Universidade do Minho

Escola de Engenharia

Mestrado em Engenharia Informática

Um Sistema de Controlo de Acessos Baseado no Modelo Cargo-Organização

José Pedro Vilaça Novais

Orientador: Prof. Doutor Pedro Nuno Miranda de Sousa

31 de Outubro de 2011

Agradecimentos

Este trabalho resultou de um longo empenho e esforço, em parte acrescido pela necessidade de o conciliar com obstáculos originados pelas vicissitudes da vida. No entanto, os resultados obtidos não teriam sido possíveis sem ajuda e apoio que me foram dados ao logo deste trabalho.

Em primeiro lugar quero agradecer ao Professor Pedro Nuno Sousa pela sua disponibilidade, interesse e orientação académica. Agradeço também ao meu orientador na empresa, Nuno Ribeiro, em especial pela sua compreensão.

Um agradecimento a todos os colegas da Ubiwhere pelo acolhimento, espirito de companheirismo e excelente ambiente de trabalho.

Por último, e não menos importante, um grande agradecimento à minha família e amigos por todo o apoio que me deram ao longo deste percurso, em especial nos momentos em que se esmoreciam os ânimos.

Resumo

O paradigma do controlo de acessos, em especial o controlo de acesso à informação, tem vindo a mudar nos últimos anos. Controlo este que inicialmente era efetuado pelas próprias aplicações de forma isolada e autónoma, sem a possibilidade de consultarem ou se integrarem com qualquer sistema centralizado. Todavia, com o crescente uso das tecnologias de informação nas organizações, novas soluções (tais como os serviços de diretoria LDAP) têm vindo a ser adotadas com o intuito de dar resposta à necessidade de uma política de acessos unificada e coesa, transversal aos diversos serviços e aplicações. Estas soluções representam uma mais-valia no desempenho das tarefas organizacionais.

Tendo em conta esta necessidade, este trabalho propõe uma nova solução para o controlo de acessos físicos e lógicos através da apresentação e implementação de um novo modelo de controlo de acessos baseado no par Cargo-Organização. É ainda apresentada e implementada neste projeto uma nova abordagem no controlo de acessos lógicos, sendo esta assim capaz de interagir e configurar aplicações que carecem do suporte de protocolos e mecanismos padrão para o controlo de acessos.

Abstract

The access control paradigm has been changing in the past years, specially what regards the access control to information. Information access control was originally achieved independently by each application without querying or interacting with a central system. However, due to the increasing usage of information technologies, new solutions (such as LDAP services) have been put in place in order to obtain a unified access policy across different services and applications. These solutions greatly improve the performance of organizational tasks.

This project aims to present a new access control solution for both physical and logical layers. Therefore, a new access control model based on the pair Role/Organization is presented, as well as a concrete implementation of this model. Bearing in mind that not all applications support access control protocols, another approach (which allows the interaction with these applications) is also taken. In this context, after presenting the conceptual organization of the proposed solution, along with some implementation details, some illustrative evaluation tests are also presented and discussed.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

NIST National Institute of Standards and Technology 7

ANSI American National Standards Institute 7

INCITS International Committee for Information Technology Standards

RBAC Role Based Access Control 7

OrBAC Organization Based Access Control 13

ROBAC Role Organization Based Access Control 7

SoD Separation of Duty 11

DAC Discretionary Access Control 1

MAC Mandatory Access Control 1

B2E Business to Employee 16

Capítulo 1

Introdução

1.1 Enquadramento e Motivação

Desde o instante em que uma nova organização é criada, o controlo de acessos físicos às instalações e recursos da organização é alvo de atenção e investimento em virtude da preservação dos seus bens. Todavia, fruto da atual era do conhecimento, a segurança da informação e propriedade intelectual das organizações torna-se, em muitos casos, o mais importante.

Essencialmente caracterizado pela proteção da informação com base no seu nível de sensibilidade, Bell-LaPadula [1] foi um dos primeiros modelos controlo de acesso que surgiu. Seguindo-se os modelos do tipo Discretionary Access Control (DAC) e Mandatory Access Control (MAC) [2]. No entanto, estes modelos são bastante restritos, baseados numa comparação directa dos atributos dos utilizadores e dos objetos. Em alternativa e complemento, vários modelos de controlo de acesso têm sido propostos, nomeadamente o Task-Based Access Control (TBAC) [3] e o Role Based Access Control [4, 2, 5, 6]. No entanto, nenhum destes modelos tem satisfeito plenamente as necessidades de controlo de acesso do ponto de vista organizacional, em especial para as organizações estruturadas em sub-organizações e departamentos. Em geral, são baseados apenas na negação e restrição de permissões aos membros de uma organização de acordo com o cargo ou tarefa que desempenham. Tradicionalmente, as regras são aplicadas a uma organização como um todo. No entanto, por vezes há a necessidade de representar uma organização como um conjunto de sub-organizações, em que cada sub-organização tem alguma autonomia nas suas políticas de acesso. Com o objetivo de colmatar estas lacunas, surgem os modelos Organization Based Access Control (OrBAC) [7] e Role Organization Based Access Control.

Atualmente existem várias tecnologias e soluções quer para o controlo de acessos físicos, quer para controlo de acessos lógicos. Como por exemplo o Active Directory da Microsoft. No entanto, soluções deste tipo são fechadas e proprietárias, não sendo compatíveis com produtos de outros fabricantes. Além disso, estas soluções são muito caras, implicando elevados encargos financeiros para as pequenas e médias empresas/organizações. Recentemente surgiram algumas alternativas para controlo de acessos lógicos baseadas nas tecnologias eXtensible Access Control Markup Language (XACML) [8, 9], XML Access Control Language (XACL) [10] e Security Assertion Markup Language (SAML) [11], tais como open PREMIS [12]. Contudo, estas soluções, para além de não poderem comunicar com aplicações que não implementam estes mesmos protocolos, são orientadas ao controlo de acessos lógicos deixando de parte o controlo de acessos físicos.

De encontro com esta necessidade e com a crescente procura por este tipo de sistemas, surge a ideia de se criar um sistema de informação de suporte ao controlo de acessos dos recursos humanos numa empresa. Em complemento ao controlo de acesso, este sistema deve guardar e manter o estado sobre quem acedeu, quando, e de onde se processaram os acessos à empresa. Este propósito geral do desenvolvimento de um sistema de controlo de acessos deste tipo surge no âmbito da implementação das normas ISO 9001:2008 [13] e NP4457 [14] na empresa Ubiwhere. A Ubiwhere[1] é uma empresa de investigação e desenvolvimento com enfoque em redes, telecomunicações e computação ubíqua.

1.2 Objetivos

O principal objetivo deste trabalho, como já referido, é o desenvolvimento de um sistema de controlo de acessos físicos a uma organização, mas não só. O acesso à informação (acessos lógicos) da empresa também deve ser levado em consideração na implementação do modelo escolhido. Um correto e apropriado controlo de acesso dos recursos humanos da organização às suas instalações deve ser assegurado pelo sistema. Na componente de controlo de acesso físico às instalações da empresa serão implementados níveis de privilégios e hierarquias de cargos. Por exemplo, apenas determinados utilizadores terão o privilegio de aceder em horários extra laborais. A autenticação dos utilizadores deve ser também adequada ao contexto, caso um utilizador seja o primeiro a chegar às instalações uma autenticação híbrida mais reforçada será utilizada.

[1] www.ubiwhere.com

Neste contexto, para se atingir o objetivo principal, definem-se como objetivos específicos deste trabalho os seguintes pontos:

- Um estudo dos atuais modelos de controlos de acesso e cenários de aplicação. Do estudo destes modelos deve resultar uma implementação, após uma adaptação se necessário, de um modelo adequado ao cenário em questão.
- É objetivo deste projeto o estudo detalhado das tecnologias Radio Frequency Identification (RFID), Cartão do Cidadão e Biometrias para a autenticação presencial dos utilizadores do sistema. A tecnologia adotada, para além do fator económico que é sempre determinante, dependerá de fatores tais como a escalabilidade, estado de implementação atual no país e evolução da tecnologia. Um ponto de equilíbrio deve ser estabelecido de forma a tornar o sistema eficaz e simples de usar.
- O sistema de informação para além do controlo de acesso físico deve também efetuar os registos de presenças e tempo de assiduidade - *time attendance* - permitindo aos colaboradores da empresa a posterior consulta (e.g. a que horas um funcionário entrou e saiu, tempos/intervalos diários, faltas, etc.).
- Durante toda a fase de desenvolvimento do sistema deve ser considerada a sua adaptabilidade e escalabilidade. O sistema deve ser o mais modular possível para que, independentemente da tecnologia de autenticação escolhida, se possa adaptar a novos cenários sem necessidade de profundas alterações.
- É objetivo complementar deste projeto conceber um sistema hábil de efetuar um correto controlo de acessos lógicos à informação da empresa. Esta parte envolverá os sistemas de informação atualmente em vigor na empresa: SVN, Redmine (gestão de projetos), LDAP e SAMBA.
- O sistema deve ainda ser passível de ser integrado e administrado pela Intranet da empresa.

1.3 Sumário das Principais Contribuições

As principais contribuições resultantes do trabalho desenvolvido consistem na criação de um modelo de acessos baseado no par Cargo-Organização e na implementação de uma aplicação para controlo de acessos físicos e lógicos de utilizadores com base nesse modelo. O modelo criado é, até então, o único modelo que associa as permissões ao par cargo-organização, tornando-o

assim capaz de diferenciar os cargos de acordo com cada organização. A aplicação desenvolvida, através deste modelo, faz uma gestão dos controlos de acessos físicos e lógicos tendo em conta as necessidades de cada sub-organização ou departamento. Como fruto deste trabalho foi publicado o artigo *A Unifying Role and Organization Based Access Control* [15], sendo o mesmo apresentado na conferência sobre redes de computadores CRC2010.

1.4 Organização da Dissertação

Este documento encontra-se organizado em seis capítulos. Cada cada capítulo é descrito da seguinte forma:

- Introdução: neste capítulo é feito o enquadramento e contextualização do trabalho expondo o âmbito da sua origem, a introdução do tema geral da dissertação e os principais objetivos do trabalho a desenvolver.
- Modelos de Controlo de Acesso: este capítulo apresenta a base teórica dos vários modelos de acesso estudados e que mais contribuíram para este trabalho. É ainda detalhado o novo modelo de acessos baseado no par Cargo-Organização criado, e implementado, no âmbito deste projeto.
- Tecnologias de Controlo de Acesso Físico: são abordadas as três alternativas de tecnologias de autenticação no controlo de acessos físicos, sendo também justificadas as opções tomadas neste contexto.
- Desenvolvimento do Sistema: subdividido em quatro secções, este capítulo descreve o ambiente de desenvolvimento e implementação de todo o sistema, o modelo de dados e a arquitetura da aplicação. Nas duas últimas secções é detalhada a implementação dos módulos de controlo de acessos físicos e lógicos integrados na aplicação desenvolvida.
- Testes e Avaliação do Sistema: neste capítulo é inicialmente descrito o ambiente no qual os testes foram realizados, seguindo-se a metodologia de testes. De seguida serão apresentados e avaliados os resultados dos vários testes de desempenho efetuados ao módulo de acessos físicos da aplicação. Em complemento e de forma mais ilustrativa, serão também apresentados os testes efetuados ao módulo de acessos lógicos.
- Conclusões: neste capítulo são apresentadas as principais conclusões obtidas a partir do trabalho. Será feita uma breve análise das principais etapas do trabalho feito, e principais

contribuições resultantes. Por último, serão abordados futuros desenvolvimentos de forma a melhorar a solução implementada.

Capítulo 2

Modelos de Controlo de Acesso

Neste capítulo, de entre os vários modelos de controlo de acessos estudados, são introduzidos aqueles que mostraram ser mais relevantes para o desenvolvimento deste trabalho de acordo com os seus objetivos. Com base nestes modelos, em especial o modelo Role Organization Based Access Control (ROBAC) da secção 2.3, é ainda apresentado um novo modelo baseado no par Cargo-Organização, fruto deste trabalho. É apresentada também uma breve comparação entre os modelos e os seus cenários de aplicação. As suas vantagens e desvantagens são abordadas, bem como quais os principais motivos que levaram à escolha do modelo adotado.

2.1 Role Based Access Control

O modelo de controlo de acesso baseado no cargo do utilizador, do inglês Role Based Access Control (RBAC), é principalmente caracterizado por os utilizadores adquirirem permissões através das funções que desempenham dentro da organização. Como o próprio nome indica, todo o controlo de acessos é baseado nas funções/cargos (*Roles*). O principal conceito deste modelo consiste na atribuição de permissões aos cargos e os utilizadores serem corretamente associados a um ou mais cargos dentro da organização, através dos quais adquirem o acesso aos objetos (ver Figura 2.1).

Devido à falta de normalização deste modelo, com base no primeiro modelo de controlo de acesso baseado no cargo, introduzido em 1992 [5], e na família de modelos ROBAC96 [6], em 2000 o National Institute of Standards and Technology (NIST) propôs uma uniformização do modelo RBAC [4], que veio posteriormente a ser adotado como padrão pela American National Standards Institute (ANSI) INCITS 359-2004 [16]. Este padrão [4], apresenta o modelo RBAC

dividido em quatro níveis, Flat RBAC, Hierarchy RBAC, Constraint RBAC e Symetric RBAC, que sucessivamente vão acrescentando funcionalidades ao sistema.

2.1.1 Flat RBAC

Neste primeiro nível, Flat RBAC, são introduzidas as três entidades fundamentais do modelo, *Utilizador*, *Permissão* e *Cargo* (Role). Um utilizador pode ser um ser humano, um agente, ou até mesmo um sistema externo e independente. As permissões definem o direito ao acesso a um determinado objeto, permitindo a execução de ações sobre eles. Os cargos geralmente representam profissões ou postos de trabalho numa organização. Os conjuntos de associações Utilizador-Cargo *(UA)* e Permissões-Cargo *(PA)* têm de suportar um relacionamento de muitos para muitos não estabelecendo nenhum limite. Deste modo, um *cargo* representa, por um lado, um conjunto de *permissões* e por outro um conjunto de *utilizadores*. Desta forma, a um utilizador podem ser atribuídos vários cargos e vice-versa. O mesmo acontece com as associações Permissões-Cargo. É requerido que as implementações deste modelo básico suportem uma análise das relações entre cargos e utilizadores, ou seja, dado um cargo saber quais os utilizadores a ele associados ou dado um utilizador determinar quais os cargos por ele desempenhados dentro da organização. Esta funcionalidade visa auxílio à administração, para conservar um correto relacionamento entre estas duas entidades. Estes requisitos funcionais são considerados como o conjunto mínimo necessário funcional para a implementação de um modelo de controlo de acessos baseado no cargo, o que torna este primeiro nível do modelo uma aproximação ao modelo de controlo de acesso orientado ao grupo [5, 6, 4].

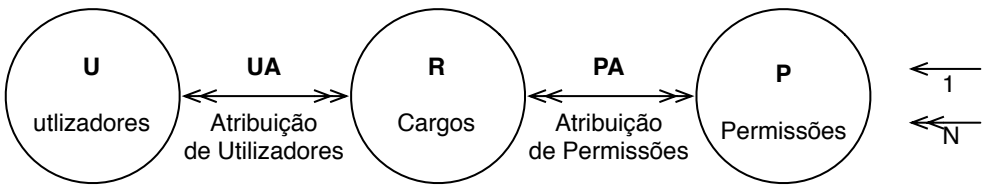


Figura 2.1: Modelo Básico de Controlo de Acessos Baseado no Cargo.

2.1.2 Hierarchy RBAC

A autoridade e subordinação entre os elementos de uma organização é espelhada na hierarquia dos seus cargos. A hierarquia de cargos resulta implicitamente numa herança de permissões dos cargos hierarquicamente inferiores (cargos júnior) para os cargos hierarquicamente acima.

Esta necessidade de hierarquia, herança e até partilha de permissões, tornam o suporte desta funcionalidade quase imprescindível à implementação de um sistema RBAC. Motivo pelo qual esta funcionalidade é requisito no segundo nível do modelo (Hierarchy RBAC), ilustrado na Figura 2.2. Contudo, nada é especificado pelos autores relativamente aos tipos de hierarquia que podem ser suportados pelo modelo, pois é uma questão interna de cada organização. Contudo, é desejável que o sistema suporte vários tipos de hierarquias: Hierarquia em árvore (ver Figura 2.3) - em que a posição no topo da árvore é autoridade máxima que herda todas as permissões dos cargos abaixo; Hierarquia em árvore invertida - onde há um cargo com um conjunto base de permissões que vão ser herdadas pelos cargos acima (ver Figura 2.4); a junção destas destas duas hierarquias dá origem à hierarquia militar, em que há um posto de autoridade máxima e mínima. Há ainda em alguns cenários o caso em que não existe nenhuma função de autoridade máxima nem mínima, como ilustrado pela Figura 2.5.

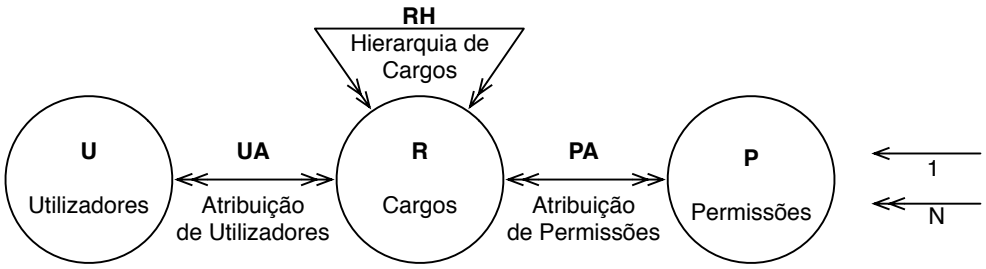


Figura 2.2: Modelo de Controlo de Acessos Baseado no Cargo com Hierarquia.

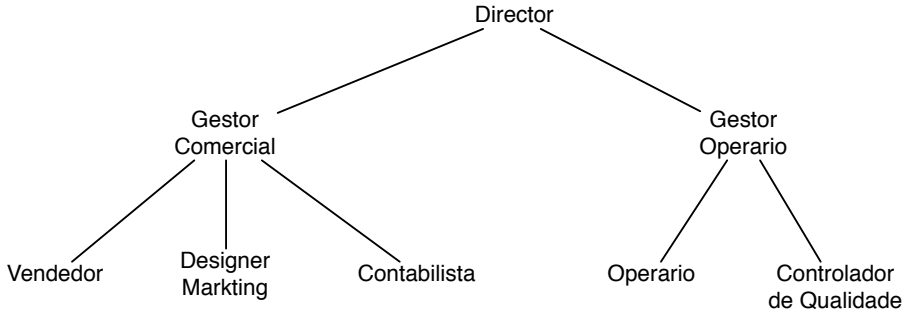


Figura 2.3: Hierarquia em Árvore

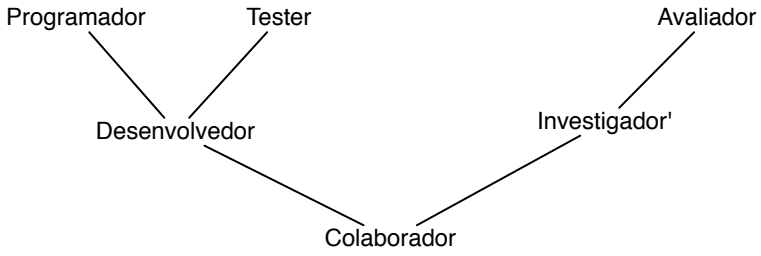


Figura 2.4: Hierarquia em Árvore Invertida

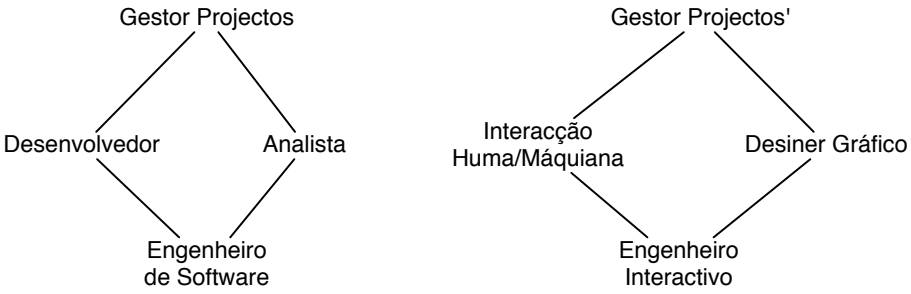


Figura 2.5: Hierarquia sem função máxima/mínima

2.1.3 Constrained RBAC

O facto do modelo baseado no cargo permitir que a um utilizador seja atribuída mais do que um cargo é uma mais-valia. Todavia, esta funcionalidade pode levar à inconsciente e perigosa atribuição de demasiada autoridade ao mesmo utilizador. Existe em muitas organizações a necessidade de definir funções que são mutuamente exclusivas, ou seja, o mesmo utilizador não pode desempenhar determinadas funções em simultâneo. Um exemplo prático são departamentos financeiros de algumas empresas, em que há um elemento que dá a ordem de compra dos produtos, e um outro gestor dá a autorização de pagamento. Esta separação de responsabilidades - Separation of Duty (SoD) - é uma forma de evitar, não só, a corrupção sem que haja pelo menos mais do que um elemento envolvido, bem como tomada de decisões de risco por erro (motivos principais para que em algumas organizações não seja adotada a hierarquia em árvore).

