servidores.

Por não-repúdio entenda-se a incapacidade de negar uma ação ou acesso à plataforma. Num exemplo simples, um cliente não poder negar que rejeitou uma

atualização vinda do distribuidor, um distribuidor não poder negar que inseriu, modificou ou removeu uma atualização da plataforma.

A confidencialidade na comunicação é também um aspeto essencial, uma vez que nas trocas de dados segue informação sensível, como por exemplo endereços de acesso e outros detalhes relacionados com o produto.

Por último e não menos importante, o acesso controlado aos servidores, tanto às consolas de distribuição e administração para controlar os utilizadores que gerem a plataforma, como ao Webservice e servidor Web HTTP, para garantir que apenas os clientes da plataforma podem aceder à mesma, evitando assim uma porta aberta a qualquer intruso com fins maliciosos.

3.6.3 Solução proposta

De modo a fazer frente a todas as questões de segurança identificadas no âmbito deste projeto, foi proposta e implementada a solução que de seguida é apresentada.

- **Não-repúdio:** Para garantir o não repúdio por parte de distribuidores, administradores ou clientes no que toca aos acessos e respetivas ações nos servidores, foi implementado um sistema de *logs* que regista toda a informação considerada relevante. Para efeitos de auditoria, esta informação consiste nos acessos ao servidor e respetivas ações, utilizador, tipo de utilizador, data e hora associados. Tanto o servidor central de distribuição como os servidores nos clientes têm as suas próprias bases de dados que operam de forma independente, onde os respetivos *logs* são armazenados.
- **Confidencialidade:** Para manter a confidencialidade nas trocas de informação em todas as comunicações cliente-servidor central e serviço de atualizações-servidor local, é utilizado o protocolo de comunicação HTTPS (*HyperText Transfer Protocol Secure*), uma implementação do protocolo de comunicação HTTP sobre uma camada adicional de segurança SSL (*Secure Sockets Layer*). Desta forma, qualquer ligação estabelecida entre ambos é encriptada e autenticada através de certificados digitais.
- **Integridade & Autenticidade:** É vital garantir a integridade e autenticidade dos pacotes de atualização quando estes chegam aos clientes, isto é, garantir que os mesmos são de origem fidedigna e não foram alterados algures no processo de transferência. Como tal, os pacotes de atualização são assinados digitalmente, permitindo assim ao cliente fazer uma verificação de segurança antes da instalação. Caso a verificação falhe, o pacote é imediatamente descartado. Os pacotes de atualização são assinados digitalmente com chaves RSA de 2048 bits. A escolha do tamanho das chaves foi feita após uma

consulta às recomendações do *National Institute of Standards and Technology* [6], que afirma que desde o ano 2011 que as chaves de 1024 bits não fornecem a segurança necessária, sendo as chaves de 2048 bits as mais recomendadas. Na figura seguinte está ilustrado um exemplo do processo de assinatura digital para melhor representar como é que a mesma assegura a integridade e autenticidade dos pacotes de atualização.

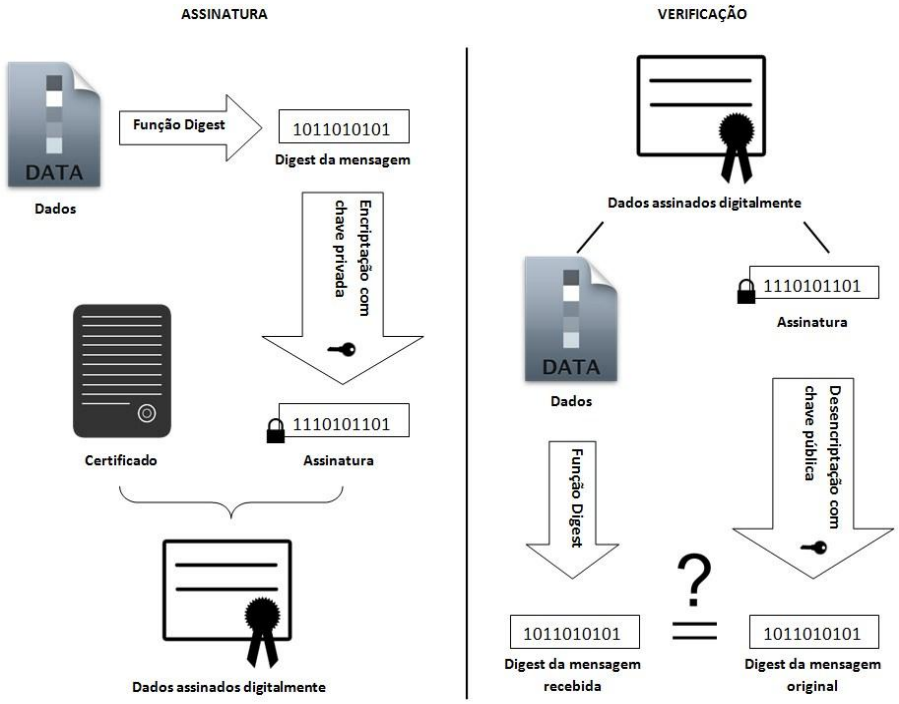


Figura 23 - Processo de assinatura digital e respetiva verificação

As assinaturas digitais funcionam com base em pares de chaves assimétricas e respetivos certificados, podendo ser obtidas através de uma CA (*Certificate Authority*) reconhecida. Ao assinar digitalmente uma mensagem, o que basicamente se está a fazer é a gerar um *message digest* (uma codificação da mensagem que funciona como identificador único), para posteriormente ser encriptado com a chave privada do emissor e enviado ao recetor, em conjunto com a própria mensagem. Se a mensagem não sofrer nenhuma alteração no processo de envio, o *message digest* manter-se-á. O recetor, por sua vez, ao receber a mensagem gera também um *message digest* da mesma, desencripta o *message digest* da mensagem inicial e compara ambos os identificadores. Caso sejam iguais, a validação é bem-sucedida no recetor, que assim tem a garantia da integridade da mensagem. Da mesma forma e uma vez que a chave pública apenas desencripta dados encriptados com a chave privada, se a desencriptação for bem-sucedida, a identidade do emissor da mensagem é

confirmada. A utilização de assinaturas digitais é um processo bastante seguro, sobretudo quando as chaves de encriptação são de elevada complexidade.