Outra desvantagem das funções com demasiado poder é serem grandes alvos de ataque. A adição de restrições à hierarquia de cargos no terceiro nível do modelo, Constrained RBAC, visa a resolução desta problemática, sendo um dos principais motivos para adoção deste modelo em alguns cenários. A implementação de restrições pode ser feita a dois níveis. O restringimento baseado no relacionamento utilizador-cargo, que é verificado no momento em que um cargo é atribuído ao utilizador se este não é mutuamente exclusivo com outro já atribuído, denominado de SoD estático. O SoD dinâmico consiste na verificação dos cargos que o utilizador activou e está a desempenhar no momento, permitindo que sejam atribuídos cargos apenas incompatíveis de exercer em simultâneo. Esta funcionalidade é usualmente implementada com uso de sessões, onde o utilizador no inicio de cada sessão seleciona os cargos que vai exercer num dado momento (modelo ilustrado na Figura 2.6).

2.1.4 Symetric RBAC

Finalmente no quarto nível, denominado de Symetric RBAC, é estabelecido como requisito o suporte à análise do relacionamento entre permissões e cargos. Para o suporte a esta funcionalidade o sistema dever ser capaz determinar quais os objetos que um determinado cargo ou utilizador tem acesso através das suas permissões. A consulta simétrica deve também ser implementada, i.e., dado um objeto saber quais os cargos e respetivos utilizadores que a ele tem acesso. Estas consultas têm de considerar não só as permissões que estão diretamente associadas a um cargo mas também todas as permissões que são herdadas dos cargos hierarquicamente subordinados. Em sistemas de larga escala, estas funcionalidades podem ser muito difíceis de

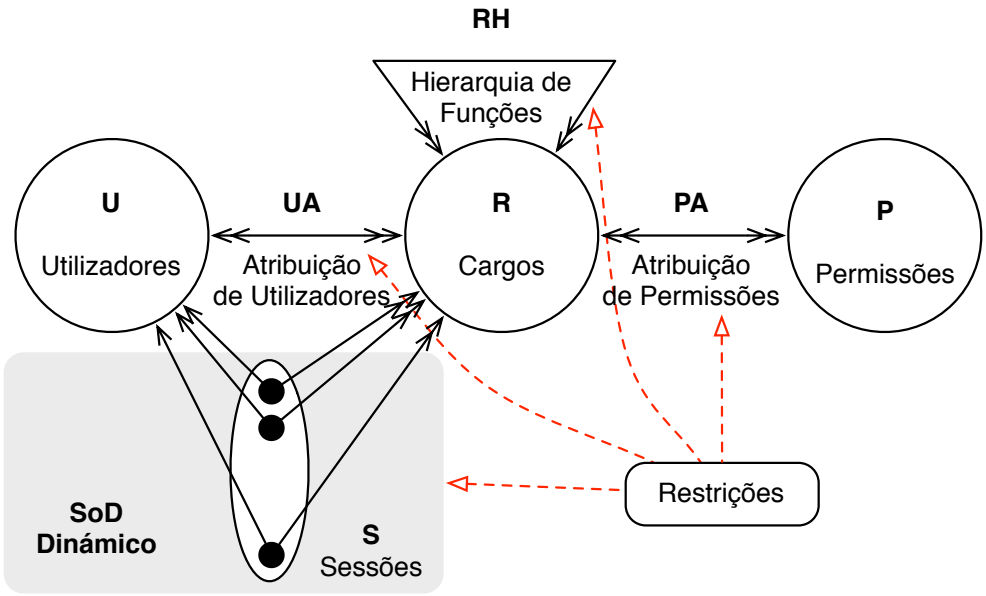


Figura 2.6: Modelo de Controlo de Acessos Baseado no Cargo com SoD (com base em [4]).

implementar, causa da sua inclusão apenas em sistemas de mais alto nível. Manter uma correta e apropriada relação entre permissões e cargos é fundamental para a eficácia de um sistema de controlo de acessos. Todavia, a administração do sistema sem estas funcionalidades tornam-se uma tarefa árdua particularmente em sistemas de grande dimensão. Com o passar do tempo as regras de acesso tendem a tornar-se obsoletas e impróprias. Por exemplo na mudança de responsabilidades de um cargo, é fundamental verificar se este não tem permissões desnecessárias para o desempenho da sua função. A revogação de algumas permissões e não mais atribuídas indica que estas não são mais necessárias para o exercício de nenhum cargo devendo ser eliminadas do sistema para sua eficiência. Conjuntamente, a entrada e saída de utilizadores no sistema, bem como a sua mudança de cargo, contribuem para a desatualização do sistema tornando estas funcionalidades essenciais.

2.2 Organization Based Access Control

Centrado nas organizações, este modelo é, em 2003, apresentado pelos seus autores como uma solução para as problemáticas até então não previstas pelos restantes modelos de acesso. Uma organização pode estar estruturada em sub-organizações, que têm as suas próprias políticas de acesso em diferentes contextos. A capacidade de modelar este tipo de cenários é a principal moti-

vação para a criação deste modelo de controlo de acessos baseada na organização - Organization Based Access Control (OrBAC) [7]. Os modelos até então propostos apenas permitem definir regras baseadas em permissões, sendo este o primeiro modelo a introduzir a especificação de regras que podem ser proibições, obrigações ou até mesmo recomendações, para além das habituais permissões.

Completamente parametrizadas pela entidade organização, as funções de relacionamento permitem criar políticas de controlo de acesso distintas para as diferentes organizações tornando a entidade *Organização* o centro deste modelo (ver Figura 2.7). Uma organização pode ser vista como um grupo de entidades ativas que cooperam entre si para atingirem um objetivo comum, i.e., por exemplo desde um departamento numa instituição a um pequeno grupo de trabalho associado a um projeto de uma empresa. Este modelo considera as seguintes entidades: Organization *(org)*, Subject *(s)*, Role *(r)*, Object *(o)*, View *(v)*, Action *(α)*, Activity *(a)* and Context *(c)* relacionadas entre si por um conjunto de doze funções.

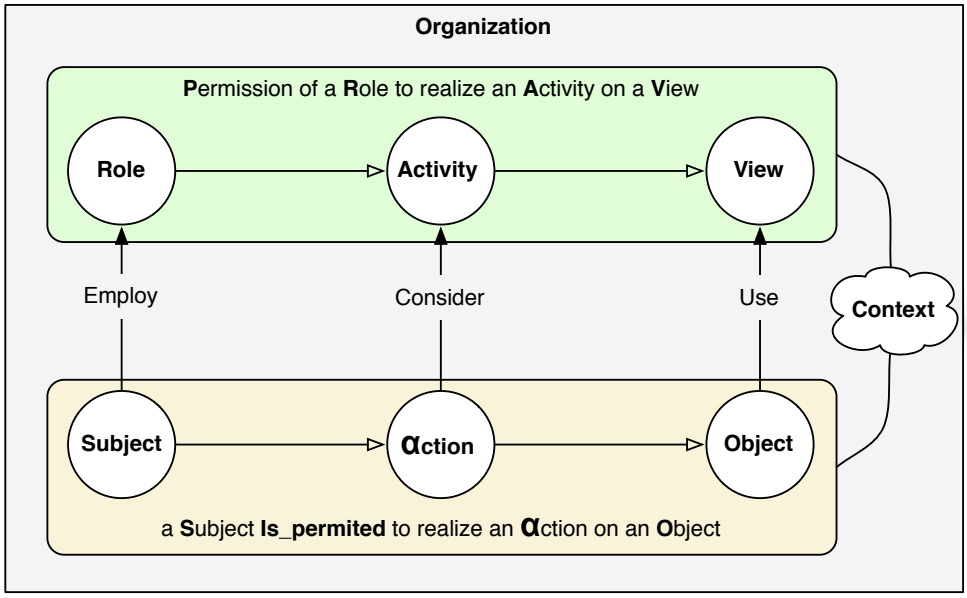


Figura 2.7: Diagrama do Modelo OrBAC.

A entidade *Subject* neste modelo pode ser um utilizador ou uma organização. A entidade *Role* é utilizada para relacionar os sujeitos com as organizações. *Departamento Financeiro* e *Projecto_alpha* são exemplos de Roles para Subjects que sejam sub-organizações de uma organização enquanto que Roles como *Programador*, *Investigador*, *Administrador* são para Subjects

que sejam utilizadores. Estas duas entidades são relacionadas através da relação *Employ(org, s, r)* que significa que a organização *org* emprega o sujeito *s* com o papel *r*.

Object é a entidade que representará objetos e recursos tais como ficheiros, arquivos, computadores, etc. Estes objetos que satisfazem uma propriedade comum ou usados em conjunto para um propósito são abstraídos e agregados em forma de *Views*. *Relatórios de vendas* pode ser uma View que englobe todos os relatórios de vendas de uma organização, incluídos na View *Relatórios_Fiscais* que contém todos os objetos de caráter financeiro. A relação *Use(org,o,v)* relaciona estas duas entidades com uma organização, significando que a organização *org* usa o objeto *o* na view *v*.

As entidades *Action* e *Activity* são usadas como forma de representar organizações que realizam a mesma atividade de formas alternativas. Por exemplo, a atividade *Consulta* numa organização pode ser a ação *Leitura* de um ficheiro enquanto que noutra organização pode ser um *Select* numa base de dados. Estas entidades são associadas pela relação *Consider(org,α,a)*, em que a organização *org* considera que a ação α faz parte da atividade *a*.

A entidade *Context* é uma das entidades que mais distingue o modelo OrBAC dos restantes modelos de controlo de acesso, pois foi introduzida no modelo para o tornar capaz de habilitar as organizações de garantir permissões num determinado contexto. Não diretamente relacionada com permissões, proibições, obrigações ou recomendações, os contextos são introduzidos como uma entidade que permite que um *subject* realize uma ação numa *view* caso o contexto se verifique. Estas entidades são relacionadas pela relação *Define(org,s,o,α,c)*. Para que se perceba melhor esta relação considere-se o seguinte exemplo: *Define(BancoRico, Finanças, conta_n2334, Consulta, Mandato Judicial)*, regra geral numa instituição bancária apenas os tutores das contas bancárias podem ter acesso às mesmas, contudo o banco *BancoRico* define que as *Finanças* podem consultar a conta *conta_n2334* sob a posse de um *mandato judicial*.

As permissões são criadas pela relação *Permission(org, r, v, a, c)* que define que a organização *org* garante ao papel *r* a permissão para realizar a atividade *a* na view *v* no contexto *c*. Contudo, neste modelo os utilizadores que exercem um determinado papel não herdam automaticamente os seus privilégios. Para que o utilizador adquira de facto este privilégio é necessário que o contexto seja válido. Assim sendo, a autorização em concreto não é garantida através das permissões. A relação que garante a um sujeito a autorização de realizar uma ação num objeto é *Is_permited(s,o,α)*, que é logicamente derivada da relação *Permission*, em conjunto com as

relações acima apresentadas, conforme descrito na secção 2.2.1. As proibições, obrigações e recomendações são, de forma análoga às permissões, criadas pelas relações *Prohibition(org, r, v, a, c)*, *Obligation(org, r, v, a, c)* e *Recomendation(org, r, v, a, c)* respetivamente.

2.2.1 Descrição formal e lógica do modelo OrBAC

A descrição formal deste modelo é definida pelos seus autores através de uma linguagem $\mathcal{L}$ de primeira ordem baseada na clássica Lógica de Primeira Ordem (LPO). Cada expressão em $\mathcal{L}$ conterá símbolos de um vocabulário finito que está dividido essencialmente em três grupos: constantes, relações e funções. As constantes correspondem às instâncias de entidades, havendo tantos tipos de entidades (θ) quantas as entidades, que neste caso é θ = *Organization, Subject, Object, Action, Role, View, Activity, Context*. As relações de $\mathcal{L}$, representadas por letras maiúsculas, são relações tipadas (não é aqui declarado quais os tipos de cada relação, pois é intuitivo após a leitura da descrição das entidades e respetivas relações).

Para cada atributo que uma entidade possa ter é assumido que existe uma função que recebe a entidade como parâmetro e retorna o valor ou conjunto de valores desse atributo. Idade(t) seria, por exemplo, a função de retornaria a idade do Subject *t*. Estas funções de extração das propriedades que caracterizam as entidades permitem que se possa estabelecer comparações, determinar interceções de atributos comuns, etc. Por exemplo, *Contactos_telefónicos(s1)* $\cap$ *Contactos_telefónicos(s2)* $\neq \emptyset$ significa que ambos os sujeitos têm pelo menos um contacto telefónico em comum. Os habituais operadores de lógica booleana são também adotados nesta linguagem. Para que se perceba melhor a usabilidade destes operadores considere-se os seguintes exemplos:

- $\forall s$*((Employ(ProjectoAlpha,s,Developer)* $\rightarrow$ *Employ(ProjectoAlpha,s,Project_Member))* – significa que na organização *ProjectoAlpha* quem é empregue como *Developer* também é automaticamente empregue como *Project_Member*.

- $\forall r \forall v \forall a$*(Permission(S.João,r,v,a,consulta)* $\rightarrow$ *Permission(S.João,r,v,a,urgência))* – significa que se a organização *S.João* dá permissão ao cargo *r* para realizar a atividade *a* na view *v* no contexto de *consulta* então também dá permissão no contexto *urgência*.

Os três axiomas apresentados abaixo são o cerne da lógica deste modelo. É através deles que são derivadas as autorizações e toda a política de acessos. O primeiro axioma descreve como as permissões entre cargos, views e atividades dão origem a autorizações concretas entre sujeitos, objetos e ações.

1. $\forall s \forall o \forall \alpha \forall r \forall v \forall a \forall c$
   $Permission(org, r, v, a, c) \vee$
   $Use(org, o, v) \vee$
   $Consider(org, \alpha, a) \vee$
   $Employ(org, s, r) \vee$
   $Define(org, s, o, \alpha, c) \rightarrow Is\_permitted(s, o, \alpha)$:
   se a organização *org* permite ao cargo *r* realizar a atividade *a* na view *v* sob o contexto *c*, usa o objeto *o* nessa mesma view *v*, considera que a ação α faz parte da atividade *a*, emprega o sujeito *s* com o cargo *r* e define que no contexto *c* esse mesmo sujeito *s* pode executar a ação α no objeto *o*, então o sujeito *s* tem permissão para executar a ação α sobre o objeto *o*.

2. $\forall r \forall v \forall a \forall c(Obligation(org, r, v, a, c)) \rightarrow Recommendation(org, r, v, a, c)$:
   todas as obrigações são automaticamente recomendações.

3. $\forall r \forall v \forall a \forall c(Recommendation(org, r, v, a, c)) \rightarrow Permission(org, r, v, a, c)$:
   cada recomendação é também uma permissão.

4. $\forall r \forall v \forall a \forall c(Permission(org, r, v, a, c)) \rightarrow \neg Prohibition(org, r, v, a, c)$:
   uma permissão implica a não proibição da mesma.

2.3 Role and Organization Based Access Control

Como se pode constatar na secção 2.1 o modelo RBAC foi pensado, e é aplicado, maioritariamente em cenários com uma lógica Organização-Funcionário (Business to Employee (B2E)). Contudo, há necessidade de modelar políticas de acesso em cenários Business to Business (B2B) e Business to Costumer (B2C) em que há hierarquia de organizações. A aplicação do modelo RBAC nestes cenários resultará na criação num enorme número de cargos conduzindo à ineficiência e degradação do sistema ao longo do tempo. Esta problemática de escalabilidade é argumentada como a principal motivação para a criação do modelo ROBAC [17] com base no modelo RBAC [6].

Neste modelo os utilizadores são associados ao par *(Cargo, Organização)* - (Role, Organization) - ao contrário de RBAC, em que são apenas relacionados com o cargo que desempenham dentro da organização. Outra mudança deste modelo em relação ao RBAC é as permissões serem definidas como a autorização de execução de operações sobre tipos de objetos em vez de objetos

em concreto. Se por um lado esta mudança evita a replicação de permissões para objetos do mesmo tipo, por outro pode ser desvantajoso pois nem sempre esta funcionalidade é desejável. Tal como em RBAC, as permissões são atribuídas aos cargos. Contudo, um utilizador pode executar uma ação sobre um objeto se e só se estiver atribuído a um par *(Cargo, Organização)*, o respetivo cargo tiver permissão para aceder ao tipo do objeto e a organização for detentora do mesmo.

À semelhança de RBAC96 [6], ROBAC está subdivido em quatro sub-modelos, ROBAC_0, ROBAC_1, ROBAC_2 e ROBAC_3. O ROBAC_0 é o modelo base, ROBAC_1 é igual ao ROBAC_0 mais hierarquia de cargos *(RH)* e organizações *(OH)*, ROBAC_2 é o ROBAC_0 mais restrições e ROBAC_3 é a combinação de ROBAC_1 e ROBAC_2. De seguida é apresentada a definição formal de cada um dos modelos.

ROBAC_0 tem os seguintes componentes:
• U – conjunto de Utilizadores;
• S – conjunto de Sessões;
• R – conjunto de Cargos;
• O – conjunto de Organizações;
• Op – conjunto de Operações;
• A – conjunto de Objetos
• At – conjunto de Tipos de Objetos;
• P $\subseteq$ Op $\times$ At – conjunto de Permissões;
• RO $\subseteq$ R $\times$ O – conjunto pares Cargo-Organização aplicáveis,*(r,o)*;
• $PA \subseteq P \times R$ – conjunto de relações Permissão-Cargo, de muitos para muitos;
• $UA \subseteq U \times RO$ – conjunto de relações entre Utilizadores e pares (Cargo,Organização), de muitos para muitos;
• $user : S \leftarrow U$ – função de associação entre uma sessão s_i e um único utilizador user(s_i);
• $atype : A \leftarrow At$ – função de associação entre um objeto e o seu tipo;
• $aorg : A \leftarrow O$ – função de associação de um Objeto e respetiva Organização a que pertence;
• $assigned\_role - orgs : U \leftarrow 2^{RO}$ – função que retorna o conjunto de pares *(r,o)* associados a um utilizador, $assigned\_role - orgs(u) = (r, o)|(u, (r, o)) \in UA$;
• $active\_role - orgs : S \leftarrow 2^{RO}$ – função que retorna os pares *(r,o)* activos numa dada sessão s_i, $active\_role - orgs(s_i) \subseteq assigned\_role - orgs(user(s_i))$;
• $can\_access(S, Op, A)$ – o predicado $can\_access(s, op, a)$ é verdadeiro se e só se $\exists(r, o) \in active\_role - orgs(s) \wedge aorg(a) = o \wedge ((op, atype(a)), r) \in PA$

$ROBAC_1$ acrescenta à anterior especificação de $ROBAC_0$ a hierarquia de organizações (OH) e hierarquia de cargos (RH). Estas duas funcionalidades implicam a mudança da definição da função *assigned_role-orgs* e do predicado *can_access*. Os seguintes itens são acrescentados à especificação do $ROBAC_0$ pelo modelo $ROBAC_1$:

• OH $\subseteq$ O$\times$ O – Conjunto de relações de ordem parcial entre as hierarquias das organizações;

• RH $\subseteq$ R$\times$ R – Conjunto de relações da hierarquia de cargos;

• assigned_role-orgs: U $\rightarrow 2^{RO}$ – função que retorna o conjunto de pares *(r,o)* redefinida como *assigned_role-orgs(u) = (r,o)* | $\exists r' \geq r \wedge \exists o' \geq o \wedge$*(u, (r',o')) $\in$ UA*;

• active_role-orgs: S $\rightarrow 2^{RO}$ – função de retorna os pares *(r,o)* redefinida como *active_role-orgs(s_i) $\subseteq$ assigned_role-orgs(user(s_i))*;

• can_access(S, Op, A) – o predicado can_access(s, op, a) é verdadeiro se e só se *(r,o) $\in$ active_role-orgs(s) $\wedge$ aorg(a) $\leq$ o $\wedge$ ($\exists$r' $\leq$ r, ((op, atype(a)), r') $\in$ PA)*;

O $ROBAC_2$ acrescenta ao modelo $ROBAC_0$ um conjunto de restrições que implementam SoD. Estas restrições, como habitual, são baseadas na atribuição de cargos incompatíveis aos utilizadores. O $ROBAC_2$ aplica as restrições de SoD a dois níveis, restrições globais e restrições locais. Uma restrição global é aplicada a todas as organizações, i.e., se *(r_i,r_j) $\in$ SoD* então os cargos r_i e r_j são mutuamente exclusivos, ou seja não podem ser desempenhados em simultâneo pelo mesmo utilizador dentro na mesma organização. A restrição local define que se *((r_i,o_f),(r_j,o_g)) $\in$ SOD* então os cargos r_i e r_j não podem ser associados ao mesmo utilizador em simultâneo nas organizações o_f e o_g respetivamente, ou seja os pares *((r_i,o_f)(r_j,o_g))* são mutuamente exclusivos impassíveis de serem atribuídos ao mesmo utilizador.

A agregação das funcionalidades acrescentadas pelos modelos $ROBAC_1$ e $ROBAC_2$ ao $ROBAC_0$ constituem o modelo completo final, $ROBAC_3$, ilustrado pelo diagrama da Figura 2.8.

2.4 Role-Organization Based Access Control

Apesar da complexidade do modelo anterior ROBAC, existe a necessidade de criar um novo modelo mais versátil e adequado às necessidades deste trabalho. Neste sentido, inspirado no modelo ROBAC apresentado na secção 2.3, é criado e apresentado nesta secção um novo modelo completamente focado nas entidades Cargo e Organização (ver Figura 2.9). Ou seja, neste novo modelo as permissões são associadas ao par (Cargo,Organização), tal como os utilizadores. A

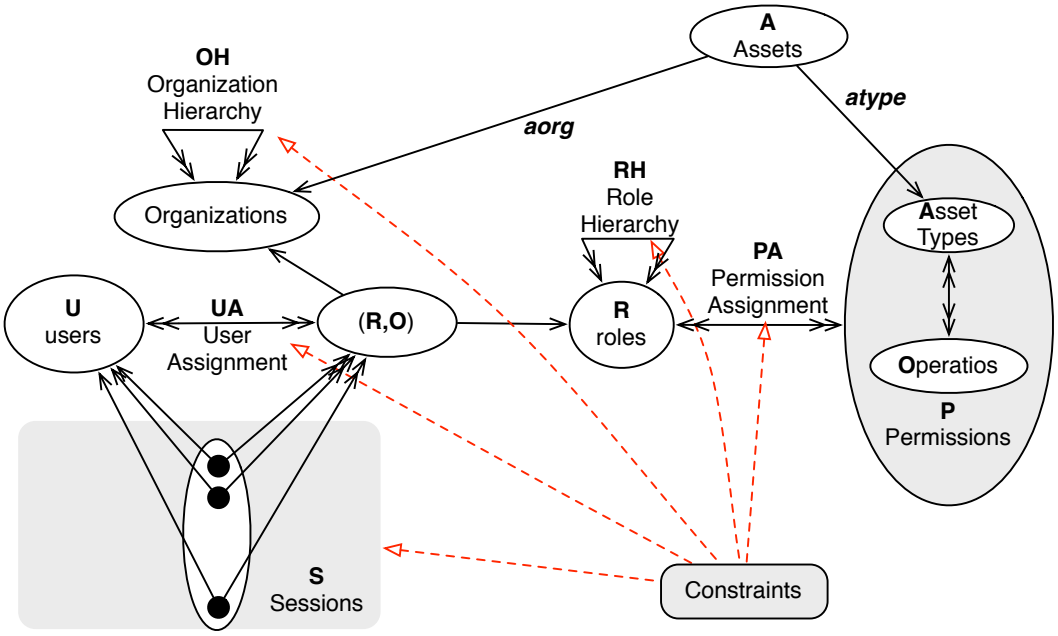


Figura 2.8: Diagrama do Modelo ROBAC (com base em [7]).

ideia principal é que um utilizador poderá desempenhar um ou mais cargos numa ou mais organizações. Cargos estes que podem ter permissões distintas de acordo com a política de acessos de cada organização ou sub-organização, ao contrário do modelo anterior em que os Cargos tem exatamente as mesmas permissões independentemente da organização.

Este modelo é constituído por quatro entidades principais: *User, Role, Organization, Permission*. *User* representa um utilizador do sistema, que tanto pode ser um ser humano bem como um agente ou sistema externo. *Role* pode ser um Cargo/Função a ser desempenhados dentro de uma organização por utilizadores. À semelhança do modelo OrBAC, apresentado na secção 2.2, uma organização pode ser vista como um grupo de entidades ativas que cooperam entre si para atingir um objetivo comum, i.e., por exemplo empresas, instituições, sub-organizações, projetos, ou até mesmo organizações virtuais (Virtual Organization (VO)) que representam um grupo de utilizadores. *Permission* simboliza uma permissão ou privilégio em geral, podendo ser permissões quer de acesso lógico, quer de acesso físico. Estas permissões podem ser modeladas de diversas formas dependendo do seu objetivo. Por exemplo, uma permissão tanto pode ser um perfil de acesso físico a um espaço físico (laboratório, escritórios, etc.), como pode representar uma ação ou evento que pode ser executado numa aplicação ou objeto.

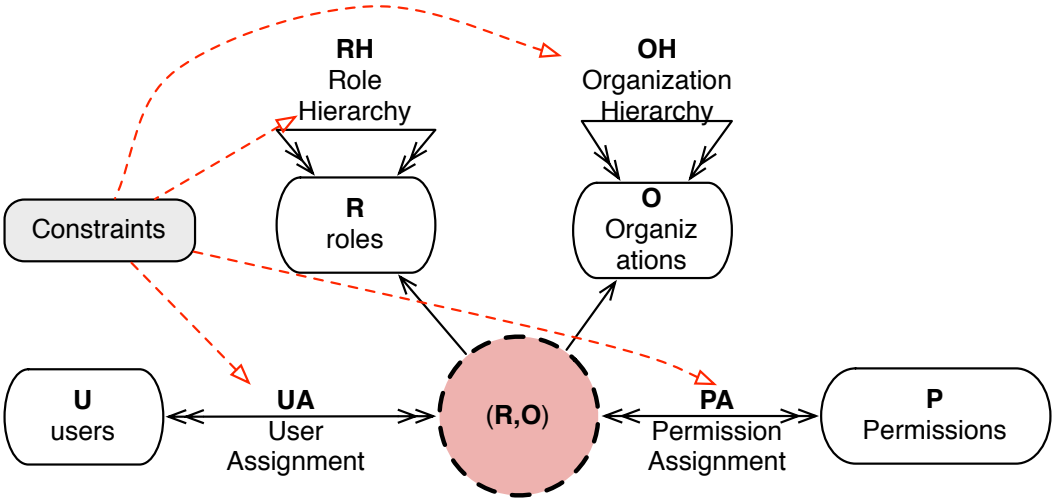


Figura 2.9: Diagrama do Modelo (R-O)BAC.

Este novo modelo, à semelhança de ROBAC, contém os seguintes conjuntos:

• U – conjunto de Utilizadores;

• R – conjunto de Cargos;

• O – conjunto de Organizações;

• P – conjunto de Permissões;

• $RH \subseteq R \times R$ – conjunto de Hierarquias de Cargos, em que o par *(r_s, r_j)* significa que o cargo r_s é super cargo de r_j;

• $OH \subseteq O \times O$ – conjunto de Hierarquias de Organizações, em que o par *(o_s, o_j)* significa que a organização o_s é super organização de o_j;

• $RO \subseteq R \times O$ – conjunto pares Cargo,Organização aplicáveis,*(r,o)*;

• $PA \subseteq RO \times P$ – conjunto de associações (Cargo,Organização)-Permissão, de muitos para muitos;

• $UA \subseteq U \times RO$ – conjunto de associações entre Utilizadores e pares (Cargo,Organização), de muitos para muitos;

• $Constraints$ – conjunto de restrições. Pares de cargos que não pode ser exercícios em simultâneo pelo mesmo utilizador, ou permissões que não podem ser atribuídas ao mesmo utilizador.

A hierarquia de cargos neste modelo é transversal a todas as organizações. Caso o cargo r_s seja especificado em RH como supra-cargo de r_j, então os utilizadores associados ao cargo r_s numa organização o herdaram as permissões também associadas ao cargo r_j em o e respetivas sub-organizações.

A funcionalidade da hierarquia de Organizações aliada à propriedade das permissões serem associadas ao par Cargo-Organização torna este modelo muito versátil e vantajoso em relação aos anteriores. Ao determinar que a organização o_j é sub-organização de o_s, todas as permissões associadas a o_j passam a ser válidas em o_s nos respetivos cargos. Os utilizadores que estiverem associados a o_s passarão automaticamente a ter acesso à organização o_j segundo a sua política de acessos. Desta forma, sempre que uma super-organização necessita de adicionar mais permissões aos cargos, além das que já estão associadas às sub-organizações, basta apenas associar essas permissões aos respetivos cargos. Por exemplo $PA \leftarrow (r_1, o_s, p_1)$, a permissão p_1 passará a ser válida para o cargo r_1 na organização o_s, mas p_1 não será válido em o_j para os seus utilizadores uma vez que as permissões não descem na hierarquia.

Tirando proveito desta herança de permissões entre as organizações, torna-se fácil para uma organização estruturada em várias sub-organizações ou departamentos, criar uma política de acessos geral. Basta para isso, por em evidencia as permissões que devem ser comuns a todos os departamentos, associa-las aos respetivos cargos numa organização virtual (VO) e posteriormente torná-la sub-organização dos vários departamentos.

Imagine-se o seguinte exemplo: regra geral todos os alunos da universidade podem aceder aos complexos pedagógicos das 8h às 20h. No entanto, os estudantes do departamento de biologia tem a necessidade de ir aos laboratórios também aos fins de semana para medir a evolução de bactérias diariamente. A biblioteca também tem um horário de funcionamento mais alargado, das 9h às 24h durante os dias úteis, em quanto que a cantina está apenas aberta das 11h às 22h. A melhor forma de modelar este cenário é criar uma organização virtual para representar a política de acessos geral da universidade. Nesta VO, denominada de *Política de acessos base*, associa-se o perfil de acesso das *8h-20h de Segunda a Sexta* ao cargo *Estudante*. Nas outras organizações, *D. Biologia*, *Biblioteca* e *Cantina* associam-se os restantes perfis de acesso que se adaptam às suas necessidades em particular, conforme ilustrado na Figura 2.10. Assim, caso seja alterado o horário normal de funcionamento (das 8h às 20h de segunda-feira a sexta-feira), através da hierarquia de organizações será automaticamente válido em todos o complexos pedagógicos que são super-organização de *Política de acessos base*.

A implementação dos modelos anteriores, RBAC e ROBAC, neste tipo de cenários levaria à criação de cargos secundários para distinguir os vários departamentos. No exemplo da Figura 2.10 seriam criados adicionalmente os cargos *Estundante_BGUM, Estundante_Cantina e Estudante_Biologia* para espelhar as diferenças de horários acesso. Esta abordagem tem duas

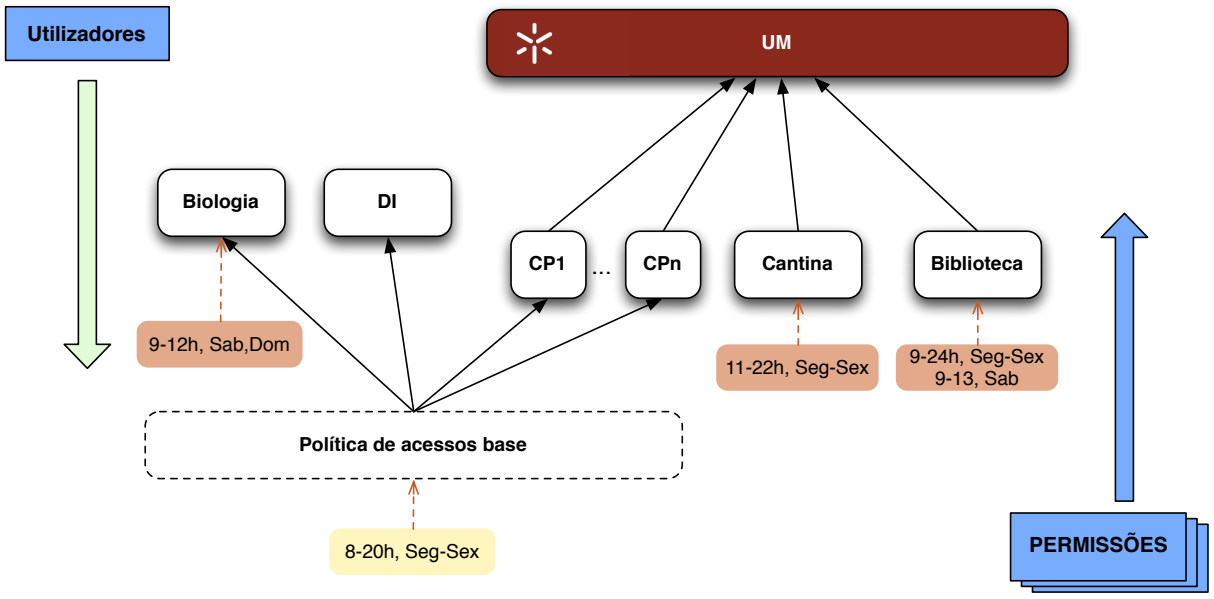


Figura 2.10: Política de Acessos Sub-estruturada.

grandes desvantagens: a primeira é a criação de cargos adicionais, que a longo prazo leva uma degradação de performance do sistema; a segunda é a dificuldade acrescida de administração e manutenção do sistema, podendo por vezes dar aso a erros inconscientes. Como se pode observar, seria necessário criar pelo menos um Cargo adicional por caga par Cargo-Organização que se diferencie. Num conjunto de O organizações e R cargos este novo modelo pode reduzir o número de cargos até $O \times R - 1$.

2.5 Sumário

Neste capítulo foram abordados os modelos de acesso RBAC, OrBAC e ROBAC, que desempenharam um papel importante neste trabalho contribuindo para uma assimilação de conceitos na área dos controlos de acessos. Por último e mais importante, foi apresentado o novo modelo de acessos baseado no par Cargo-Organização, criado no âmbito deste trabalho. Este novo modelo é implementado e aplicado a um cenário real, apresentado na secção 4.1, para o controlo quer de acessos físicos quer de acessos lógicos, tal como posteriormente descrito nas secções 4.3 e 4.4 do capítulo 4.

Capítulo 3

Tecnologias de Controlo de Acesso Físico

Após apresentação do modelo de acessos a utilizar neste trabalho, neste capítulo serão apresentadas as três principais tecnologias de autenticação física estudadas: Cartão do Cidadão, Radio Frequency Identification (RFID) e Biometrias. Será efetuada uma breve descrição do estado atual de cada tecnologia bem como das principais normas a elas associadas. No fim deste capítulo é apresentado um resumo justificativo da escolha da tecnologia adotada no desenvolvimento do sistema proposto.

3.1 Cartão do Cidadão

Atualmente ainda em fase de implementação em Portugal, o novo Cartão do Cidadão (CC) possui duas vertentes, uma como documento físico e outra como tecnológico. Em formato de um smart card, como documento físico, permite ao cidadão *"provar a sua identidade perante terceiros através da leitura de elementos visíveis, coadjuvada pela leitura óptica de uma zona específica"* [18]. Como documento eletrónico, o CC permite a identificação do cidadão perante serviços informatizados. Além disto, através de assinatura digital, o CC também permite autenticar de forma unívoca documentos eletrónicos com a mesma validade jurídica de um documento assinado à mão, em que *"Nenhuma autoridade ou entidade pública ou privada pode recusar o valor probatório da assinatura electrónica [...]"* [18].

Nas faces do CC, para além de informação de identificação pessoal, estão presentes alguns mecanismos de segurança física de verificação rápida e fácil. Na face frontal, ilustrada na Figura 3.1, estão presentes quatro elementos de segurança: um micro relevo em Brialle; a fotografia do titular gravada em *Multiple Laser Image* (MLI), visível com variação angular, onde contém

também os três últimos caracteres do número do documento; um elemento difractivo opticamente variável (Diffractive Optically Variable Image Device (DOVID)) sobre a fotografa do titular que contém bandeira nacional, observável com a rotação do documento pela ocorrência de uma mudança cromática de verde para vermelho. Toda a face frontal é ainda gravada em tinta opticamente variável. No verso do CC é também acrescentado um feixe holográfico em DOVID.

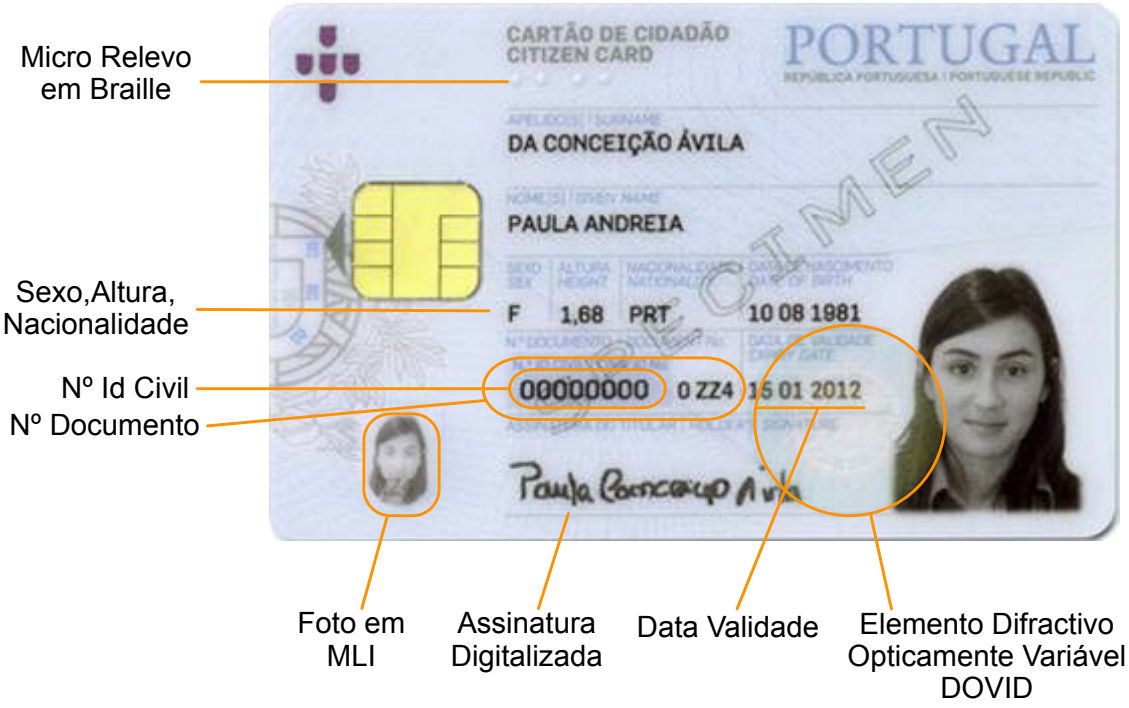


Figura 3.1: Face frontal do Cartão do Cidadão

No chip do cartão, para além da informação habitual contida num cartão de identificação (nome, data nascimento, nacionalidade, sexo, etc...), estão armazenados certificados digitais necessários à autenticação e assinatura digital, cujo o acesso está protegido por códigos Personal Identification Number (PIN) (ver Figura 3.2). Este chip está dotado de uma camada de *software middleware* que possui três aplicações e três interfaces criptográficas que permitem interagir com a informação presente no CC. As aplicações presentes no CC tem as seguintes funcionalidades:

- IAS - aplicação responsável pelas operações de autenticação e assinatura digital de documentos;
- EMV-CAP - aplicação responsável pela geração de palavras-chave únicas por canais alternativos (por exemplo telefone);

- *Match-on-card* - Criada essencialmente para as autoridades judiciais e policiais para procedimentos de elevado nível de segurança, *Match-on-card* é um mecanismo de verificação da titularidade do CC através da verificação da impressão digital. Uma vez conservadas as minúsculas das impressões digitais no *chip*, esta aplicação permite a sua comparação com outras lidas por um sistema externo sem que estas sejam enviadas para o exterior do *chip*.

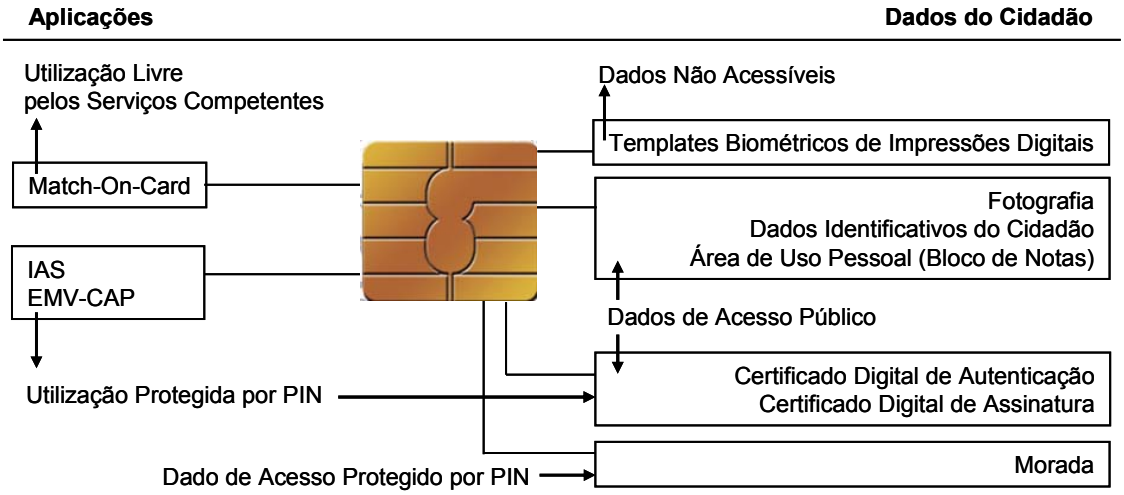


Figura 3.2: Informação e aplicações residentes no chip do CC (fonte: [18]).