- **Acesso controlado:** Um aspeto igualmente importante prende-se com o controlo de acessos aos servidores, que é feito através de autenticação integrada por LDAP (*Lightweight Directory Access Protocol*), mais concretamente *Active Directory* quando o distribuidor, administrador ou serviço de atualizações acede ao respetivo servidor.

Resumindo, na figura seguinte estão representados os mecanismos de segurança acima referidos, integrados no funcionamento da plataforma no âmbito da rede interna do cliente.

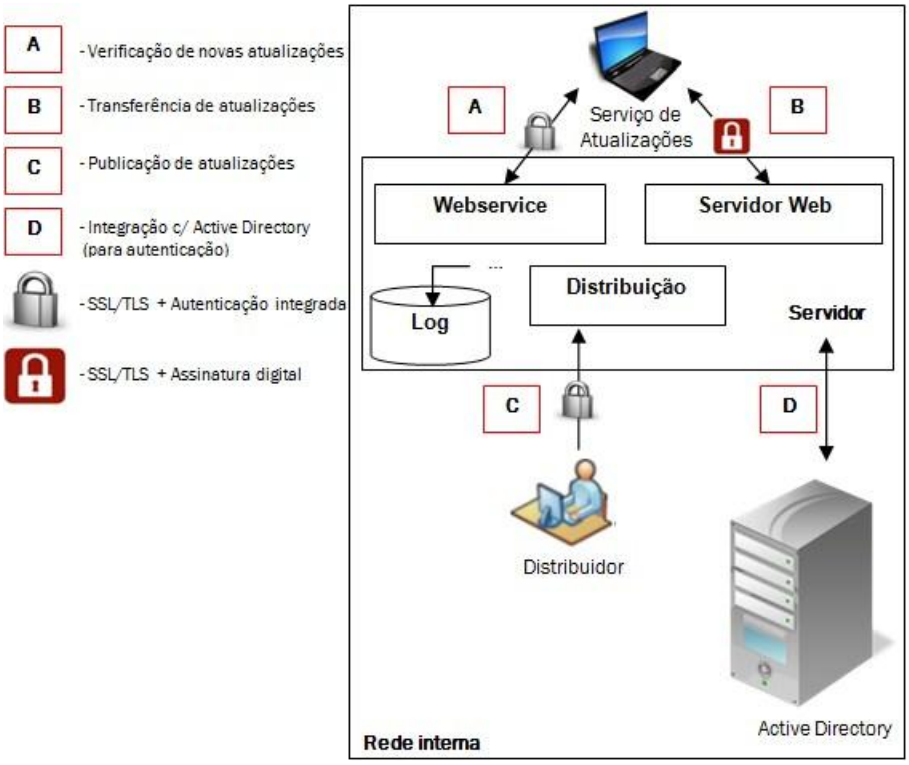


Figura 24 - Segurança do csSECURE - Software Update Service no cliente

Na Figura 24 podemos ver, tal como foi referido, que todas as comunicações serviço-servidor e distribuidor-servidor são protegidas por SSL/TLS e autenticação integrada. Esta autenticação é validada através da integração do servidor com *Active Directory* que por sua vez possui a informação e credenciais de todos os utilizadores no domínio.

Todos os acessos e ações no servidor são registadas em *log*, para efeitos de não-repúdio e auditoria.

Para além da encriptação, autenticação e registo de acessos, estão também representadas a assinatura digital que acompanha o pacote de atualização no momento da sua transferência, para que o cliente e posteriormente o serviço de atualizações comprovem a integridade e autenticidade dos mesmos.

No caso da comunicação entre cliente e servidor, exposta ao exterior, os mecanismos de segurança baseiam-se na encriptação das ligações, na assinatura digital dos pacotes de atualização e na autenticação feita com base em credenciais fornecidas ao cliente para o efeito. A representação deste cenário está presente na Figura 25.

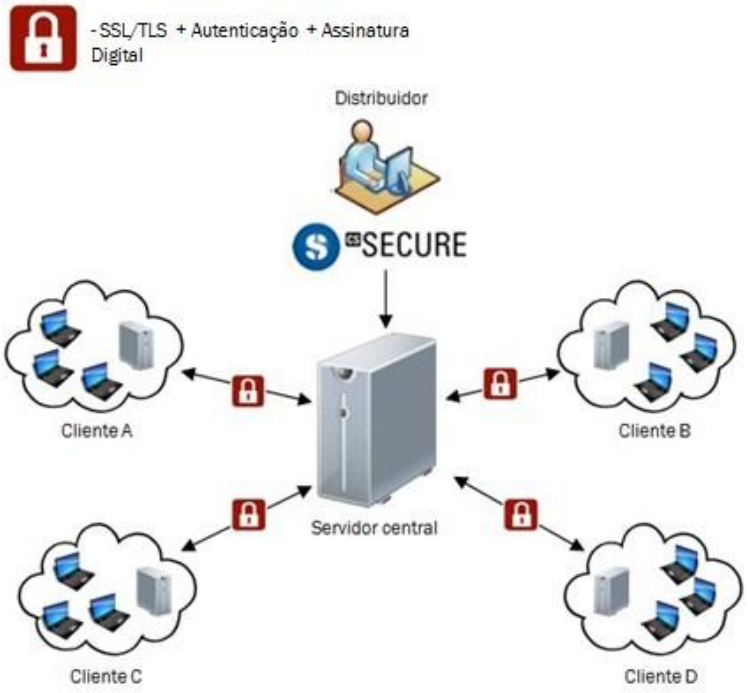


Figura 25 - Segurança do csSECURE - Software Update Service entre cliente e servidor