As três interfaces presentes no *middleware* do CC, CryptoAPI [19], PKCS#11 [20] e eID [21] são orientadas ao desenvolvimento de novas aplicações que usem o Cartão do Cidadão. As duas primeiras são APIs padrão para operações criptográficas, enquanto que eID, não padrão, é mais orientada à leitura dos dados identificativos do cidadão não contendo funcionalidades criptográficas.

O Cryptographic Service Provider (CSP) implementa operações criptográficas para aplicações standard Microsoft, como por exemplo Office ou Outlook. A invocação das operações criptográficas implementadas pelo CSP é efetuada através da interface CryptoAPI. A implementação do CSP utiliza a interface PKCS#11 conforme ilustrado na Figura 3.3. O Microsoft® Cryptographic API 2.0 (CryptoAPI) permite aos programadores adicionarem funcionalidades de autenticação, encoding e encriptação às suas aplicações baseadas em Win32®.

O PKCS#11 é a principal interface criptográfica do *middleware*, pois é a interface que serve de suporte às outras interfaces de mais alto nível (ver Figura 3.3). Esta interface para além de ser bastante adotada para o acesso a *smart cards* também é utilizada por aplicações multi-plataforma que necessitam de utilizar standards, como por exemplo o Mozzila Firefox, OpenSSL, OpenVPN, OpenSSH entre outros. O PKCS#11 (PKCS v2.20) define uma API para criar e trabalhar com os mais comuns objetos criptográficos tais como certificados X.509, chaves RSA, DES, etc.

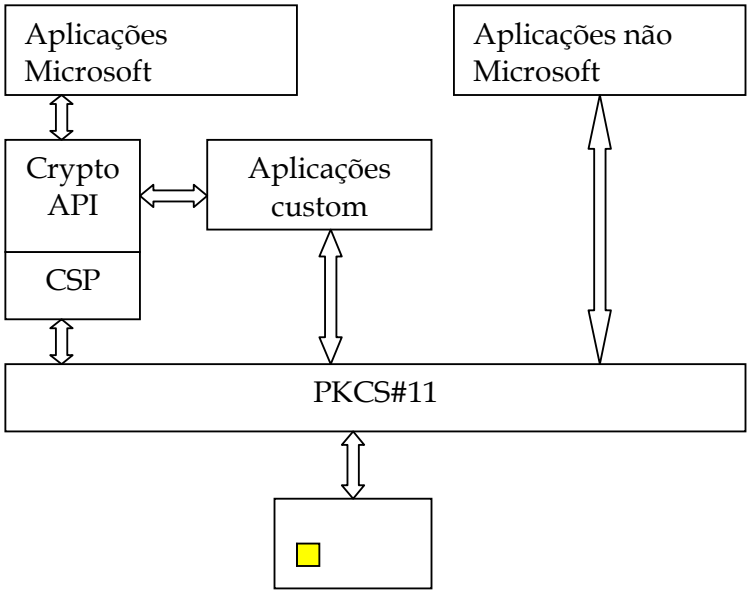


Figura 3.3: Arquitectura das interfaces criptográficas presentes no CC (fonte: [18]).

O eID é um Software Development Kit (SDK) que está destinado ao desenvolvimento de aplicações que apenas necessitam de aceder a dados identificativos presentes no cartão e não necessitem de utilizar operações criptográficas. A gestão das várias versões do cartão são automaticamente abstraídas e geridas pela SDK. Este kit de desenvolvimento é fornecido como uma interface C/C++ ou ainda com dois *wrappers* para as linguagens Java e C#.

3.2 Radio Frequency Identification

Normalizada pela Automatic Identification (Auto-ID), EPC-Global e International Organization for Standardization (ISO), o Radio Frequency Identification (RFID) denota-se como a

tecnologia de identificação unívoca sem contacto físico nem visual do identificando [22, 23]. Inicialmente utilizada apenas em processos fechados para a identificação de produtos (e.g. ciclo de produção, registo de livros, cadeias de distribuição, controlos de manutenção periódicos) [22] a tecnologia RFID revelou-se uma excelente solução para autenticação e identificação unívoca de produtos. Desta forma, esta tecnologia assume uma firme posição no que diz respeito à identificação. Para além da identificação de objetos, atualmente a tecnologia RFID é utilizada em diversas aplicações para a identificação de pessoas, como por exemplo nos E-Passports [24].

A arquitetura física da tecnologia RFID é baseada essencialmente em duas componentes, etiquetas e leitores (*tags* e *readers*): os leitores são antenas que lêem e escrevem informação nas *tags* por meio de ondas eletromagnéticas; as *tags*, na forma mais simples, podem ser constituídas apenas por um *chip* e uma antena.

3.2.1 Tags RFID

Com o objetivo de ser uma tecnologia de custo muito reduzido (menos de 30 cêntimos por *tag*), as etiquetas RFID mais simples são constituídas apenas por uma antena e um *chip*. Todas as tags RFID contém alojado no *chip* um número de série único, normalmente denominado de *tagID*, que é emitido pela tag quando intercetada pelos leitores. As *tags* desta classe são as mais utilizadas, nomeadamente em etiquetas adesivas para identificação de produtos e em sistemas anti-roubo. Outra aplicação deste tipo de *tags* são os cartões de proximidade e *smart cards* em sistemas controlo de acesso. Em alternativa, há classes de *tags* mais completas, que contêm unidades memória regravável, capazes de armazenar informação adicional. Estas tags, mais versáteis e de custo mais elevado, são utilizadas para registo de ciclo de vida de produtos e processos de manutenção, por exemplo em viaturas, aviões, etc.

Construídas e embebidas de diversas formas, as tags RFID, são agrupadas em três tipos:

- **tags passivas** – são *tags* que não contém nenhuma autonomia energética. Assim, apenas conseguem enviar a sua informação aos leitores RFID após terem recebido e transformado a energia que receberam através das ondas eletromagnéticas emitidas pelos leitores RFID, quando são colocadas ao alcance deles.

- **tags ativas** – como nome indica, estas tags permanecem ativas/acordadas, caracterizadas pela sua autonomia energética contendo uma bateria interna para o efeito. As *tags ativas* iniciam a comunicação imediatamente após a deteção de um leitor RFID, acelerando o

processo de comunicação. Desta forma, ao contrário das *tags passivas*, não necessitam de ser abastecidas de energia para responder ao leitor.

- **tags semi-passivas** – estas *tags* também contém uma bateria interna para aumento do seu alcance e rapidez na comunicação. Contudo, permanecem adormecidas até serem acordadas pelos leitores para iniciarem a comunicação. Este tipo de *tags* é normalmente usado em sistemas de maior alcance que usa uma gama de frequências mais alta e necessitam de mais energia.

A EPCglobal, que tem como principal objetivo a monitorização do trajeto de objetos em cadeias de distribuição, apresenta algumas normas para tags RFID. A diferença mais notória, em relação às tags convencionais que contém um *tag ID*, é a inclusão do Electronic Product Code (EPC), que visa substituir os atuais códigos de barras. Estas normas estão divididas em 4 classes, com sucessivo incremento de funcionalidades entre si [25].

- ***Class-1: Indentity Tags*** – tags passivas que contém as seguintes funcionalidades: um EPC, um *tag ID*, uma funcionalidade que permita desativar a tag, ou seja, torna-la não responsiva aos leitores. Como opção podem ser implementadas três funcionalidades extra: ativação e desativação temporária da tag, acesso protegido por *password* e uma unidade de memória complementar.

- ***Class-2: Higher-Functionality Tags*** – Para além dos requisitos da *Class-1*, as tags desta classe devem implementar autenticação no controlo de acesso aos dados da tag. O *tag ID* e a memória complementar devem ser de maior capacidade.

- ***Class-3: Semi-Passive Tags in UHF Gen2*** – Esta classe acrescenta às anteriores o uso de uma bateria para alimentação da tag e/ou sensores nela contidos. No entanto, tags *Class-3* continuam a comunicar passivamente, ou seja, apenas enviam a sua informação só após serem despertadas pelos interrogadores.

- ***Class-4: Active Tags*** – as tags desta classe, acrescentam às funcionalidades das classes anteriores autonomia completa. Desta forma, tags *Class-4* podem iniciar a comunicação com os interrogadores, ou até mesmo com outras tags, sem que tenham de ser ativadas pelos interrogadores. Estas tags não devem interferir com os protocolos de comunicação usados pelas tags das classes 1,2 e 3.

3.2.2 Antenas RFID

As antenas RFID são a parte integrante dos leitores RFID que geram ondas radioelétricas, através dos quais recebem e enviam os dados para as *tags*. Esta comunicação entre as antenas e as *tags* pode ser efetuada sem necessidade de linha de vista para a *tag*. Contudo, certas condições desfavoráveis podem criar problemas de comunicação tais como metais, líquidos ou até mesmo a densidade de humidade na atmosfera, reduzindo alcance da comunicação. Na tabela 3.1 são apresentadas as várias gamas de frequências RFID e respetivas aplicações [26].

Um fator importante para o desempenho da antena, para além da frequência, é a sua forma, alcance e energia transmitida. A energia transmitida é fundamental para as *tags* passivas, pois usam essa mesma energia para se ativarem automaticamente e enviar a resposta de volta.

Tabela 3.1: Tabela de gamas de Frequências RFID

Frequência RFID	**Cenário de Uso**
125 KHz (LF)	Padrão mundial, extremamente utilizado em *tags* de baixo custo, largamente usado na identificação de produtos e animais. Também usada nos cartões de proximidade da serie EM-4000 para controlos de acesso bem como no E-Passport.
13.56 MHz (HF)	Padrão mundial usada nos *smart cards*, também em tags de baixo custo para a identificação de objetos e pessoas. Alternativa aos 125KHz para maiores distancias.
400 MHz	Usado ocasionalmente em comandos de sistemas de fecho central de viaturas.
868 MHz (UHF)	Frequência padrão na Europa para tags ativas e passivas. Usado em processos logísticos, por exemplo ciclos de produção e manutenção de produtos.
915 MHz (UHF)	Sistema análogo aos 868MHz mas usado nos Estados Unidos da America. Normalmente os leitores trabalham na gama de frequências 850 - 950Mhz, sendo também compatíveis com a norma Europeia.
2.45 GHz	Frequência globalmente reservava para uso Industrial, Scientific and Medical (ISM) que não necessita de licença ou reserva. Usado em transmissores como por exemplo sensores de temperatura, ou localização GPS.

3.2.3 Normas e padrões RFID

As organizações Auto-ID, EPC-Global e ISO são aquelas que, lado a lado, mais tem contribuído para a normalização e desenvolvimento da tecnologia RFID e sua aplicabilidade. Embora em alguns aspetos estas entidades se complementem, algumas normas por elas criadas estão sobrepostas, levando a um entrave no arranque inicial da tecnologia. A ISO, como habitual, criou especificações de mais baixo nível, tais como protocolos de comunicação entre *tags* e leitores (ver Tabela 3.2). A Auto-ID é um grupo formado por vários laboratórios de investigação académica na área das redes RFID. A Auto-ID foi uma das primeiras entidades a propor normas nomeadamente para as *tags*. A EPC-Global é uma organização, que em cooperação com a Auto-ID, projetou, normalizou e desenvolveu uma arquitetura RFID para a gestão de cadeias de distribuição. Esta arquitetura, verticalmente elaborada pela EPC-Global, conforme ilustrado na Figura 3.4, contém deste o protocolo de comunicação com as *tags* até ao sistema de localização dos objetos (*Object Name Service (ONS)*, semelhante ao DNS).

Tabela 3.2: Normas ISO

Norma	**Descrição**
ISO 11784	Define a estrutura de dados nas *tags*.
ISO 11785	Protocolo de comunicação sem fios.
ISO 14443	Norma para cartões de identificação sem contacto na gama dos 13.56MHz, em particular *Smart Cards*. Este *standard* abrange desde o formato físico até ao protocolo de transmissão de dados entre os cartões e os leitores.
ISO 15693	Protocolo de comunicação sem fios par cartões de pagamento.
ISO 18047	Para testes de conformidade das *tags* RFID com as normas.
ISO 18046	Para efetuar testes de performance às *tags* e leitores RFID.
ISO 18000–1	Parâmetros genéricos para interfaces sem fios nas frequências globais.
ISO 18000–2	Protocolo sem fios para os 135 KHz.
ISO 18000–3	Protocolo sem fios para os 13.56 MHz.
ISO 18000–4	Protocolo sem fios para os 2.45 GHz.
ISO 18000–5	Protocolo sem fios para os 5.8 GHz.
ISO 18000–6	Protocolo sem fios para os 860-930 MHz.
ISO 18000–7	Protocolo sem fios para os 433.92 MHz.

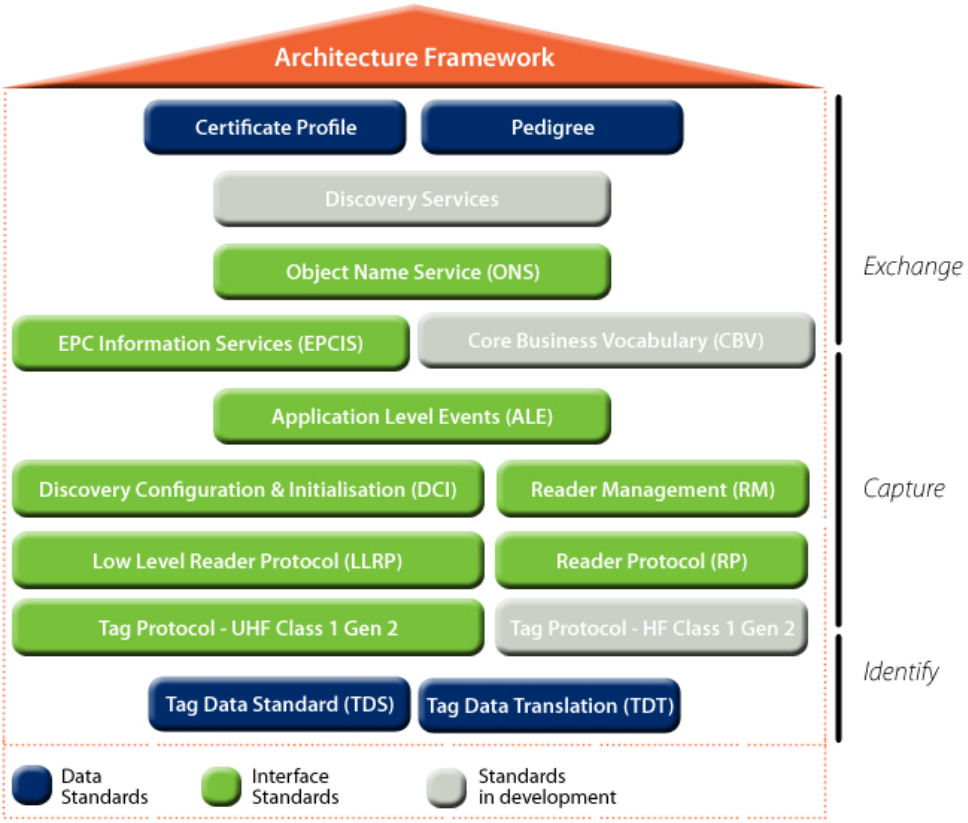


Figura 3.4: Arquitectura EPC-Global para gestão de redes de distribuição (fonte: http://www.epcglobalinc.org/standards)

3.3 Biometrias

As biometrias, devido à sua natureza de unicidade, são uma forte fonte de autenticação do indivíduo, acrescidas da vantagem de serem intransmissíveis. O reconhecimento de uma pessoa através das suas biometrias pode ser efetuado através da mais simples leitura da sua impressão digital ou da mais completa análise das várias biometrias do indivíduo. No limite, pode ser feita uma avaliação do seu perfil de comportamento. Devido à complexidade das biometrias (impressões digitais, facial, íris, fala, etc.), o seu reconhecimento com 100% de certeza é um processo custoso e muito demorado. No entanto, as tecnologias de leitura biométrica (especialmente para controlos de acesso) efetuam reconhecimentos com alguma margem de erro em prol de o fazerem em tempo útil, conduzindo assim a possíveis erros no reconhecimento do indivíduo. Uma desvantagem da adoção singular desta tecnologia é o facto de que nem todas pessoas são portadoras de algumas das suas originais biometrias. Por exemplo nos Estados Unidos da América foi

publicado que aproximadamente dois por cento da população já não possuía impressões digitais pelos mais diversos motivos (acidente, deficiência, etc) [27]. Em consequência, sistemas de autenticação multi-modal tem vindo a ser propostos. Estes sistemas consistem no reconhecimento de mais do que uma biometria em simultâneo, aumentando assim o fator segurança no reconhecimento sem aumentar linearmente o tempo de resposta dado que os reconhecimentos parcelares são efetuados em simultâneo.

Um sistema biométrico pode operar essencialmente em dois modos:

- Verificação – determina se a pessoa é quem diz ser. Compara a amostra da pessoa capturada no momento com a respetiva amostra referência na base de dados. Estas amostras de referência denominam-se de *templates*.
- Identificação – identifica a pessoa. Compara a amostra da pessoa com todas as amostras na base de dados até encontrar uma igual (ou parecida dependendo do nível de certeza pretendido).

Os sistemas biométricos são compostos por um conjunto de componentes, cuja complexidade depende do número e das propriedades das biometrias com que trabalham [28]. Estes componentes são os seguintes:

- **Captura** – É o processo de captura da amostra biométrica do utilizador. Este processo envolve o uso de sensores, dependendo do tipo de biometria (por exemplo, sensores óticos para impressões ditais, câmaras para reconhecimento facial, etc.).
- **Extração** – Este componente usa a amostra obtida na fase de captura e extrai a informação biométrica nela contida. O resultado deste processo é denominado como vetor de características.
- **Armazenamento** – Este elemento serve para armazenar os vetores de características biométricas. A sua complexidade depende do tipo de sistema, verificação ou identificação.
- **Classificação** – Este processo compara as duas amostras: a original armazenada no sistema e a nova amostra recolhida no momento. O resultado desta operação é um valor de 0 a 1 que indica o quanto estas duas amostras são iguais.
- **Decisão** – Este componente é responsável por tomar a decisão de permitir ou negar o acesso ao utilizador. Esta decisão é tomada com base num valor limite (por exemplo, 0.98

para amostras que tenham 98% de igualdade à amostra na base de dados). Caso a amostra seja igual ou superior ao limite, então o acesso é permitido. Caso seja inferior é rejeitado.

A Figura 3.5 ilustra o fluxo de comunicação entres o vários componentes num processo de autenticação.

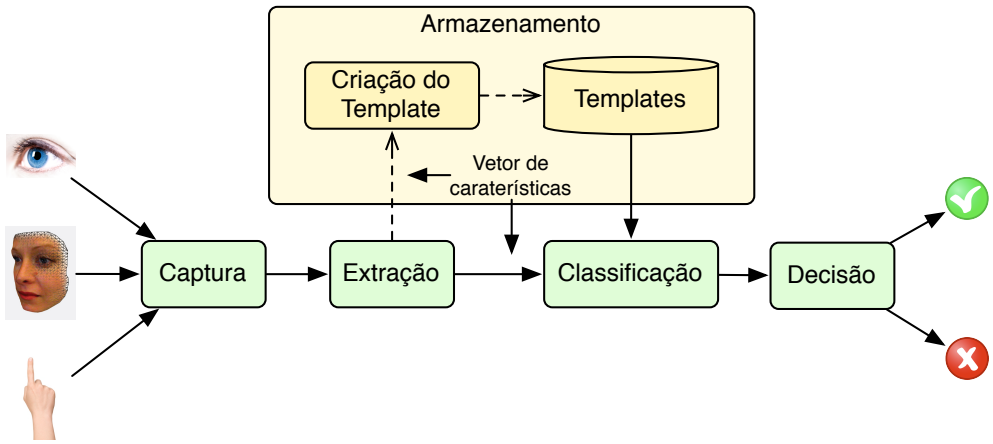


Figura 3.5: Fluxo de comunicação entre os componentes biométricos.

3.4 Sumário

Neste capítulo foram abordadas as tecnologias Cartão do Cidadão, RFID e Biometrias, inicialmente estabelecidas como alvo de estudo para uso no controlo de acessos físicos. O uso do Cartão do Cidadão não se revelou ser uma boa alternativa, pois à falta de outro mecanismo de autenticação alternativo, o sistema apenas poderia ser usado por utilizadores portadores do cartão do cidadão português.

A tecnologia escolhida foi a RFID. O que conduziu à adoção da RFID foram dois fatores: baixo custo e versatilidade. Os leitores RFID tem um custo significativamente mais baixo que os leitores de biometrias. Em especial se considerarmos leitores biométricos de uso exterior, resistes ao frio e à chuva. Para além do baixo custo, a RFID revelou ser mais versátil quer em relação em relação às biometrias quer ao Cartão do Cidadão. Dado que a RFID é uma tecnologia sem fios, permitiu que o leitor fosse colocado do lado de dentro da porta, não ficando assim exposto a atos de vandalismo nem condições climatéricas adversas, assegurando uma maior fiabilidade do

sistema. Uma vez tomada esta decisão pela direção da empresa, não foram mais aprofundados os estudos sobre as biometrias, prosseguindo-se assim para a implementação do módulo de controlo de acessos físicos.

Capítulo 4

Desenvolvimento do Sistema

Neste capítulo, para melhor compreensão e justificação de algumas opções tomadas, será exposto o cenário real onde o sistema baseado no modelo ***Role-Organization Based Access Control****, introduzido na Secção 2.4, foi desenvolvido e implementado. De seguida serão descritas as tecnologias utilizadas no desenvolvimento do trabalho, a estrutura do modelo de dados utilizado e a arquitetura da aplicação desenvolvida. Por fim, é descrita com detalhe a forma como foram implementados os módulos de controlo de acessos físicos e lógicos integrados na aplicação desenvolvida.*

4.1 Contexto do Trabalho na Empresa

Em prol de uma melhor compreensão e contextualização do problema, importa referir que este trabalho foi implementado no âmbito da implementação de um Sistema de Gestão Integrado de acordo com as normas ISO 9001 [13] e NP 4457 [14]. Numa primeira instância, é pretendido que o sistema, de forma automática, seja capaz de controlar o acesso físico dos colaboradores às infraestruturas da organização. Atualmente a empresa conta com instalações em duas cidades e com perspetivas de futura expansão.

No contexto da empresa, um conjunto de ferramentas *open source* é utilizado para suporte e auxílio no desenvolvimento de software, bem como na partilha e difusão de conhecimento e novas ideias dentro da organização. Para o controlo de versões de código fonte, bem como de outros arquivos digitais, é utilizado o *Apache Subversion* [29], mais conhecido como SVN. Na manutenção e administração de projetos é utilizada a ferramenta Redmine, utilizada também como repositório de conhecimento técnico e partilha interna de ideias. Apesar de ainda em fase

de melhoria e enriquecimento de funcionalidades pela comunidade *open source*, a Redmine [30] é uma ferramenta *web* baseada na *framework Ruby on Rails*, que já oferece bastantes funcionalidades no que respeita à gestão de projetos. A política de controlo de acessos interna do Redmine é baseada no cargo (RBAC).

É também utilizado na empresa um serviço de diretoria para efeito de autenticação e autorização dos utilizadores de alguns serviços disponibilizados pela organização (e.g. Internet, SVN, SAMBA, Redmine, Intranet, Impressoras, etc). Desta forma, quando um novo colaborador entra para a organização é necessário criar as credenciais no Lightweight Directory Access Protocol (LDAP) [31] para que este tenha posterior acesso nos respetivos serviços. A ferramenta utilizada para este serviço é o OpenLDAP [32].

Foi mediante este contexto que se pretendeu desenvolver um sistema que fosse capaz de tornar a tarefa de gestão dos acessos dos utilizadores automatizada. Este sistema, que gere os controlos de acessos físicos e lógicos de acordo com a política de acessos da empresa, é denominado de *Ubiaccess*.

4.2 Descrição do Sistema Desenvolvido

Nesta secção serão mencionadas as tecnologias usadas no desenvolvimento do sistema (secção 4.2.1). Posteriormente é apresentado o modelo de dados (secção 4.2.2) e a arquitetura da aplicação desenvolvida (secção 4.2.3).

4.2.1 Tecnologias Utilizadas

A tecnologia utilizada para o desenvolvimento deste trabalho foi Java Enterprise, nomeadamente as versões *Java SE 6* e *Java EE 6* [33]. O que conduziu à adoção da tecnologia Java, para além de previamente familiarizado com a mesma, foi a sua versatilidade, em particular, a quantidade de bibliotecas e frameworks existentes em Java, permitindo interagir com outros sistemas e aplicações mais facilmente. Devido ao facto de ser *open source*, não tendo encargos financeiros foi adotado o servidor aplicacional GlassFish v3.0.1 [34]. Para além do mais, o GlassFish é desenvolvido pela mesma organização que desenvolve a tecnologia Java, a Oracle, e com perspectivas de suporte a longo prazo, pois a Oracle pretende disponibilizar um conjunto de ferramentas *open source* alternativas para as empresas [35].

O Sistema de Gestão de Bases de Dados (SGBD) utilizado para armazenamento dos dados de suporte ao modelo foi o MySQL [36], versão 5.1. Para além de ser um dos SGBD *open source* mais utilizados, o MySQL atualmente suportado pela Oracle, cumpre exigentes padrões de desempenho e disponibilidade. Outro fator para o uso desta ferramenta, é que já se encontrava em utilização para suporte de dados a outros projetos da organização. Em complemento, foi também utilizado o serviço de diretoria LDAP [37], pois este é utilizado para efeito de autenticação e autorização por alguns serviços e ferramentas usados pela empresa.

De forma a assegurar a alta disponibilidade do *Ubiaccess* em caso de falha, o serviço encontra-se a correr em duas máquinas: principal e secundária. Caso a máquina principal falhe, o sistema automaticamente reconfigura a máquina secundária para responder aos pedidos. Esta funcionalidade foi implementada através da migração do mesmo IP entre as duas máquinas. Ou seja, quando a máquina principal deixa de responder o seu IP é automaticamente migrado para a máquina secundária. Quando a máquina principal voltar ao seu estado normal, o IP é migrado de volta para a mesma, passando esta a responder aos pedidos novamente. O recurso ao DNS para a implementação de *failover* no sistema não é possível porque os leitores RFID utilizados não suportam DNS. Desta forma, esta solução de *failover* foi implementada através das ferramentas Heartbeat [38] e Pacemaker [39].

4.2.2 Modelo de Dados

Com base no modelo de controlo de acessos *Role-Organization Based Access Control*, apresentado na secção 2.4, foi criada uma base de dados para o armazenamento da política de acessos da organização. De forma a tornar o modelo capaz de suportar o controlo de acesso, quer físico, quer lógico, foram criadas algumas tabelas adicionais para o armazenamento de informação necessária à implementação destas funcionalidades. O modelo relacional da base de dados encontra-se ilustrado na Figura 4.1.

Neste trabalho a entidade organização é vista de quatro formas distintas: instalações físicas, projetos, organizações virtuais e as organizações reais. Este polimorfismo da entidade *organização* é representado pelo campo *kind* da tabela *Organizations*, no qual estes quatro tipos são diferenciados pelas etiquetas `ROOM`, `PROJECT`, `VO` e `ORG` respetivamente. Esta distinção torna-se indispensável para a implementação quer do controlo de acessos físicos quer lógicos. Com este propósito foram criadas as tabelas *Rooms* e *Projects* com os atributos adicionais necessários.

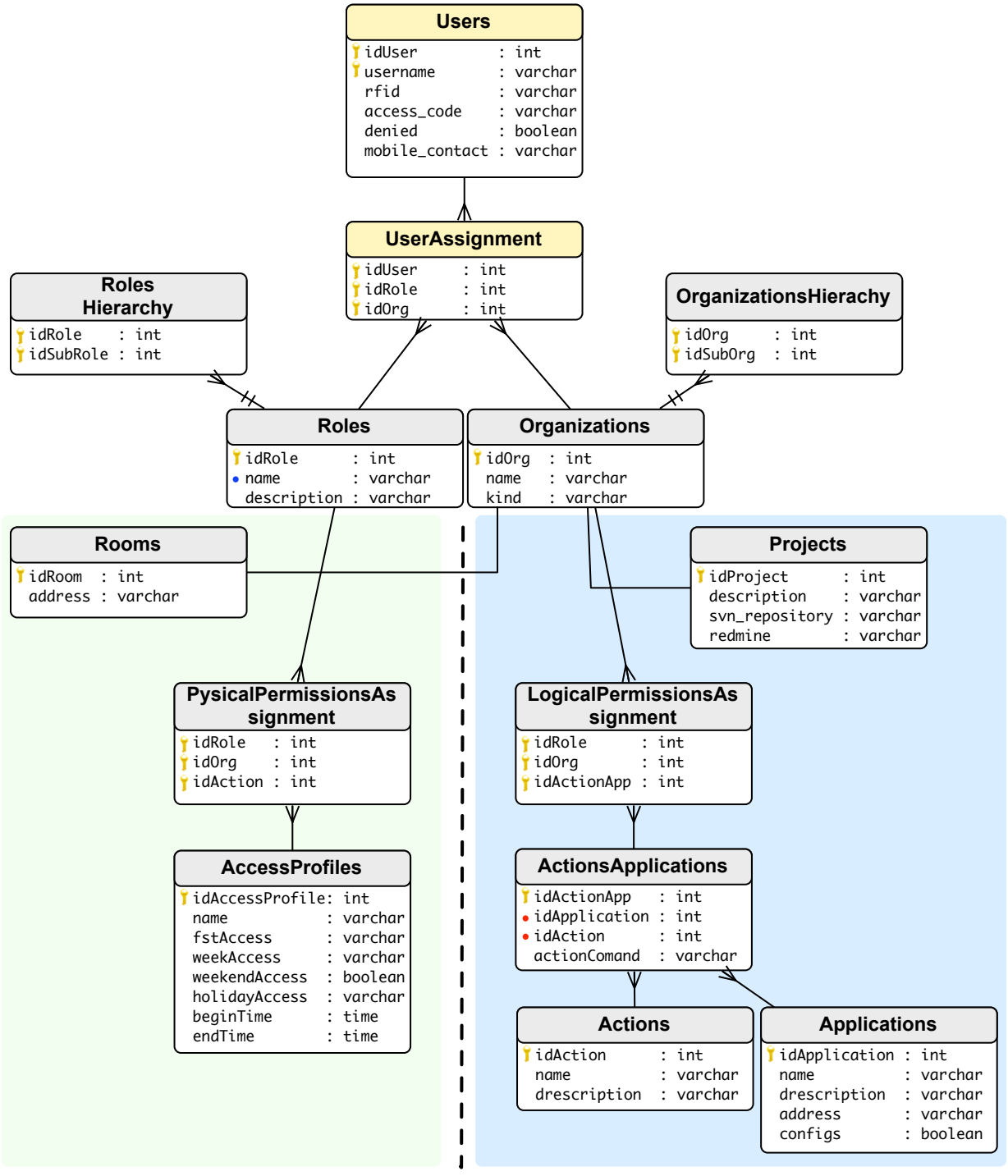


Figura 4.1: Modelo Relacional da Base de Dados

Por cada organização que existe na tabela *Organizations* do tipo `ROOM` ou `PROJECT` terá de existir informação adicional na tabela *Rooms* ou *Projects* respetivamente. Caso a organização seja definida como sendo do tipo `ORG` significa que esta é uma organização real, e os seus projetos e escritórios devem ser declarados como sub-organizações desta na tabela *OrganizationsHierachy*. As organizações do tipo `VO` servem apenas para representar políticas de acesso. A este tipo de organização(ões) - `VO` (*virtual organizaton*) - deve ser associado o conjunto de privilégios adequado a cada *Role* de forma a espelhar a política de acessos geral de empresa. Assim, sempre que é criado um novo projeto basta declará-lo como super-organização desta organização virtual e todos os privilégios serão herdados automaticamente e configurados nas respetivas aplicações (e.g. SVN e Redmine).

Na tabela *Rooms* são representadas as instalações físicas de cada organização real, como por exemplo salas e escritórios.

Os vários projetos de ID&I a decorrer na organização são representados na tabela *Projects*. O atributo *svn_repository* indica qual a diretoria do projeto no repositório svn associado. Este atributo é utilizado sempre que é associado ou desassociado um colaborador ao projeto para a atribuição ou revogação dos privilégios de leitura e/ou escrita, conforme o cargo que este desempenha no projeto. O campo *redmine* serve apenas para indicar qual o identificador do projeto na aplicação Redmine, e é utilizado para a associação de utilizadores ao projeto na aplicação.

A tabela *Applications* representa as várias aplicações que o sistema desenvolvido deve configurar de acordo com a política de acessos global. O atributo *address* indica o endereço, IP ou DNS, da máquina onde a aplicação está a correr. No campo *configurations* encontram-se guardadas todas as restantes configurações necessárias para a interação com a aplicação. Estas configurações encontram-se guardadas no formato `atributo=valor`. A configuração da aplicação é efetuada pela respetiva classe especificada no campo *name*. Há ações que podem ser executadas pelos utilizadores nas aplicações - permissões/privilégios - ou ações que simplesmente têm de ser executadas pelo sistema para configurar a aplicação corretamente. Esta relação entre as ações e aplicações é representada pela tabela *ActionsApplications*, onde cada linha desta tabela deve ser vista como uma permissão. O atributo *actionComand*, caso necessário, serve apenas para indicar parâmetros adicionais para a execução da ação na aplicação.

Os utilizadores do sistema são registados na tabela *Users*, com os atributos necessários para o controlo de acessos físico e lógico. O campo *rfid* contém o identificador do cartão RFID do

utilizador para acesso às instalações da organização. O *accessCode* é o código pessoal de acesso. O atributo *denied* indica se o utilizador está banido do sistema, isto é, caso esta *flag* se encontre a verdadeiro todos os acessos às instalações físicas serão negados ao utilizador.

A tabela *AccessProfiles* contém os diversos perfis de acesso físico às instalações da organização. Em cada perfil é especificado: se este é valido durante os dias úteis da semana e/ou fins de semana, e feriados, respetivamente pelos atributos *weekAccess, weekendAccess, holidays*; o intervalo de tempo durante o dia em que o acesso é permitido - *beginTime – endTime*. O campo *fstAccess* garante aos utilizadores o privilégio de poderem ser os primeiros a aceder a um escritório/sala da organização, i.e., caso o campo *fstAcccess* esteja a falso, os utilizados que apenas têm acesso a uma sala através desse perfil de acesso não podem entrar caso esta se encontre vazia. Desta forma, terá de haver pelo menos um perfil de acesso, com o campo *fstAccess* a verdadeiro, associado a cada sala, caso contrário ninguém conseguirá entrar. Será descrito com mais detalhe a forma como esta tabela e os seus campos são interpretados pelo módulo de controlo de acesso físicos, apresentado na secção 4.3.

4.2.3 Arquitectura da Aplicação

De forma a tornar as funcionalidades de controlo de acessos físico e lógico independentes, a aplicação de controlo de acessos - *UbiAccess* - foi desenvolvida em três módulos aplicacionais: camada de dados, módulo controlo físico e módulo controlo lógico, ver Figura 4.2. Constituindo a base do *UbiAccess*, o primeiro módulo implementa a lógica do modelo *Role,Organization Based Access* tornando-se assim o motor principal da política de acessos. Assim, é nesta primeira camada que são estabelecidas e determinadas todas as relações entre as diversas entidades do modelo: hierarquia de organizações, salas, projetos, utilizadores e privilégios. Este módulo é também responsável por toda a camada de dados, assegurando a persistência dos objetos na respetiva base de dados através do Object-relational mapping (ORM) eclipseLink. Os módulos de controlo de acessos físico e lógico serão mais tarde explorados em detalhe nas secções 4.3 e 4.4 respetivamente.

Assegurando o controlo de acessos dos utilizadores às instalações da organização, o segundo módulo implementa a lógica de controlo de acessos físicos. Este controlo é efetuado através do uso de um *Sigleton EJB* `PhysicallAccess.java` , ou seja, objeto único que mantém o estado das salas e utilizadores entre os pedidos de acesso dos utilizadores. Neste módulo encontram-se três packages essenciais: *AVEA, PhysicallAccess, TimeAttendance*. O package

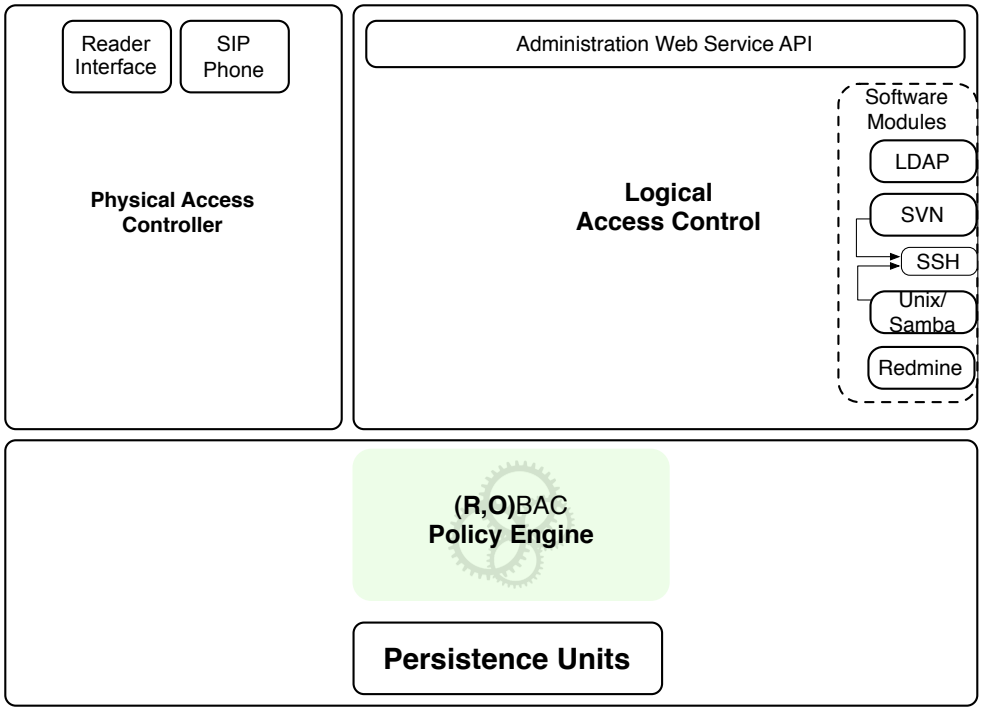


Figura 4.2: Arquitetura da aplicação de controlo de acessos - Ubiaccess

PhysicalAccess é apenas composto pelo EJB `PhysicalAccess.java` e respetiva interface. No *package AVEA* encontra-se o *Servlet* HTTP que atende os pedidos relativos aos eventos ocorridos no leitor RFID. Devido ao leitor RFID ter a limitação de não manter a mesma conexão Transmission Control Protocol (TCP) aberta por mais do que seis segundos, foi criado o *Singleton EJB* `DoorState.java`, intermediário entre o servlet HTTP e o bean `PhysicalAccess`, para controlar o estado do leitor entre os vários pedidos. Esta classe também faz parte do *package AVEA*. O package *TimeAttendance* é constituído pelas classes que implementam a funcionalidade de registo de ponto (i.e. entradas, saídas, intervalos). Na secção 4.3 é detalhadamente explicado como estes módulos interagem e as respetivas funcionalidades.

No desenho do componente de controlo de acessos lógicos, por forma a tornar a aplicação versátil e passível de uma futura integração fácil com novas aplicações e serviços, foi adotada uma arquitetura modular no seu desenvolvimento. Desta forma, conforme ilustrado na Figura 4.2, por cada aplicação/serviço com que e o UbiAccess é compatível foi desenvolvido um módulo que implementa o protocolo de comunicação da respetiva aplicação de forma a controlar a mesma. Assim, quando necessário interagir com uma nova aplicação externa, basta apenas desenvolver o

respetivo módulo que implementa e abstrai a interface/protocolo de comunicação com a respetiva aplicação. De salientar que o módulo SSH é apenas um auxiliar aos outros módulos, como por exemplo SVN e SAMBA, para a execução de comandos remotamente nas respetivas máquinas onde os serviços se encontram alojados. Cada um destes módulos contém o seu próprio *package*, dentro do package *SoftwareModules*. A interação entre os vários módulos e respetivas aplicações é descrito na secção 4.4. O principal elemento deste módulo, é o packge *Logical*, onde se encontra a classe *Administration* que implementa a funcionalidade de administração do sistema de controlo de acessos como um todo. O acesso a esta classe é feito através de um webservice Simple Object Access Protocol (SOAP) criado para integração com a Intranet da empresa.

Comum a todos os componentes da aplicação, foi desenvolvido um componente de *logging* para registar todos os eventos que vão sendo executadas pela aplicação, dependendo do nível de *log*. Os eventos são registados num ficheiro de texto com uma nomenclatura intuitiva para fácil diagnóstico de eventuais anormalidades.