4 Resultados

Nesta secção é apresentado um pequeno resumo de todo o trabalho realizado pelo estagiário ao longo do projeto de estágio *csSECURE- Software Update Service*, mais concretamente em termos de documentação e *software* produzidos.

4.1 Documentação Produzida

Durante o período de estágio foi produzido o seguinte conjunto de documentos:

- **Metodologia de Desenvolvimento [1]:** Justificação e descrição de todas as práticas e ferramentas utilizadas para colocar em prática a metodologia de desenvolvimento ágil SCRUM.
- **Sprint Backlogs [2]:** Especifica as *user stories* realizadas em cada *sprint* do período de desenvolvimento e apresenta os respetivos *burndown charts, velocity* e *achievement*.
- **Estudo do Estado da Arte [3]:** Estudo de mercado com o objetivo de identificar soluções na área das atualizações automáticas de *software* presentes no mercado, as suas vantagens e desvantagens, assim como as tecnologias e arquiteturas mais utilizadas.
- **Especificação de Requisitos [4]:** Documenta a análise de requisitos efetuada no âmbito do projeto, apresentando o *Product Backlog* e os requisitos não funcionais do *csSECURE – Software Update Service*.
- **Especificação de Casos de Teste de Aceitação [5]:** Define um conjunto de testes para a validação da solução.

4.2 Software Desenvolvido

Tal como referido previamente na secção 3.3.5, a implementação do projeto *csSECURE – Software Update Service* foi guiada e priorizada de forma a obter uma versão funcional do mesmo no tempo disponível, ainda antes de terminar o período de estágio.

Durante este período, foram desenvolvidos três módulos de *software*:

- **Servidor de atualizações:** Servidor onde o distribuidor publica as novas versões de produto e os clientes acedem periodicamente para descarregar as mesmas.
- **Sistema de gestão de bases de dados:** Suporta toda a estrutura de dados necessária ao funcionamento do servidor.
- **Serviço de atualizações:** Serviço instalado nas máquinas dos utilizadores, responsável por atualizar a respetiva versão do csSECURE.

4.2.1 Servidor de atualizações

O módulo servidor de atualizações foi implementado de forma a assumir o papel de servidor central e servidor local (ambos presentes na Figura 8). Desta forma, neste servidor foi implementada uma aplicação Web como consola de distribuição, onde o distribuidor pode publicar novas atualizações e consultar as versões instaladas nas máquinas dos utilizadores (as figuras Figura 26, Figura 27, Figura 28, Figura 29 e Figura 30 apresentam a consola de distribuição). Para responder aos pedidos dos clientes (representados pelos serviços de atualização instalados nas máquinas dos utilizadores), foi também implementado um Webservice, ao qual os mesmos acedem periodicamente para verificar a existência de novas atualizações. O Webservice, em comunicação com o sistema de gestão de base de dados, fornece as informações necessárias aos clientes para que estes possam transferir os pacotes de atualização. Essa transferência é feita através de um servidor Web, que se encontra alojado no próprio servidor do *csSECURE – Software Update Service* e permite o acesso aos pacotes de atualização armazenados no sistema de ficheiros.

Todas as componentes de segurança associadas ao servidor foram também implementadas, desde a encriptação das comunicações cliente-servidor, ao controlo de acessos através de autenticação integrada, mecanismo de *logs* e assinatura digital dos pacotes de atualização.

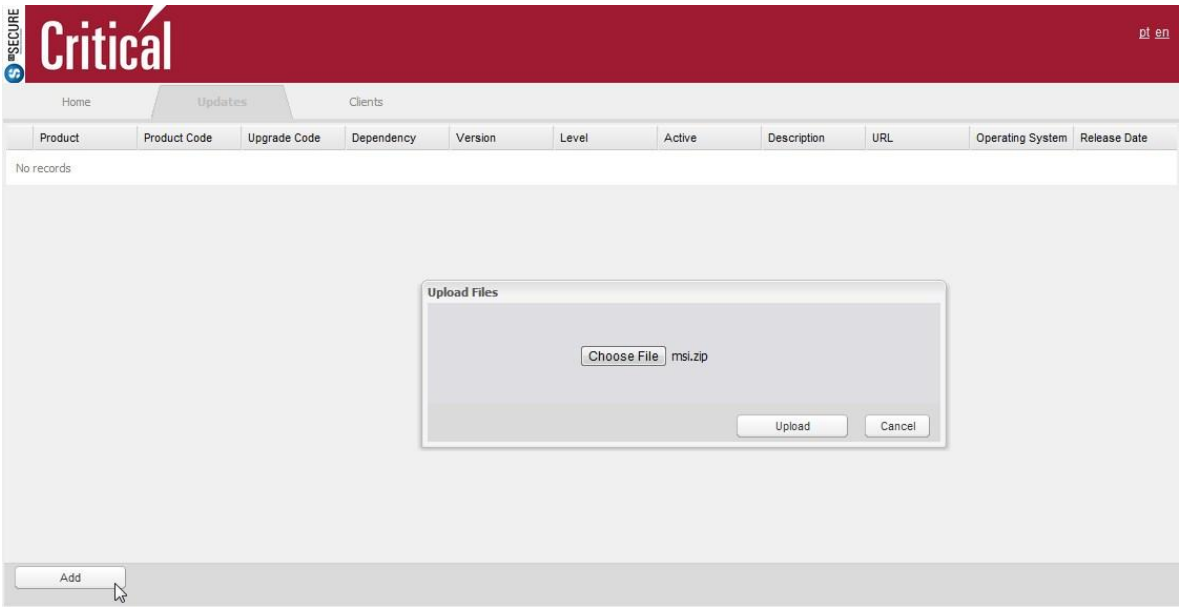


Figura 26 - Consola de distribuição, publicação de atualização (upload de ficheiros)