As configurações dos parâmetros da aplicação são lidos de um ficheiro de texto no arranque na aplicação (ver Figura 4.3). Este ficheiro contém as definições de:

- Logger - nível de *logging* e local do ficheiro de log;
- SIP - parâmetros sip para o sipphone utilizado no controlo de acessos físico;
- Email - credenciais do correio eletrónico para envio de alertas, por exemplo tentativas de acesso não autorizadas;
- Acesso - intervalo de tempo em que é obrigatória a autenticação por RFID + Telefone. Tempo limite da chamada e número de tentativas para introduzir código pessoal;

4.3 Controlo de Acesso Físico

A autenticação dos utilizadores no acesso às instalações é efetuada através de cartões RFID. No entanto dada a vulnerabilidade do cartão RFID ser um token que pode ser facilmente extraviado, ou até mesmo ser roubado do seu detentor, também é utilizado o telemóvel do utilizador no processo de autenticação de forma a reforçar a segurança do mecanismo de autenticação dos utilizadores. A finalidade do uso do telemóvel é solicitar ao utilizador o seu código pessoal de acesso, estabelecendo uma chamada telefónica, ver Figura 4.4. A digitação do código de acesso

```
1  # ---- UbiAccess Confs --------
2
3  # SipPhone Configs:
4  sip_server = sip.ubiwhere.com
5  sip_via_addr = 192.192.192.192
6  sip_username = my_username
7  sip_passwrod = my_password
8
9  # call_duration_limit = x seconds
11 # call's duration limit, in seconds,
12 # to the user type his AccessCode
13 # most call_access_code_tries times
14 call_duration_limit = 58
15 call_access_code_tries = 3
16
17 # 24H format:  hh:mm:ss
18 not_mandatory_call_time_init = 10:00:00
19 not_mandatory_call_time_end  = 17:00:00
20
21 # eMail configurations for warning notifications
22 mail_host = maildns.ubiwhere.com
23 mail_host_port = 1234
24 mail_username = ubiaccess@ubiwhere.com
25 mail_password = my_password
26 mail_notifiers_list = jpnovais@ubiwhere.com, systemmanager@ubiwhere.com
27
28 # LOG configurarions:
29 log_directory = /path_to/logs
30 log_level = ALL
31
32 # log levels:
33 #   OFF, SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST, ALL
```

Figura 4.3: Ficheiro de configurações do *Ubiaccess*

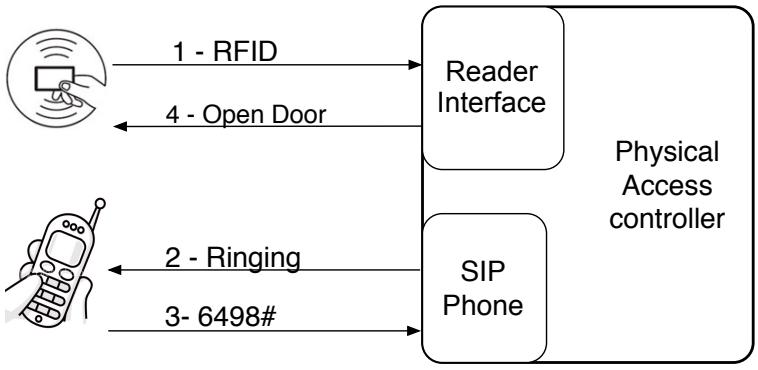


Figura 4.4: Processo de autenticação para acesso físico.

é reconhecida através de tons DTMF [40]. No entanto, de forma a simplificar o processo de autenticação, este código de acesso só é solicitado ao primeiro utilizador que tenta aceder a um dado escritório da empresa, os restantes utilizadores apenas têm de se autenticar com o cartão RFID à entrada. Pois uma vez que já se encontra um utilizador nas instalações devidamente autenticado, caso haja um intruso que tente entrar com um cartão RFID perdido/roubado será de

imediato detetado por esse mesmo utilizador. Entende-se assim, que esta relação complexidade vs. risco é a mais equilibrada.

Quando o utilizador se autêntica com o seu cartão de acesso RFID, o leitor envia um pedido de acesso HTTP GET, onde contém o código RFID do cartão que identifica o utilizador. Este pedido é atendido pelo Servlet `avea` que extrai as variáveis do pedido GET, e de acordo com parâmetros do pedido, passa este pedido para o statefull EJB `DoorState`, que por sua vez reencaminha o pedido ao singleton bean `PhysicalAccess` que efetivamente controla os acessos físicos (ver diagrama de sequência na Figura 4.5). Nesta comunicação intermédia entre o servlet que recebe os pedidos do leitor e o objeto `PhysicalAccess`, o objeto `DoorState` mantém um estado de autorização dos pedidos. Este objeto intermédio, com o estado dos pedidos, é necessário para os casos em que o utilizador é o primeiro a aceder às instalações e o controlador `PhysicalAccess` precisa de efetuar uma chamada para o telemóvel do utilizador para este introduzir o seu código de acesso. Dado que, caso o leitor RFID não receba a resposta ao pedido efetuado dentro de seis segundos fecha a conexão TCP, tempo insuficiente para obter o código de acesso através do telemóvel do utilizador, é necessário guardar no objeto `DoorState` o estado de que a porta deve ser aberta da próxima vez que o leitor enviar um pedido ao sistema. Normalmente, este pedido é efetuado automaticamente sempre que ocorre um *heartbeat*, que está configurado para 10 segundos. Assim, o atraso entre o momento em que o código pessoal é inserido no telemóvel e a porta é efetivamente aberta é de 10 segundos.

Após carregar a informação do utilizador com base no código RFID, e da sala através do seu id no pedido de acesso - `boolean canAccess(String rfid, String roomID)` -, o controlador de acessos físicos - `PhysicalAccess` - efetua os seguintes passos para autenticação e permissão de acesso ao utilizador:

- Primeiro verifica se o utilizador não foi banido do sistema - *denied* -, caso sim é imediatamente retornado falso.
- Através do motor do modelo ROBAC na camada abaixo, é obtida uma lista dos perfis de acesso que estão diretamente ou indiretamente atribuídos à sala em questão e aos cargos que o utilizador desempenha nessa sala. Caso esta lista seja vazia, é de imediato retornado o valor falso, pois o utilizador não tem, de forma alguma, acesso à sala em questão.
- Para cada perfil de acesso da lista anteriormente obtida verificar se, para aquele dado momento (hora/dia da semana), há pelo menos um que garanta o acesso ao utilizador.

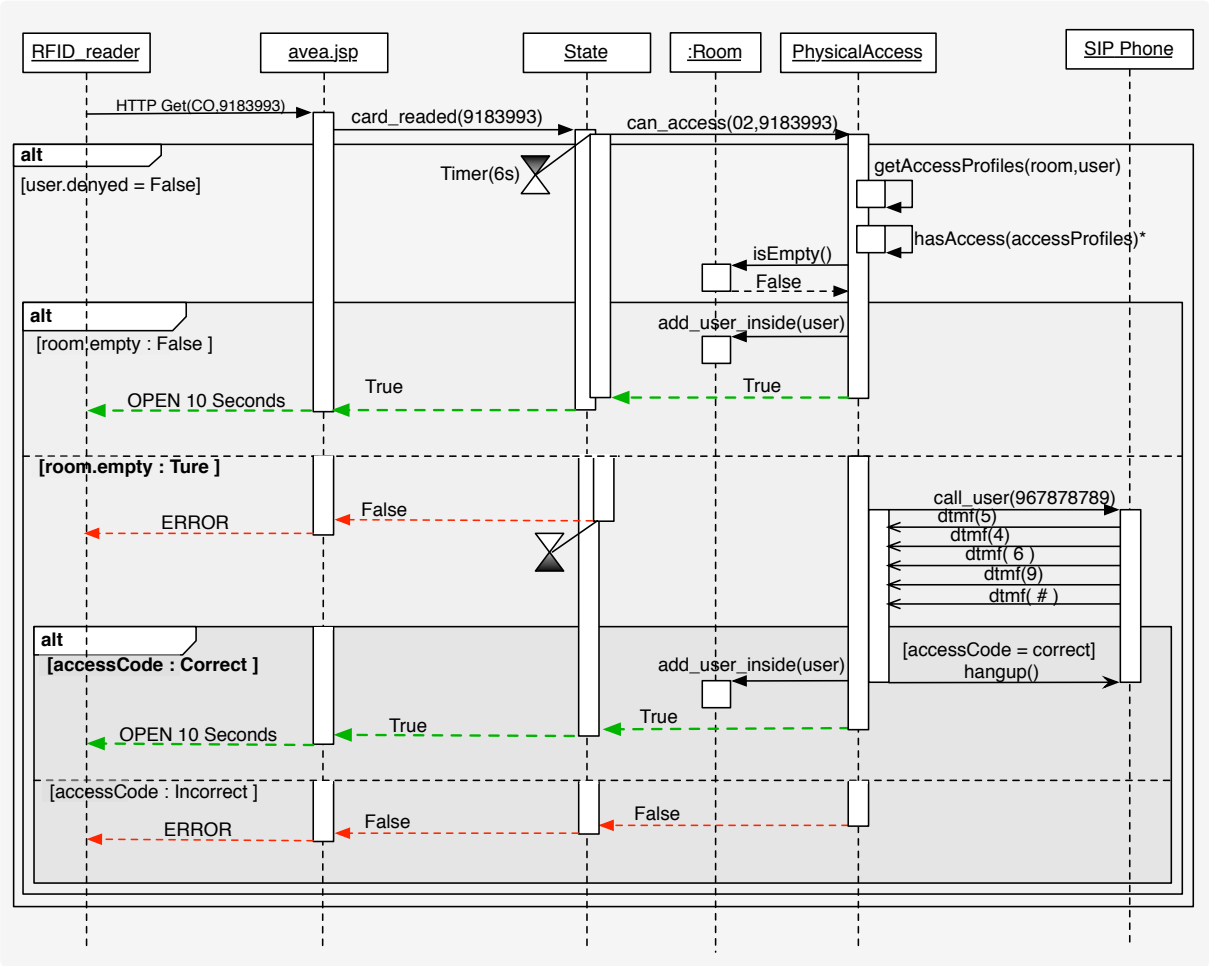


Figura 4.5: Processo de autenticação para acesso físico.

- É verificado o estado da sala:

  ◦ Caso já se encontre alguém na sala, é imediatamente garantido o acesso ao utilizador.

  ◦ Se a sala estiver vazia, o sistema efetua uma chamada para o telemóvel do utilizador. Após atender a chamada, o utilizador deve marcar o seu código de acesso seguido de #, num prazo de 60 segundos. Validado o código de acesso correto, é finalmente permitido o acesso ao utilizador à sala.

- Após ser permitido o acesso ao utilizador, é registada o seu tempo de entrada na base de dados.

Sempre que um utilizador sai da sala deve registar a sua saída no leitor RFID. Cada saída é assinalada através de um código antes de passar o cartão RFID no leitor, que indica o propósito da saída, por exemplo almoço, intervalo ocasional, saída final do dia de trabalho, etc. No entanto, estas marcações de saída explícitas estão sujeitas ao esquecimento dos utilizadores, comprometendo assim a segurança das instalações. Caso os utilizadores se esqueçam de dar saída e seja roubado o cartão de um utilizador seria o suficiente para entrar nas instalações. De forma a minimizar este risco ao máximo, são tomadas várias medidas de segurança adicionais: a primeira, e mais simples, é o uso de um código específico para o último utilizador a sair indicar que é efetivamente o último a sair, o sistema colocar a sala no estado *empty*; a segunda é a especificação de um intervalo de tempo para o qual a autenticação com RFID e telefone é obrigatória, mesmo que na contagem do sistema ainda haja utilizadores dentro da sala; ainda em complemento, para cada sala é mantida uma lista dos utilizadores que estão dentro da mesma bem como o tempo da última entrada na sala. Assim, caso não haja atividade na sala por mais de um determinado tempo, é assumido automaticamente que a sala se encontra vazia. Estes tempos são especificados no ficheiro de configurações.

Sempre que algum comportamento anormal é detetado pelo sistema, tais como tentativas de acesso não autorizadas, é enviado um alerta por correio eletrónico para uma lista de endereços de correio eletrónico previamente especificados no ficheiro de configurações.

O telefone SIP [41] utilizado neste trabalho é o MjSip [42]. Apesar deste software já implementar bastantes funcionalidades, foi necessário acrescentar o reconhecimento de dígitos DTMF segundo a rfc2833 [40], o que obrigou a investir algum tempo a estudar o seu código fonte para acrescentar esta funcionalidade. Foi ainda necessária uma pequena alteração vertical nas classes que implementam, desde o *User Agent* até à classe de leitura dos pacotes RTP [43], para o suporte de *callbacks* sempre que é reconhecido um digito DMTF.

4.4 Controlo de Acesso Lógico

O acesso às várias aplicações e serviços da empresa não é diretamente garantido ou revogado pelo sistema desenvolvido, mas sim pelas próprias aplicações, que são corretamente configuradas pelo *Ubiaccess*. Para que esta gestão de permissões seja efetuada de forma mais simples e unificada, o sistema reconfigura as várias aplicações e serviços de forma a espelharem uma política de acessos única, definida no sistema, de acordo com a estrutura da organização. Assim,

sempre que é efetuado um pedido ao *Ubiaccess*, por exemplo associar um utilizador a um projeto com o dado cargo, as aplicações são reconfiguradas por forma a refletirem essa nova alteração.

A adoção do modelo ROBAC para a implementação de uma única política de acessos, de forma a unificar os diversos modelos de acesso de cada uma das aplicações, conduziu ao uso de permissões como associação de privilégios ou ações que podem ser executadas numa determinada aplicação. Uma permissão ou privilégio, é interpretado de forma abstracta como sendo uma *ação* que um *utilizador* pode efetuar numa determinada *aplicação* ou serviço. Por exemplo o utilizador *Joaquin* pode *ler* a diretoria *Projecto Alfa* no repositório de *svn*. Esta abstração levou à definição de uma interface para a aplicação de regras nas aplicações.

A abstração e uniformização do conceito de permissão levou a que fosse criada uma interface para que as regras fossem aplicadas transversalmente às aplicações e serviços na empresa. Esta interface, abaixo apresentada na Figura 4.6, deve ser implementada por cada um dos módulos respetivos às aplicações a serem configuradas.

```
1  public boolean apply_rule(Users user,
2                            Roles role,
3                            OrganizationsAbstract org,
4                            Actions action)
5                 throws Exception;
6
7  public boolean remove_rule(Users user,
8                             Roles role,
9                             OrganizationsAbstract org,
11                            Actions action)
12                throws Exception;
```

Figura 4.6: Interface SoftwareModule.java

De forma a compreender melhor o funcionamento da aplicação de regras, imagine-se o seguinte exemplo: na empresa sempre que um novo projeto surge, este tem de ser criado no Redmine sendo também é criada a respetiva diretoria no repositório SVN. Quando um novo utilizador é adicionado ao sistema, este deve ser adicionado ao LDAP, e criada a respetiva conta no domínio samba da empresa. Dado que estes procedimentos são repetidos sempre que é criado um novo projeto ou utilizador respetivamente, vamos criar uma organização chamada *Política de acessos*

lógicos do tipo VO onde estas regras vão ser associas aos cargos *CREATE_PROJECT* e *CREATE_USER* respetivamente, como ilustrado na Figura 4.7. Estes dois cargos são auxiliares, pois não podem ser atribuídos a nenhum utilizador em nenhuma organização, sendo apenas utilizados quando novos projetos ou utilizadores são criados.

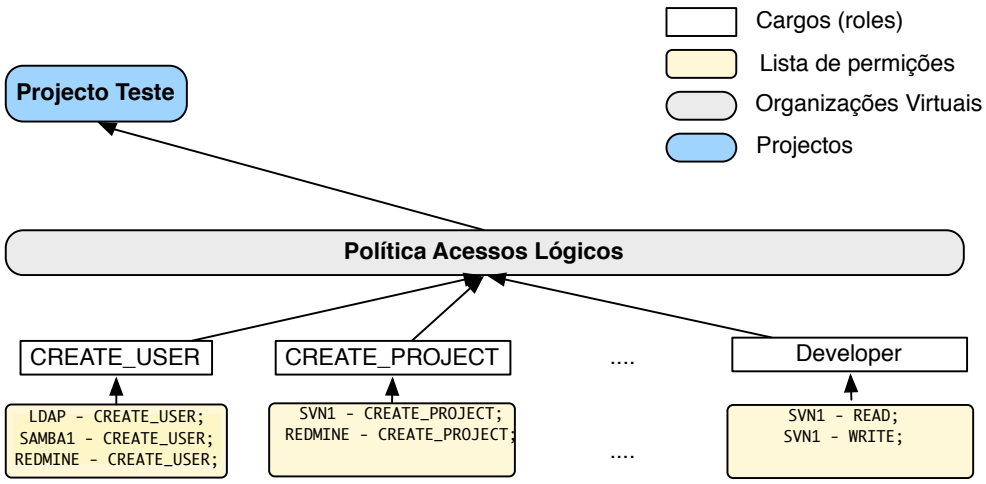


Figura 4.7: Política de acessos lógicos

Assim, na criação de um projeto basta invocar apenas dois métodos no webservice de administração: *createProject(Projecto Teste,...)* que apenas adicionará o projeto à base de dados do Ubiaccess, e de seguida o método *createOrganizationHierarchy(Projecto Teste, Política de acessos lógicos)*. O segundo método estabelece que a organização *Política de acessos lógicos* passa a sub-organização de *Projecto Alfa*, o que implica que *Projecto Alfa* herda as regras que estão definidas em *Política de acessos lógicos*. Assim, ao executar este método são determinadas e posteriormente executadas as permissões/regras que *Projecto Alfa* herda de *Política de acessos lógicos*.

De acordo com o exemplo descrito e ilustrado na Figura 4.7, ao executar *createOrganizationHierarchy(Projecto Teste, Política de acessos lógicos)* o Projecto Teste herda as ações CREATE_PROJECT nas aplicações SVN1 e Redmine. Assim, neste processo serão executados nos módulos SVN e Redmine o método *apply_rule(null, CREATE_PROJECT, Projecto Alfa, CREATE_PROJECT)*. O módulo SVN criará a diretoria especificada pelo atributo *svn_repository* no respetivo repositório SVN. O módulo Redmine criará um projeto na respetiva aplicação Redmine com o identificador especificado pelo atributo *redmine*.

Ainda no seguimento do mesmo exemplo, suponha-se que se adiciona um segundo repositório SVN ao *Ubiaccess*, como ilustrado na Figura 4.8. Repositório este que passa a ser utilizado para alguns projetos que surgem desde então. Tal como foi feito antes, para tornar o processo de criação de projetos automático será criada mais uma VO chamada *Política 2*, onde serão adicionadas as ações CREATE_PROJECT, mas agora a referenciar o novo repositório SVN. Quanto ao Redmine, manter-se-á o mesmo (ver Figura 4.8). Agora, sempre que se pretenda que um novo projeto, por exemplo *Projecto Ômega*, utilize como repositório o segundo SVN adicionado ao sistema, basta torná-lo super-organização da organização virtual *Política 2*, através do método *createOrganizationHierarchy(Projecto Ômega, Política 2)*.

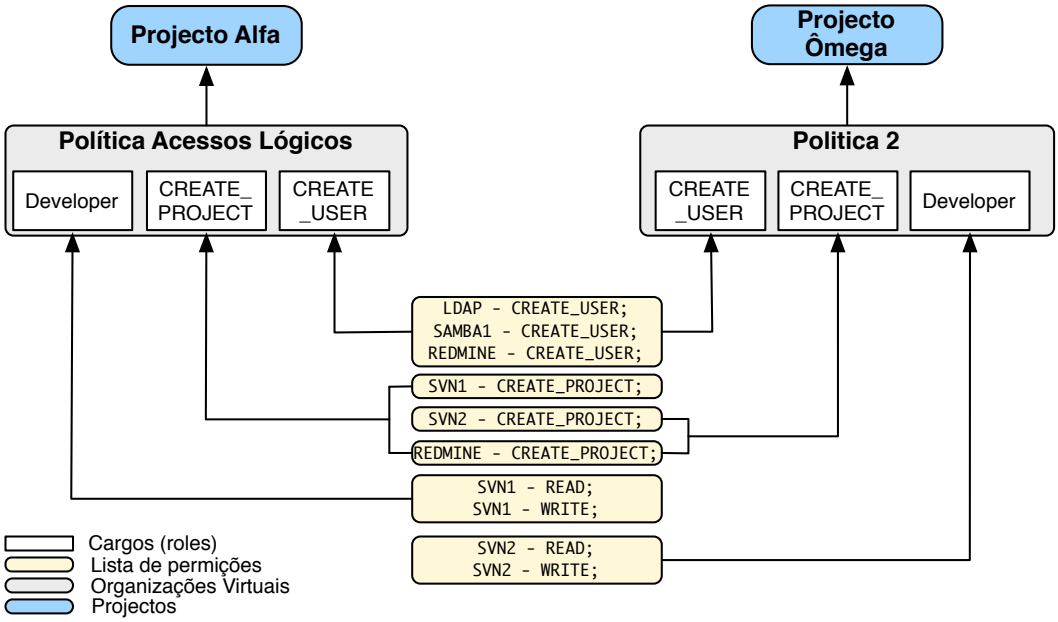


Figura 4.8: Política de acessos lógicos 2

Para completar o exemplo, imagine-se que no seguimento do *Projecto Alfa* surge um novo projeto, denominado de *Projecto Beta*, e se pretende que os programadores deste novo projeto também tenham acesso ao código fonte e documentação do projeto Alfa. A forma mais fácil de executar estes requisitos é tornar o *Projecto Beta* super-organização do *Projecto Alfa*. Assim, sempre que um novo programador (cargo Developer) é associado ao projeto *Beta*, automaticamente também será associado ao projeto *Alfa*, caso não tenha sido antes, de forma a que este tenha acesso aos dois projetos como é pretendido.

Apesar de numa primeira análise das Figuras 4.7 e 4.8, parecer que os programadores apenas serão associados aos projetos com as permissões de leitura e escrita no repositório de SVN, estes também serão automaticamente associados na respetiva aplicação Redmine. Esta associação é determinada através do triplo *(Role,Organization,Permission) CREATE_PROJECT, Política de acessos lógicos, REDMINE - CREATE_PROJECT* [1] que indica em qual Redmine o projeto foi criado.

A associação de um utilizador a um projeto é efetuada através do método *boolean assignUser(String username, String role, String Organization)*. Ao invocar *assignUser(adleman, Developer, Beta)*, são determinadas as permissões associadas aos pares (Developer,Beta) e (CREATE_PROJECT,BETA), tendo em conta o exemplo acima, serão obtidas as listas {SVN1 - READ, SVN1 - WRITE} e {SVN1 - CREATE_PROJECT, REDMINE - CREATE_PROJECT} respetivamente. Para cada uma destas ações é invocado o método *apply_rule(Users user, Roles role, OrganizationsAbstract org, Actions action)* na instância da respetiva aplicação. No processo de iteração das ações acima, para serem reconfiguradas as respetivas aplicações, o role CREATE_PROJECT é tratado de forma diferente. Neste contexto de associação de utilizadores a projetos, a ação REDMINE - CREATE_PROJECT será invocada da forma *redemineInstance.apply_rule(adleman, Developer, Beta, ASSIGN_USER)*. A iteração das ações obtidas resultará nas invocações apresentadas na Figura 4.9. A forma como cada uma é aplicada por cada módulo é explicado com mais detalhe abaixo nas secções 4.4.1 e 4.4.2.

```
1 REDMINE.apply_rule( adleman, Developer, Beta, ASSIGN_USER )
2 SVN.apply_rule( adleman, Developer, Beta, ASSIGN_USER )
3
4 SVN.apply_rule( adleman, Developer, Beta, READ )
5 SVN.apply_rule( adleman, Developer, Beta, WRITE )
```

Figura 4.9: Regras aplicadas

Graças ao elevado nível de abstração da interface *SoftwareModule* e das relações entre aplicações e ações/permissões, a invocação do método *apply_rule* para um conjunto de ações associadas e herdadas por um par *(Role, Organization)* torna-se simples e intuitivo, como ilustrado

[1] São apenas utilizadas etiquetas como CREATE_PROJECT e REDMINE para melhor perceção. Pois este trio é representado pela tabela LogicalPermissionsAssignment com os respetivos ids do cargo CREATE_PROJECT, Política de acessos lógicos, e id da tabela ActionsApplications, que por sua vez referência a ação CREATE_PROJECT numa instância da aplicação Redmine. Ver modelo de dados na Secção **??**

excerto de código da Figura 4.10.

```
1  Collection<ActionsApplications> actions_app;
2  actions_app = this.find_inherited_logical_permissions(role, p);
3  for( ActionsApplications action_app : actions_app)
4  {
5      Applications   app        = action.getApplication();
6      SoftwareModule app_module = this.getSoftwareModule(app);
7      app_module.apply_rule(user,
8                            role,
9                            p ,
11                           action_pp.getActionLabel()
12                           );
13 }
```

Figura 4.10: Código exemplo do processo de aplicação de regras

Todos os métodos que implicam a revogação ou eliminação de ações/permissões, procedem de mesma forma semelhante ao exemplo em cima, mas utilizam o método *remove_rule(Users user, Roles role, OrganizationsAbstract org, Actions action)* para apagar as regras nas respetivas aplicações. Por exemplo, a invocação de *remove_rule(adleman, Developer, Beta, WRITE)* sobre aplicação SVN faria que o utilizador *adleman* perdesse permissão de escrita na diretoria do projeto *Beta* no respetivo repositório de SVN.

4.4.1 SVN

Este módulo foi implementado com recurso ao módulo SSH, que é utilizado para executar comandos remotamente nas máquinas onde estão a correr as aplicações de svn. Para editar o ficheiro de permissões do svn, foi criada uma classe que é capaz de fazer parser do ficheiro e criar uma estrutura lógica de dados que representa esse mesmo ficheiro. Sempre que a sua estrutura lógica é alterada, por exemplo quando a diretoria de um novo projeto é adicionada, esta estrutura volta a ser gravada para ficheiro.

As ações suportadas por este módulo são: CREATE_PREJECT, READ e WRITE. CREATE_PREJECT, como o próprio nome indica, serve para criar a diretoria de um novo projeto, ou remover essa diretoria caso esta seja invocada pelo método *remove_rule*. READ e WRITE são utilizadas para dar permissão de leitura e ecrita aos utilizadores associados aos projetos. Quando uma destas ações é invocada no módulo svn, o ficheiro de permissões remoto é copiado por ssh

para a máquina local, acrescentada a nova ação ao ficheiro e copiado de volta para máquina de origem. Em particular, na invocação da CREATE_PROJECT é executado o comando *svn mkdir -m project_name* remotamente através de ssh para criar a respetiva diretoria do projeto.

4.4.2 Redmine

O módulo de redmine foi desenvolvido com uso da, única e oficial, API *Redmine & Chili-project Java API* [44], versão 1.3.0. No entanto, esta API ainda não implementa algumas das funcionalidades que são desejáveis para realização completa deste trabalho. Tendo em conta o atual conjunto de funcionalidades implementadas por esta versão da API, aquelas que no âmbito deste trabalho importam para a implementação deste módulo são as funcionalidades de criar e apagar projetos e utilizadores. Assim, as ações implementadas por este módulo são duas: CREATE_PROJECT e CREATE_USER. Apesar de não implementar todas as funcionalidades necessárias, esta API continua em desenvolvimento, tendo como prioridades do seu *road map* algumas funcionalidades necessárias para este módulo, tais como associação de utilizadores a projetos.

4.4.3 LDAP

A aplicação LDAP é utilizada para efeitos de autenticação de forma a centralizar as credenciais de todos os dos utilizadores. Assim as restantes aplicações e serviços autenticam os utilizadores no LDAP, tendo cada utilizador um username e password universal. Os serviços que autenticam os utilizadores junto do LDAP são os seguintes: portal web da intranet, domínio samba, serviço de impressoras, Redmine e SVN. Dado que o LDAP é utilizado apenas para centralização de credenciais, este módulo apenas implementa a ação CREATE_USER.

4.4.4 SSH

O objetivo deste módulo é executar uma sequência de comandos, com vários argumentos, remotamente através do protocolo SSHv2. Durante a execução do comando vai sendo obtido *output* que o comando envia para o *stdout* e registado no ficheiro de log. No fim da execução do comando na máquina remota é obtido o código do resultado da execução do comando, caso este seja diferente de 0, significa que ocorreu um erro, sendo lançada uma exceção. Este módulo foi implementado com uso da biblioteca Ganymed SSH-2 for Java.

Este modulo foi implementado apenas para permitir aos módulos SVN e LDAP executar comandos remotamente, este módulo não implementa nenhuma ação da interface SofwareModules.

4.5 Sumário

Este capítulo centrou-se na descrição da aplicação desenvolvida para o controlo de acessos. Inicialmente foi feita uma contextualização ao cenário real de implementação do sistema. De seguida, foi apresentada a arquitetura da solução, detalhando o modelo dados de suporte à aplicação desenvolvida bem como a sua arquitetura. De seguida foi efetuada a descrição da implementação do módulo de controlo de acessos físicos e do mecanismo de autenticação por RFID e telefone. Por último, no controlo de acessos lógicos, foi descrito como as permissões do sistema são configuradas nas diversas aplicações e serviços da empresa.

Capítulo 5

Testes e Avaliação do Sistema

De modo a validar a aplicação desenvolvida, esta foi inicialmente submetida a alguns testes mais simples. De seguida, para avaliar o desempenho da aplicação face a cenários de maior amplitude e exigência foram criados testes de carga. O ambiente de testes é igual ao ambiente de produção, em que a máquina onde se encontra o Ubiaccess é partilhada por outros serviços. Serão apresentados e comentados os resultados obtidos, bem como uma breve auto-avaliação do sistema.

5.1 Ambiente de Testes

Esta secção tem como objetivo apresentar o ambiente de testes e descrever as características técnicas, mais relevantes, dos componentes e tecnologias utilizadas. A topologia do ambiente de testes (*testbed*) está representada na Figura 5.1, onde se pode observar desde já que a maior parte dos serviços e aplicações utilizadas pela empresa estão alojados na máquina *Services*. A rede de core da empresa desde da máquina de testes cliente até ao servidor utiliza a tecnologia *fast Ethernet 100BASE-T*.

Os principais componentes da testbed são os seguintes:

- Máquina *Services* – este componente é uma máquina virtual onde se encontram alojados os seguintes serviços utilizados pela empresa: Apache2, Asterisk, MySQL, OpenLDAP, Redmine, SVN e o servidor aplicacional Glassfish v3.0.1 onde se encontra a correr o Ubiaccess. Esta máquina está configurada com um processador dual-core de 2.544 GHz com 4MB de cache e 1GB de memória.

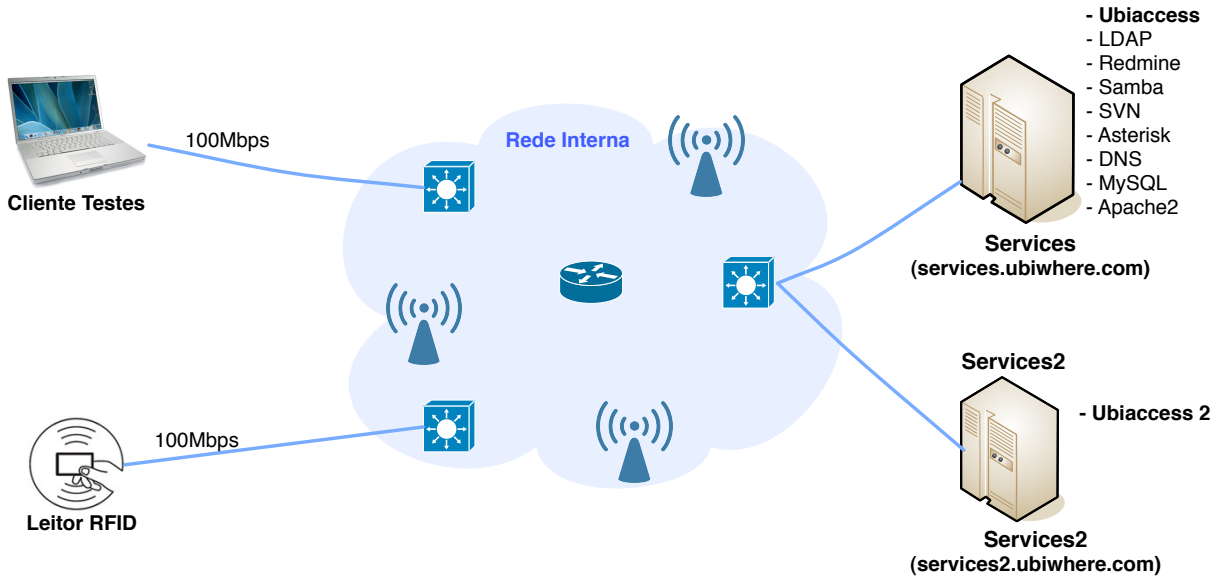


Figura 5.1: Ambiente de Testes

- Máquina *Services2* – também é uma máquina virtual, que de momento, apenas tem uma instância do servidor Glassfish com a aplicação Ubiaccess. Esta instância foi criada apenas para suporte de tolerância a falhas (*fail-over*) do Ubiaccess. Assim, esta apenas recebe os pedidos caso máquina *Services* falhe. Esta máquina está configurada com um processador single-core de 2.533GHz com 4MB de cache e 512MB de memória.

- Máquina Cliente – este componente é um computador físico, utilizado para simular pedidos do leitor RFID e medir os tempos de resposta. Este computador é constituído por um processador core-2-duo 2.4GHz com 3MB cache e 3GB de memória DRR3 a 667MHz.

- MySQL – Neste SGBD é onde se encontra a base de dados da política de acessos utilizada pelo *Ubiaccess* (apresentada na seção 4.2.2) e a base de dados de registo dos tempos das entradas e saídas dos utilizadores (*Time Attendance*). O motor de dados utilizado nestas duas bases de dados é o InnoDB. A versão do MySQL utilizada foi a versão 5.0.51a.

- GlassFish – é o servidor aplicacional J2EE que está a alojar a aplicação *Ubiaccess*. Atualmente estão a ser utilizadas as configurações default do servidor, com tamanho máximo de memória heap de 512MB (especificado pelo parâmetro -XX:MaxMemSize = 512m). Dado que a informação da política de acessos e dos registos de entradas dos utilizadores (*time attendance*) está em bases de dados diferentes, o acesso a estas obriga a que existam duas *pools* de acesso às bases de dados respetivamente. Estas *pools* atualmente também

estão configuradas com os valores atribuídos por omissão pelo GlassFish. A versão do JDK instalado nas máquinas *Services* e *Services2* é a versão 1.6.0.22.

5.2 Metodologia

De forma a poder avaliar o comportamento e desempenho da aplicação desenvolvida, foram efetuados vários testes de carga à aplicação Ubiaccess. Para este fim, com base na atual política de acessos da empresa, foi criada uma cópia da base de dados atual para uma nova base de dados de testes. A esta nova base de dados foram acrescentados 200 000 utilizadores virtuais, que foram associados ao par cargo-organização *FullAccess-Ubiwhere Aveiro*. A organização *Ubiwhere Aveiro* é super-organização da organização virtual (VO) *Physical Access Policy 1*. Na organização *Physical Access Policy 1* ao cargo *FullAccess* é associado o perfil de acesso físico *FullTimeAccess*, o qual permite o acesso às instalações em qualquer horário. Assim, através da hierarquia de organizações estes utilizadores associados ao par *FullAccess-Ubiwhere Aveiro* passam a ter acesso a qualquer hora à organização *Ubiwhere Aveiro*.

Neste tipo de sistemas, um fator significante para os tempos de resposta é o acesso aos dados, ou seja, se os dados já estão disponíveis na memória primária da aplicação (memória RAM), ou se ainda tem de ser lidos da base de dados pela primeira vez. Assim, tendo em conta estes dois cenários, para avaliar o desempenho do Ubiaccess na generalidade serão realizados os seguintes testes de carga:

- **Teste A** – Este teste consiste num conjunto de 1000 pedidos de acesso de 1000 utilizadores distintos. Estes pedidos são efetuados sequencialmente desde o utilizador 1 até ao utilizador 1000, com um tempo de intervalo entre os testes A1 e A2 de 10 segundos.

  A1 – Pedidos de acesso efetuados após ter sido efetuado o deploy da aplicação. Ou seja, quando a aplicação receber o primeiro pedido, antes de responder ao primeiro pedido terá de efetuar alocação de recursos de memória bem como a leitura do ficheiro de configurações;

  A2 – São repetidos os mesmos pedidos de acesso em A1, de forma avaliar os tempos de resposta com os dados já na memória da aplicação.

- **Teste B** – Ao contrário do teste A, onde os utilizadores eram sequenciais de 1 a 1000, o objetivo deste teste é simular um sistema real em que os utilizadores são aleatórios. Assim, neste teste são gerados 5 conjuntos de 1000 pedidos de acesso com utilizadores aleatórios

entre 0 e 200 0000. Estes 5 conjuntos de pedidos foram gerados através de um pequeno programa, criado para efeito, que utiliza a função Random do Java para gerar os números aleatórios, seguindo uma distribuição normal[1] N(101077,55492[2]).

B1 – estes são os pedidos de acesso do 1º conjunto de 1000 utilizadores aleatórios após ter sido efetuado o deploy da aplicação;

B2-1 – 2º conjunto de pedidos de acesso aleatórios;

B2-2 – 3º conjunto de pedidos de acesso aleatórios;

B2-3 – 4º conjunto de pedidos de acesso aleatórios;

B2-4 – 5º conjunto de pedidos de acesso aleatórios;

Dado que o leitor RFID envia os pedidos de acesso para abrir a porta através de pedidos HTTP GET, foi utilizada uma ferramenta geradora de pedidos HTTP para simular os pedidos de acesso do leitor. A ferramenta utilizada foi o curl [45]. Numa primeira abordagem, foi utilizada a ferramenta Apache JMeter [46, 47], pois é muito mais versátil. No entanto, ao comparar os resultados obtidos através do curl e do JMeter, verificou-se que o JMeter introduzia um atraso significativo na geração e processamento de pedidos. Para um conjunto de 1000 pedidos, o JMeter por vezes demorava 5 segundos a mais que o curl. Assim, utilizado a ferramenta curl, são obtidas as métricas:

- tempo de resposta por pedido (**tr**) – representa o tempo desde que o pedido sai da máquina cliente até que é recebida e processada a sua resposta.
- tempo de resposta total (**trt**) – tempo total que o servidor demora a responder a um conjunto de *N* pedidos de acesso. Estes pedidos são efetuados individualmente de forma sequencial.

Do lado do servidor, ou seja na máquina *Services*, foi utilizado o comando `top` do linux para medir a percentagem de processador (%CPU) consumida por cada processo durante os vários testes. Uma vez que a máquina *Services* tem dois processadores, as amostras da percentagem de CPU, por defeito, serão numa escala de 0% a 200%. No entanto para melhor perceção, na apresentação de resultados, os valores são apresentados na escala de 0% a 100%. Este comando foi configurado para fazer amostragens dos processos do GlassFish e MySQL com uma frequência de 1 segundo.

[1] Apresentada como $N(x, \sigma^2)$, em que x é a média da distribuição e σ é o desvio padrão.

5.3 Testes Realizados

Nesta secção serão apresentados e avaliados os resultados dos vários testes de carga efetuados ao módulo de acessos físicos. Serão ainda ilustrados também alguns testes efetuados ao módulo de acessos lógicos, com enfoque na configuração do LDAP, SVN e Redmine quando são criados utilizadores e projetos no sistema.

5.3.1 Testes de Desempenho ao Módulo de Acessos Físicos

Teste A

Na execução do teste **A1** (primeiros 1000 pedidos após a aplicação ter reiniciado) foi obtido um tempo de resposta total (**trt**) de 32.586 segundos (ver Tabela 5.1), o que perfaz um tempo médio de resposta de 32.586 milissegundos. Ao observar o gráfico CPU A1 da Figura 5.2 podemos constatar que, durante a execução do teste, o GlassFish consumiu apenas 39% de CPU e o MySQL 8%.

Ao executar o teste **A2** (repetir os mesmos 1000 pedidos em A1), foi obtido um tempo de resposta total de 17.342 segundos, ou seja aproximadamente metade do tempo obtido no teste A1 (53%). Esta diferença deve-se ao facto de os dados dos utilizadores já se encontrarem na memória do GlassFish, e também já terem sido em A1 previamente criados na base de dados os registos de entrada dos utilizadores (*time attendance*), refletindo-se num consumo do CPU pelo MySQL de apenas 1% (ver gráfico CPU A2 da Figura 5.2). Além do tempo de resposta médio, o facto dos dados já estarem em memória, também faz reduzir bastante a variância dos tempos de resposta. Neste caso reduz de 8.42^2 em A1 para 3.4^2 em A2 (ver Tabela 5.1). Esta redução do tempo de resposta e variância pode ser observada no Gráfico A da Figura 5.2, que contém os tempos de resposta dos primeiros 100 utilizadores[2].

Avaliando estes resultados, pode-se desde já concluir que caso o Ubiaccess tenha de se aceder à base de dados para obter os dados do utilizador, o tempo de reposta ao seu pedido de acesso pode ter uma penalização na ordem dos 100%. O facto da percentagem de CPU consumida pelo GlassFish neste teste A2 ter aumentado em apenas em 1%, aumentando assim para os 40%, permite também identificar uma potencial oportunidade de melhoria de desempenho do sistema. Pois, uma vez que o Ubiaccess não tem de esperar pela resposta da base de dados, este apenas

[2]Dado que o valor do primeiro pedido é muito discrepante dos restantes, foi excluído para tornar o gráfico mais percetível.

tem de processar a lógica de controlo de acessos, sendo assim expectável que o consumo de CPU aumentasse de forma a dar resposta aos pedidos mais rapidamente.

Tabela 5.1: Valores estatísticos do tempo de resposta para o teste A.