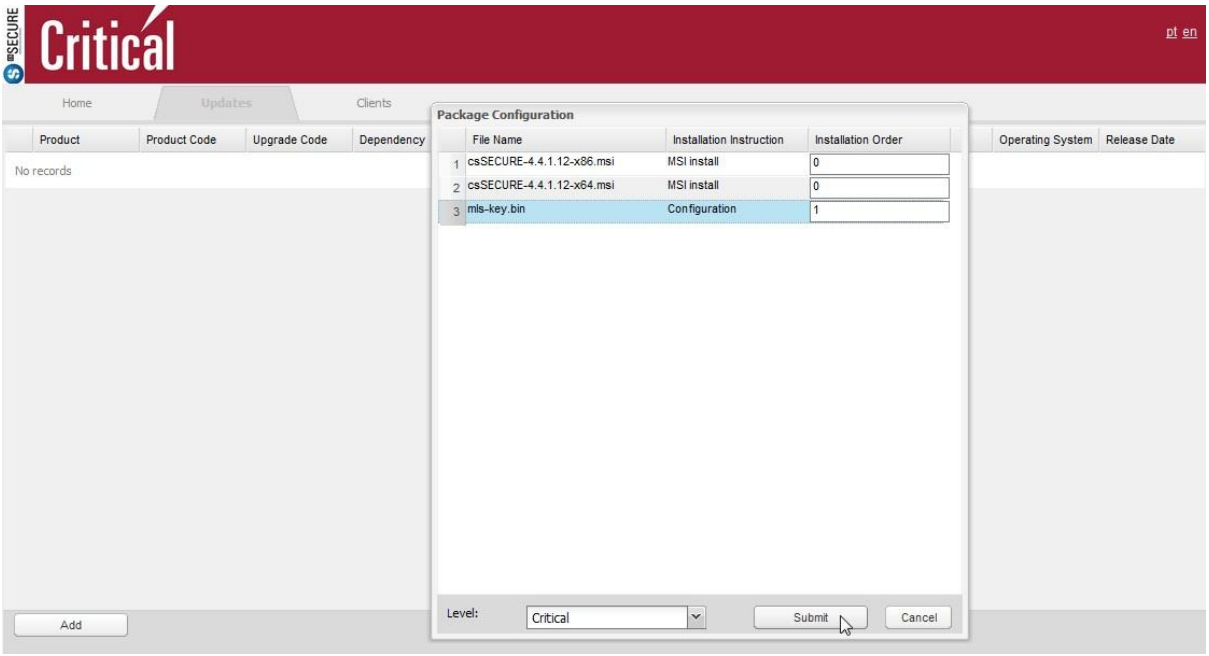


Figura 27 - Consola de distribuição, publicação de atualização (configurações)

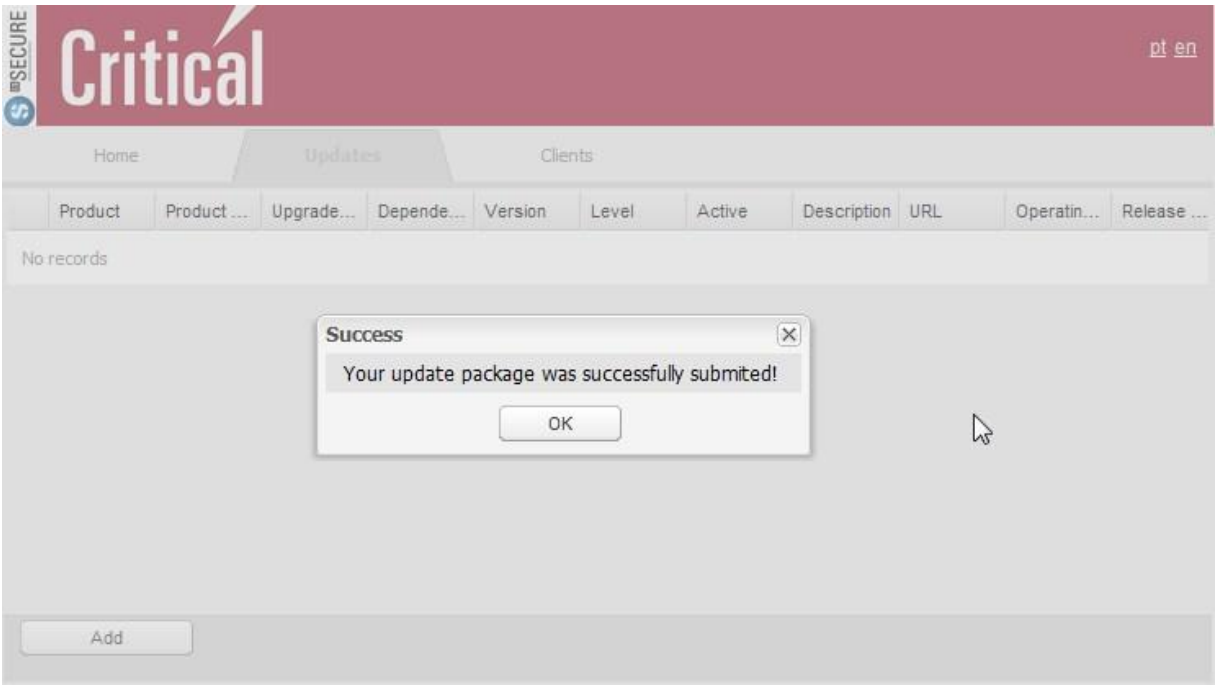


Figura 28 - Consola de distribuição, publicação de atualização (submissão)

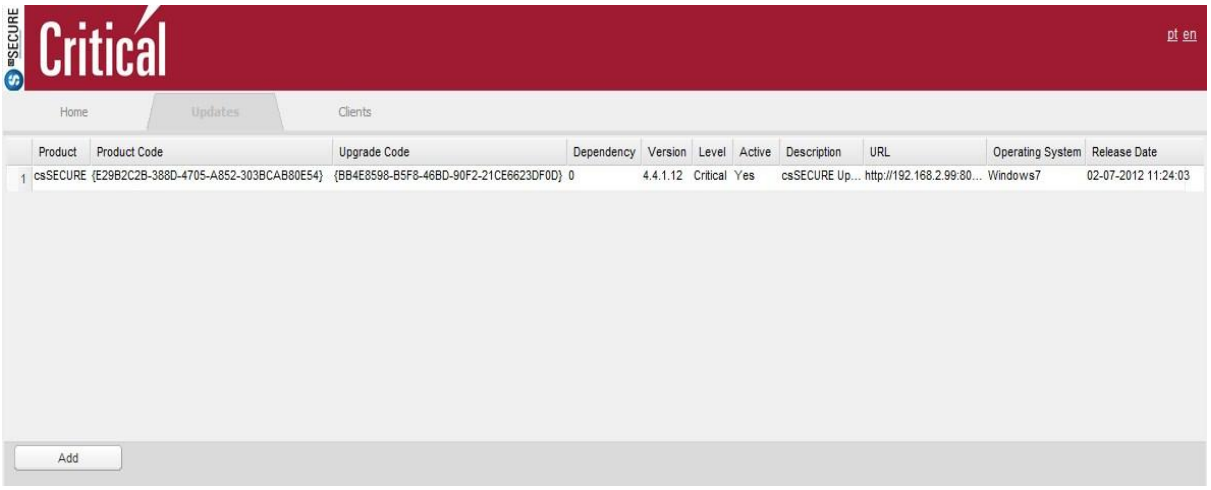


Figura 29 - Consola de distribuição, listagem de versões de produto no servidor

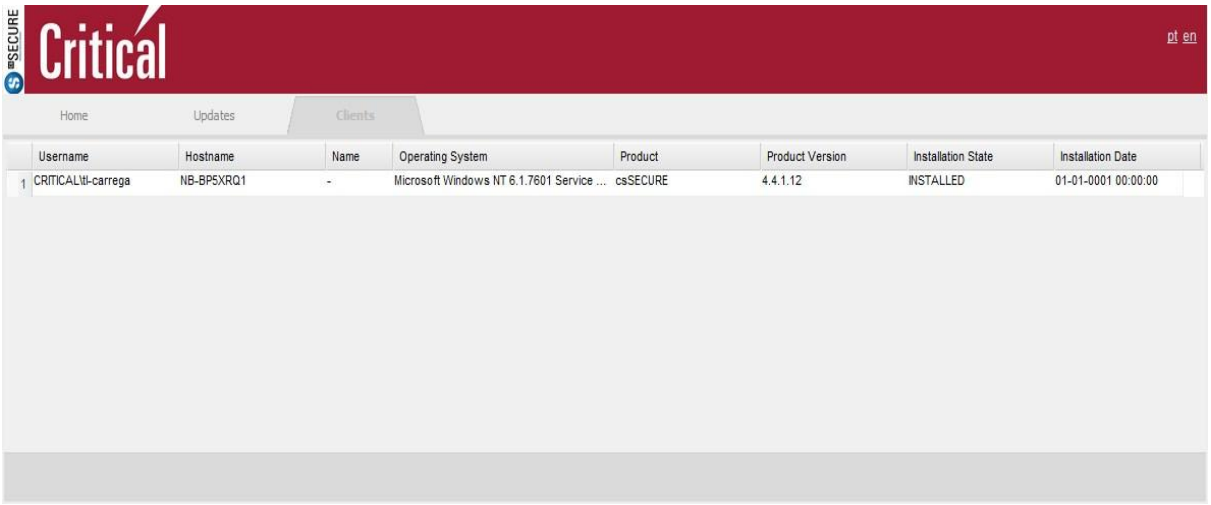


Figura 30 - Consola de distribuição, listagem de versões instaladas

4.2.2 Sistema de gestão de bases de dados

O sistema de gestão de bases de dados Microsoft SQL Server foi implementado de forma a oferecer a estrutura de suporte e armazenamento que o servidor requer. Este módulo interage com todas as bases de dados do sistema em que se encontra inserido, cliente ou servidor. É através do sistema de gestão de bases de dados que qualquer componente do sistema acede à camada de dados da plataforma.

4.2.3 Serviço de atualizações

O serviço de atualizações, instalado nas máquinas dos utilizadores, é basicamente o módulo central de maior importância, uma vez que é através deste que o produto é atualizado (foi na implementação do serviço de atualizações que surgiram os desafios

técnicos mais complexos e aliciantes). O serviço de atualizações apresenta uma arquitetura de dois componentes internos (um serviço e uma aplicação, ambos representados na Figura 15) que, em conjunto são responsáveis pela instalação da nova versão de produto e respetiva notificação ao utilizador. Este módulo foi implementado na sua totalidade no que diz respeito à sua natureza automática, transparente e não intrusiva. No que toca aos tipos de atualização, o único que é suportado funciona com base em instaladores MSI. Outros tipos de atualização ficaram previstos em *Product Backlog* mas não foram implementados, devido a não constarem das prioridades definidas ao longo do período de desenvolvimento. Nas figuras seguintes estão representados alguns detalhes gráficos da reduzida interação do serviço de atualizações com o utilizador.

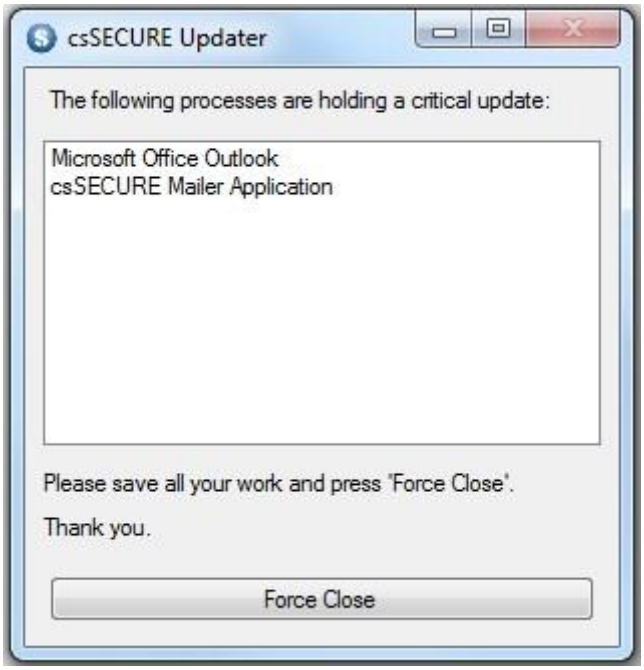


Figura 31 - Serviço de atualizações, atualização crítica com recursos bloqueados requer a intervenção do utilizador

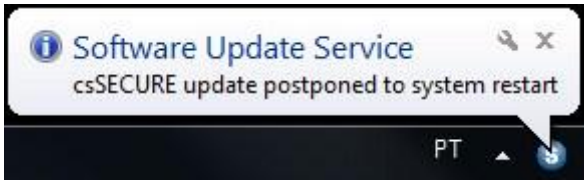


Figura 32 - Serviço de atualizações, notificação de atualização no próximo reiniciar de sistema

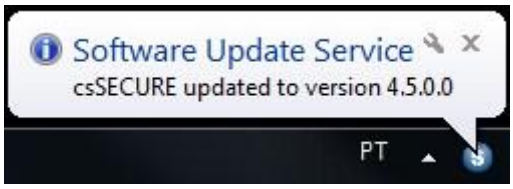


Figura 33 - Serviço de atualizações, notificação de atualização concluída

No serviço de atualizações foi também implementado um mecanismo de segurança que verifica as assinaturas digitais dos pacotes de atualização. A instalação de uma nova versão só é feita se a assinatura digital do respetivo pacote for validada com sucesso.

4.3 Trabalho Futuro

Nesta secção pretende-se identificar o que não foi implementado e que, portanto, representa trabalho futuro que será continuado no final do período de estágio para conferir a maturidade necessária ao *software* para este ser integrado num contexto real, nos clientes do csSECURE.

Em traços gerais, eis os próximos passos que se seguem na lista de prioridades:

- Implementar aplicação Web como consola de administração onde, do lado do cliente, o administrador pode aprovar a atualização e gerir a distribuição das versões de produto pelos utilizadores do domínio através da integração com *Active Directory*.
- Agendar atualizações através da consola de distribuição.
- Instalar servidor central na Critical Software e instalar a plataforma nos clientes do csSECURE, para colocar todo o sistema em funcionamento.
- Implementar suporte a outros tipos de atualização.
- Implementar suporte a múltiplos produtos, não apenas ao csSECURE.
- Integrar plataforma de atualizações com outros produtos da Critical Software e respetivos clientes.

Uma vez cumpridas todas estas etapas, espera-se que o *csSECURE – Software Update Service* se transforme no *Software Update Service*, o que permitirá fazer chegar a todos os clientes da Critical Software as novas versões do csSECURE, entre outros produtos.

5 Conclusões

O presente projeto de estágio teve como principal objetivo a criação de uma solução de atualizações automáticas de *software* para o csSECURE. Pode dizer-se que esse objetivo foi em grande parte atingido. O *csSECURE – Software Update Service* é uma plataforma de atualizações automáticas no modelo cliente-servidor, onde o distribuidor pode facilmente publicar atualizações num servidor central, ao qual os clientes acedem periodicamente transferindo as mesmas para o seu próprio servidor local, para que estas cheguem à máquina do utilizador final, atualizando o csSECURE de forma automática e transparente. Com base em níveis de criticidade, o serviço de atualizações reconhece automaticamente a urgência das atualizações, adaptando assim de forma dinâmica o seu comportamento no momento da instalação, tendo como principal objetivo reduzir o nível de intrusão de todo o processo de atualização do produto.

Neste momento existe já uma versão de pré-produção instalada na Critical Software que se encontra a ser testada por *beta-testers*, para dar início a uma nova etapa deste projeto. Esta passará pelo amadurecimento do *software* e respetiva instalação nos clientes do csSECURE.

Relativamente ao período que agora termina existem alguns aspetos a realçar. Este estágio revelou-se como o projeto mais desafiante, motivador e aliciante de todo o meu percurso académico. Desde o primeiro dia que me senti totalmente identificado com a equipa e o propósito do projeto *csSECURE – Software Update Service*.

O contacto com especialistas da empresa e em particular com ambos os orientadores de estágio, o Professor Doutor Edmundo Monteiro (DEI, FCTUC) e o Eng.º Bernardo Patrão (Critical Software S.A.) foi fundamental. O sucesso deste projeto deve-se em grande parte a toda a passagem de conhecimento e experiência profissional que foi feita pelos mesmos, de forma exemplar, ao longo de todo o período de estágio.

Ao nível de competências adquiridas, é de realçar a evolução na criação e estruturação de documentação técnica, no trabalho em equipa subsequente da integração num ambiente dinâmico de desenvolvimento ágil, no planeamento detalhado e organizado de todas as fases de desenvolvimento de um projeto, nos conhecimentos técnicos de engenharia ao nível da implementação de sistemas distribuídos em ambientes Windows e respetivas tecnologias e, finalmente, nos conhecimentos técnicos na área da segurança.

Em suma, o projeto *csSECURE – Software Update Service* revelou-se uma experiência extremamente aliciante e enriquecedora ao nível pessoal, académico e profissional.

REFERÊNCIAS

[1] Metodologia de Desenvolvimento, Critical Software S.A.
CSW-CSSECPRD-2012-MMO-00360.

[2] Sprint Backlogs, Critical Software S.A.
CSW-CSSECPRD-2012-RPT-02054.

[3] Estudo do Estado da Arte, Critical Software S.A.
CSW-CSSECPRD-2011-RPT-04494.

[4] Especificação de Requisitos, Critical Software S.A.
CSW-CSSECPRD-2011-YRS-05558.

[5] Especificação de Casos de Teste de Aceitação, Critical Software S.A.
CSW-CSSECPRD-2012-TCS-02840.

[6] NAppUpdate. [Online] https://github.com/synhershko/NAppUpdate.
[Viewed: October, 2011]

[7] WyUpdate. [Online] http://wyday.com/wyupdate.
[Viewed: October, 2011]

[8] Sharp AutoUpdater. [Online] http://csautoupdater.sourceforge.net.
[Viewed: November, 2011]

[9] TrueUpdate. [Online] http://www.indigorose.com/products/trueupdate.
[Viewed: November, 2011]

[10] AutoUpdate Plus. [Online] http://www.autoupdateplus.com.
[Viewed: November, 2011]

[11] The Software Update Wizard. [Online] http://www.powerprogrammer.co.uk.
[Viewed: November, 2011]

[12] AppLife Update. [Online] http://www.kineticjump.com/update.
[Viewed: November, 2011]

[13] Posey, B. Using Group Policy Objects to Deploy Applications (2008). [Online]
http://www.windowsnetworking.com/articles_tutorials/group-policy-deploy-applications.html.
[Viewed: December, 2011]

[14] Bellissimo, A.; Burgess J.; Fu, K. Secure Software Updates: Disappointments and New Challenges. [Online] http://people.cs.umass.edu/~kevinfu/papers/secureupdates-hotsec06.pdf.
[Viewed: December, 2011]

[15] White, D. A Unified Architecture for Automatic Software Updates. (June 2004).
[Viewed: December, 2011]

[16] Martin, R. Design Principles and Design Patterns. [Online]
http://www.objectmentor.com/resources/articles/Principles_and_Patterns.pdf.
[Viewed: March, 2012]

[17] Bredemeyer. Software Architecture, Architects and Architecting. [Online]
http://www.bredemeyer.com/index.html.
[Viewed: February, 2012]

[18] Clements, P.; Bachman, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Merson, P.; Nord, R.; Stafford, J. Documenting Software Architectures: Views and Beyond, Second Edition (October 2010). SEI Series in Software Engineering.
[Viewed: July, 2012]

[19] Microsoft. Microsoft Application Architecture Guide (Second Edition). Microsoft Patterns & Practices. [Online] http://msdn.microsoft.com/en-us/library/ee658124.
[Viewed: July, 2012]

[20] Microsoft. Application Archetypes. Microsoft Patterns & Practices. [Online]
http://msdn.microsoft.com/en-us/library/ee658104.aspx.
[Viewed: March, 2012]

[21] Microsoft. Understanding Windows Services Architecture (2005). [Online]
http://technet.microsoft.com/en-us/library/aa998749%28EXCHG.65%29.aspx.
[Viewed: January, 2012]

[22] Brown, K. The .NET Developer's Guide to Windows Security (2004). Microsoft .NET Development Series.
[Viewed: February, 2012]

[23] InTech. Applied Cryptography and Network Security (March 2012). [Online]
http://www.intechopen.com/books/applied-cryptography-and-network-security.
[Viewed: May, 2012]

[24] Microsoft TechNet. Cryptography for Network and Information Security. [Online] http://technet.microsoft.com/en-us/library/cc962027.
[Viewed: February, 2012]

[25] Burnett, S.; Paine, S. RSA Security's Official Guide to Cryptography. RSA Press.
[Viewed: February, 2012]

[26] Recommendation for Key Management, National Institute of Standards and Technology (March 2007). [Online] http://csrc.nist.gov/publications/nistpubs/800-57/SP800-57-Part2.pdf.
[Viewed: May, 2012]

[27] Microsoft. Managing Certificates with Certificate Stores. [Online] http://msdn.microsoft.com/en-us/library/windows/desktop/aa386971%28v=vs.85%29.aspx.
[Viewed: June, 2012]

[28] Atwood, J. Keeping Private Keys Private (February 2006). [Online] http://www.codinghorror.com/blog/2006/02/keeping-private-keys-private.html.
[Viewed: June, 2012]

[29] Liu, S. Authentication in ASP.NET Web Services. [Online] http://progtutorials.tripod.com/Authen.htm.
[Viewed: April, 2012]

[30] Microsoft. Building Secure Web Services. Microsoft Patterns & Practices. [Online] http://msdn.microsoft.com/en-us/library/ff648643.aspx.
[Viewed: April, 2012]

[31] Microsoft. HTTP Security and ASP.NET Web Services. [Online] http://msdn.microsoft.com/en-us/library/ms996415.aspx.
[Viewed: June, 2012]

[32] ForumSystems. Anatomy of a Web Services Attack: A Guide to Threats and Preventive Countermeasures. [Online] http://www.forumsys.com/resources/resources/whitepapers/Anatomy_of_Attack_wp.pdf.
[Viewed: April, 2012]

[33] Ye, R. Authenticated Software Update. [Online] http://iris.lib.neu.edu/cgi/viewcontent.cgi?article=1000&context=comp_sci_diss.
[Viewed: April, 2012]

[34] CodePlex. DotNetZip Library. [Online] http://msdn.microsoft.com/en-us/library/ms685141%28v=vs.85%29.aspx.
[Viewed: March, 2012]

[35] Microsoft. Understanding and Configuring User Account Control. [Online] http://technet.microsoft.com/en-us/library/cc709628.aspx.
[Viewed: April, 2012]

[36] Microsoft. User Account Control. [Online] http://msdn.microsoft.com/en-us/library/windows/desktop/aa511445.aspx.
[Viewed: April, 2012]

[37] Microsoft. Windows Service Applications. [Online] http://msdn.microsoft.com/en-us/library/ms685141%28v=vs.85%29.aspx.
[Viewed: January, 2012]

[38] Microsoft. About Windows Installer. [Online] http://msdn.microsoft.com/en-us/library/aa367449.aspx.
[Viewed: April, 2012]

[39] Microsoft. Windows Installer. [Online] http://msdn.microsoft.com/en-us/library/cc185688%28VS.85%29.aspx.
[Viewed: April, 2012]

[40] Microsoft. Windows Installer Database Tables. [Online] http://msdn.microsoft.com/en-us/library/windows/desktop/aa368259%28v=vs.85%29.aspx.
[Viewed: April, 2012]

[41] Microsoft. Windows Installer Command-Line Options (Msiexec). [Online] http://msdn.microsoft.com/en-us/library/windows/desktop/aa367988%28v=vs.85%29.aspx.
[Viewed: April, 2012]

[42] Microsoft. Windows Installer Error Messages. [Online] http://msdn.microsoft.com/en-us/library/windows/desktop/aa372835%28v=vs.85%29.aspx.
[Viewed: April, 2012]

[43] Microsoft. Restart Manager Reference. [Online] http://msdn.microsoft.com/en-us/library/windows/desktop/aa373656%28v=vs.85%29.aspx.
[Viewed: May, 2012]

[44] Microsoft. Guidelines for Applications and Services. [Online] http://msdn.microsoft.com/en-us/library/windows/desktop/aa373652%28v=vs.85%29.aspx.

[45] PInvoke. Pinvoke .NET Developers WIKI. [Online] http://www.pinvoke.net/.

[46] Microsoft. Introduction to Internet Information Services Architecture. [Online] http://learn.iis.net/page.aspx/101/introduction-to-iis-architecture.

[47] Mitch Lacey, Microsoft (2010). MSF for Agile Software Development v5.0, Process Guidance. [Online] http://msdn.microsoft.com/en-us/library/dd380647.aspx.

[48] Mitch Lacey & Associates, Inc. Introduction to Agile Software Development (SCRUM). [Online] http://www.mitchlacey.com/intro-to-agile/scrum.

[49] Viewing the Hour Burndown Chart (2012). Documentation for GreenHopper [Online] https://confluence.atlassian.com/display/GH/Viewing+the+Hour+Burndown+Chart. [Viewed: July, 2012]

[50] Microsoft. Microsoft Internet Information Services – Dynamic IP Restrictions [Online] http://www.iis.net/download/dynamiciprestrictions. [Viewed: June, 2012]

[51] Microsoft. Business Layer Guidelines. [Online] http://msdn.microsoft.com/en-us/library/ee658103.aspx [Viewed: July, 2012]

[52] Microsoft. ORCA. [Online] http://msdn.microsoft.com/en-us/library/windows/desktop/aa370557%28v=vs.85%29.aspx [Viewed: May, 2012]

ANEXOS

NOTA: devido ao estatuto de confidencialidade dos anexos, estes não serão submetidos na plataforma, nem impressos. No entanto, estes documentos estão disponíveis e poderão ser consultados pelo júri nos CDs entregues juntamente com o relatório final de estágio.

[Anexo - 1] Metodologia de Desenvolvimento, Critical Software S.A.
CSW-CSSECPRD-2012-MMO-00360.

[Anexo - 2] Sprint Backlogs, Critical Software S.A.
CSW-CSSECPRD-2012-RPT-02054.

[Anexo - 3] Estudo do Estado da Arte, Critical Software S.A.
CSW-CSSECPRD-2011-RPT-04494.

[Anexo - 4] Especificação de Requisitos, Critical Software S.A.
CSW-CSSECPRD-2011-YRS-05558.

[Anexo - 5] Especificação de Casos de Teste de Aceitação, Critical Software S.A.
CSW-CSSECPRD-2012-TCS-02840.