Tempos de Resposta (milissegundos)	**A1**	**A2**
Médio	32.586	17.342
Mínimo	21.0	13.0
Máximo	1 940.0	49.0
Desvio Padrão	8.42	3.40
Total (segundos)	32.586	17.342

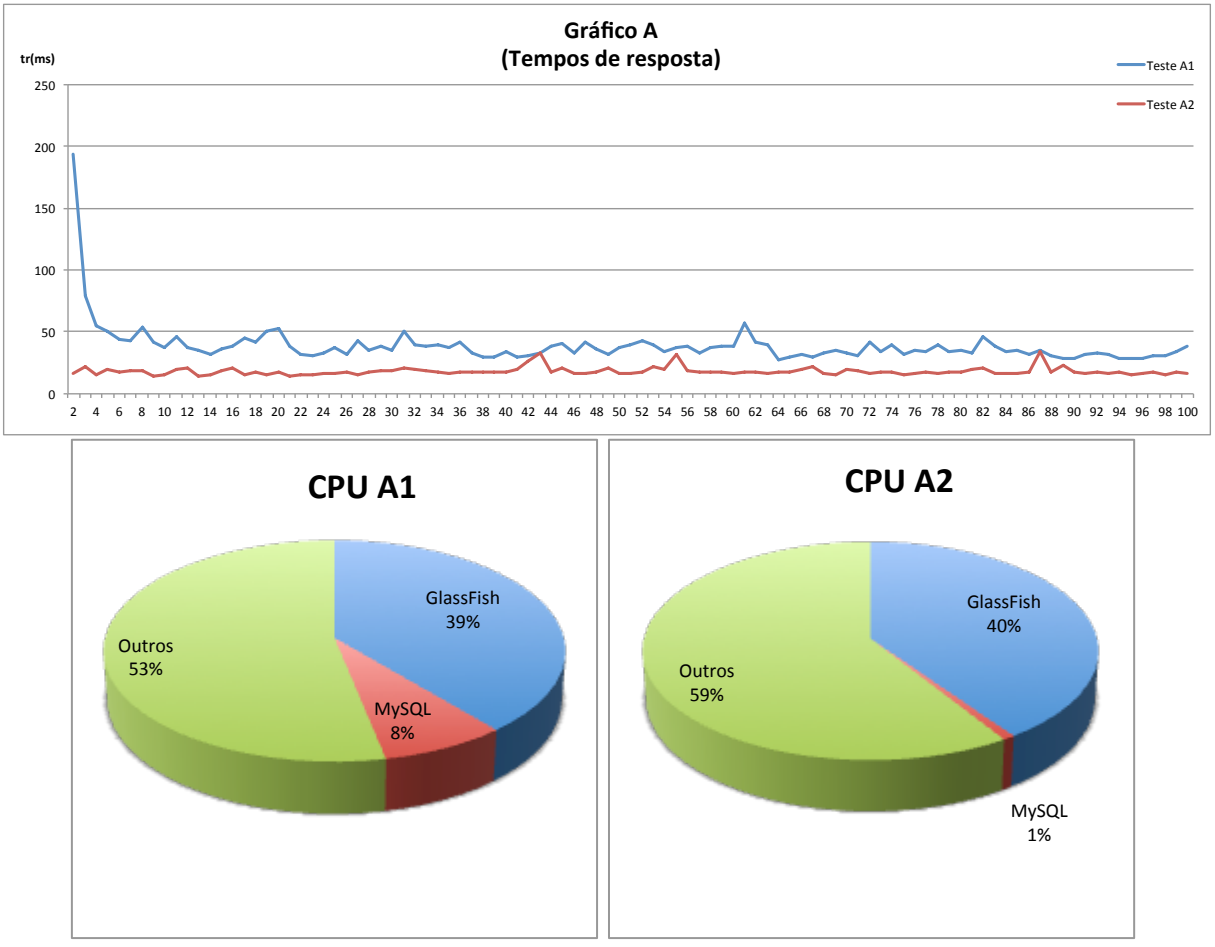


Figura 5.2: Gráficos de tempos de resposta e percentagem de CPU durante os testes A1 e A2.

Teste B

Na execução do teste **B1** foi obtido um tempo de resposta total de 29.988 segundos e um tempo médio de 29.988 milissegundos com uma variância de 6.57^2. Os valores obtidos são semelhantes ao teste A1, pois em ambos os testes a aplicação tem de ler os dados da base de dados. No Gráfigo B da Figura 5.3 pode-se observar que os tempos de resposta no teste B1 e B2-1 tem uma variância semelhante, ao contrário do que aconteceu no teste A. Ao analisar os valores da Tabela 5.2, pode-se constatar que o tempo médio de resposta após o teste B2-1 começa a aumentar. No entanto, se comparamos estes valores com os valores de percentagem de CPU consumidos pelo GlassFish e pelo MySQL, ilustrados nos gráficos da Figura 5.3 e no Gráfico CPU B da Figura 5.4, verificamos que este aumento do tempo de resposta pode eventualmente ser justificado pela subida de carga de CPU do MySQL, uma vez que a percentagem de CPU consumida pelo GlassFish diminuiu.

Tabela 5.2: Valores estatísticos do tempo de resposta para o teste B.

Tempos de Resposta (milissegundos)	**B1**	**B2-1**	**B2-2**	**B2-3**	**B2-4**
Médio	29.988	24.841	26.872	28.186	29.627
Mínimo	13.0	12.0	12.0	12.0	12.0
Máximo	2 564.0	80.0	68.0	110.0	60.0
Desvio Padrão	6.57	5.47	5.94	6.78	6.40
Total (segundos)	29.988	24.841	26.872	28.186	29.627
Total Utilizadores (armazenados no sistema)	995	1989	2978	3960	4945

Analisado os valores do desvio padrão deste teste, que são relativamente aproximados, e tendo em conta que de B1 a B2-4 apenas 0.99% dos utilizadores são repetidos e já estão em memória, a aplicação Ubiacces não apresenta degradação de performance significativa face à quantidade de utilizadores que vão sendo armazenados em memória. Por outro lado, o tempo de resposta médio aumentou, embora que relativamente pouco tendo em conta toda a lógica de controlo de acessos que tem de ser processada. Este aumento do tempo de resposta, permite também identificar um futuro aspecto a ter em consideração caso este sistema venha posteriormente a ser implementado em cenários com milhares de utilizadores com elevada frequência de utilização.

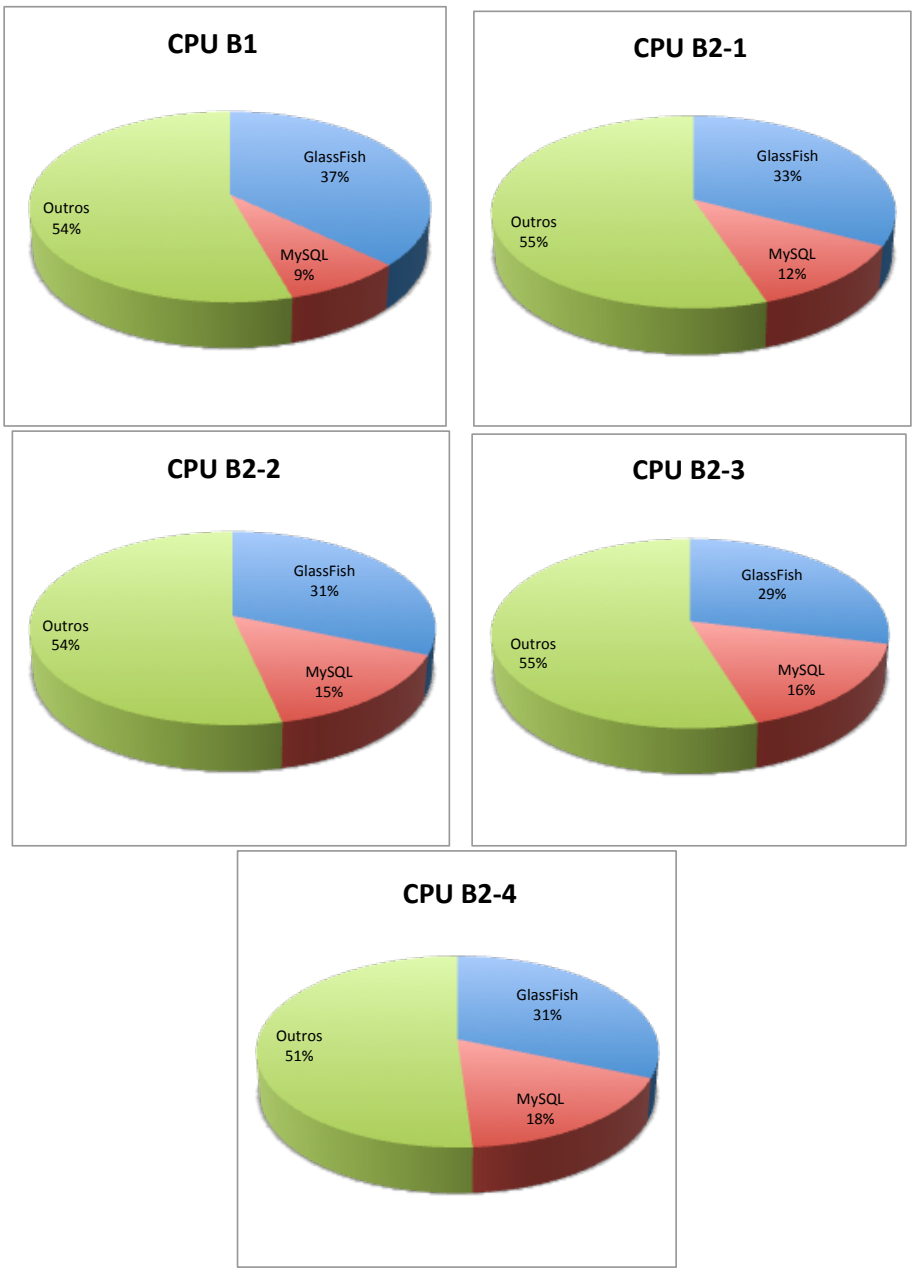


Figura 5.3: Gráficos de percentagem de CPU durante o teste B.

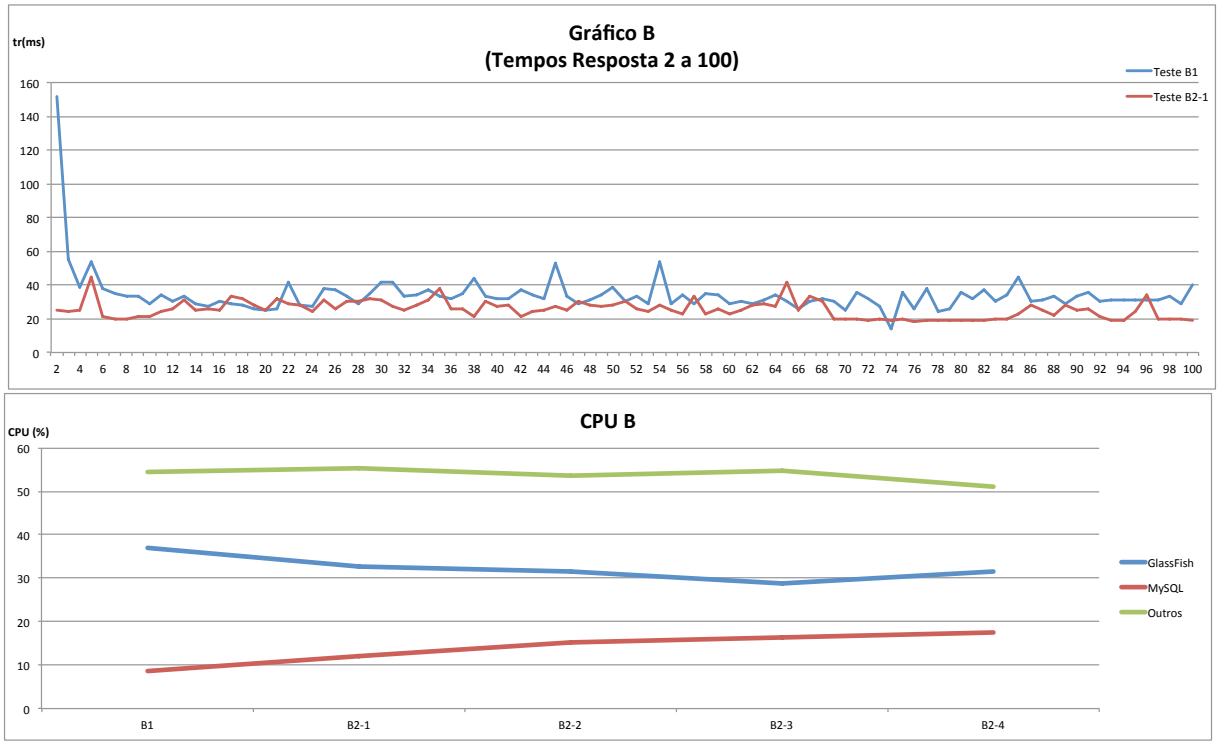


Figura 5.4: Gráficos de tempo de resposta e percentagem de CPU durante o teste B.

5.3.2 Testes ao Módulo Acessos Lógicos

De forma um pouco ilustrativa, nesta secção serão apresentados os testes efetuados sobre a interface de administração do Ubiaccess. Deste modo, pretende-se exemplificar os resultados obtidos após as execuções das ações nas aplicações SVN, LDAP e Redmine ao adicionar novos utilizadores, organizações e projetos ao sistema.

Assim, de forma a simular as três ações mais recorrentes na empresa - criação de utilizadores, criação de projetos e associação de utilizadores a projetos - serão apenas executados os respetivos métodos no webservice de administração do Ubiaccess. Os métodos testados que correspondem a estas três ações são os seguintes:

- **criar utilizador** – *boolean createUserSimple(String username, String email, String password, String rfid, String access_code) throws Exception.*
  Após a inserção do utilizador na base de dados, este método procura na política de acessos

lógicos base da empresa quais as ações a executar. Neste momento, apenas tem configurada a ação CREATE_USER no LDAP[3]. No teste efetuado, este método foi invocado com os parâmetros: *createUserSimple(colaboradorTest, colaboradorTest@ubiwhere.com, colaboradorTest, 8889990000888, 1234)*.

- **criar projeto** – *boolean createProject(String name, String svn_repository, String redmine, String description) throws Exception*.
  Este método ao criar um novo projeto na base de dados do Ubiaccess, automaticamente define esse projeto como super-organização da organização virtual que contém a política de acessos lógicos da empresa. A esta organização virtual está associado a ação CREATE_PROJECT para as aplicações Redmine e SVN, sendo executada nas respetivas aplicações no acto de criação do projeto. O projeto criado neste teste tem os parâmetros: *createProject(Projecto Alfa , /Projects/Univ. Traneeships/Projecto Alfa , 1112ProjectoAlfa , Projecto cirado apenas para efeitos de teste)*.

- **associar utilizador** – *boolean assignUser(String username, String role, String organization) throws Exception*.
  Este método executa todas as ações associadas ao par cargo-organização[4] a que vai associar o utilizador. A invocação deste método foi efetuada com os parâmetros: *assignUser(colaboradorTest , Developer , Projecto Alfa)*.

Criar Utilizador

O utilizador criado para efeitos de teste tinha os seguintes atributos:

name:	*colaboradorTest*;
email:	*colaboradorTest@ubiwhere.com*;
password:	*colaboradorTest*;
rfid:	*8889990000888*;
access_code:	*1234*.

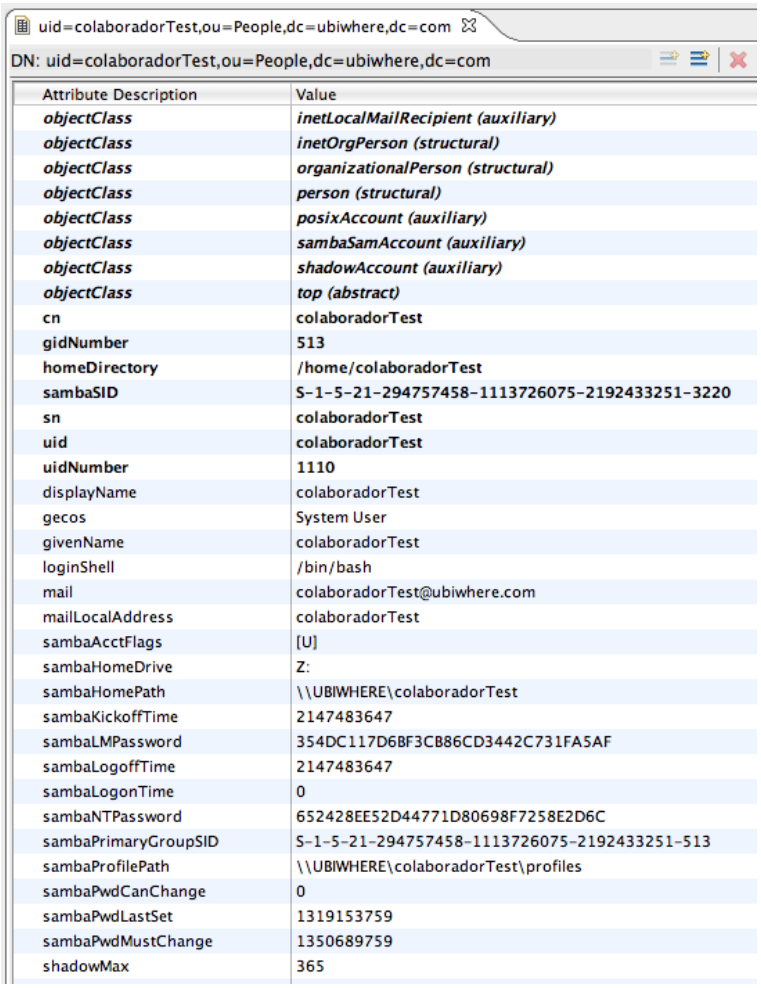
A criação deste utilizador gerou as mensagens de log ilustradas na Figura 5.5. A Figura 5.6 mostra as credências criadas no LDAP pelo Ubiacces para o utilizador colaboradorTest.

[3] A ação CREATE_USER não está também associada à aplicação Redmine porque este está configurado para autenticar e aceitar os utilizadores existentes no LDAP.

[4] Relembrar que a lista de ações/permissões do par *cargo-organização*, através da hierarquia de organizações, contém também as ações que estão associadas a esse cargo nas sub-organizações da *organização*.

```
1 FINEST [2011,10,21 00:35 | LogicalPermissionsSession : find_inherited_logical_permissions ]
2 Getting suborgs of (Role , Org) = (15-CREATE_USER , 3-LogicalAccess Policy Base)
3   INFO [2011,10,21 00:35 | Logical.Administration : createUser ] User colaboradorTest was created in UbiAccess
4 database
5   INFO [2011,10,21 00:35 | Logical.Administration : createUser ] app=LDAP action=CREATE USER
6   INFO [2011,10,21 00:35 | Logical.Administration : createUser ] LDAP :ldap.services.ubiwhere.com
7   INFO [2011,10,21 00:35 | Logical.Administration : createUser ] User colaboradorTest was created in LDAP
```

Figura 5.5: Mensagens de Log durante criação de um utilizador.



uid=colaboradorTest,ou=People,dc=ubiwhere,dc=com

DN: uid=colaboradorTest,ou=People,dc=ubiwhere,dc=com

Attribute Description	Value
objectClass	***inetLocalMailRecipient (auxiliary)***
objectClass	***inetOrgPerson (structural)***
objectClass	***organizationalPerson (structural)***
objectClass	***person (structural)***
objectClass	***posixAccount (auxiliary)***
objectClass	***sambaSamAccount (auxiliary)***
objectClass	***shadowAccount (auxiliary)***
objectClass	***top (abstract)***
cn	**colaboradorTest**
gidNumber	**513**
homeDirectory	**/home/colaboradorTest**
sambaSID	**S-1-5-21-294757458-1113726075-2192433251-3220**
sn	**colaboradorTest**
uid	**colaboradorTest**
uidNumber	**1110**
displayName	colaboradorTest
gecos	System User
givenName	colaboradorTest
loginShell	/bin/bash
mail	colaboradorTest@ubiwhere.com
mailLocalAddress	colaboradorTest
sambaAcctFlags	[U]
sambaHomeDrive	Z:
sambaHomePath	\\UBIWHERE\colaboradorTest
sambaKickoffTime	2147483647
sambaLMPassword	354DC117D6BF3CB86CD3442C731FA5AF
sambaLogoffTime	2147483647
sambaLogonTime	0
sambaNTPassword	652428EE52D44771D80698F7258E2D6C
sambaPrimaryGroupSID	S-1-5-21-294757458-1113726075-2192433251-513
sambaProfilePath	\\UBIWHERE\colaboradorTest\profiles
sambaPwdCanChange	0
sambaPwdLastSet	1319153759
sambaPwdMustChange	1350689759
shadowMax	365

Figura 5.6: Credenciais de um utilizador no LDAP.

Criar Projecto

O projeto criado tem os seguintes atributos:

name:	*Projecto Alfa*;
svn_repository:	*/Projects/Univ. Traneeships/Projecto Alfa*;
redmine:	*1112ProjectoAlfa*;
description:	*Projecto cirado apenas para efeitos de teste*.

As mensagens de Log da Figura 5.8 descrevem alguns dos passos de criação do projeto, em especial na aplicação SVN. O resultado da criação deste projeto no Redmine é ilustrado na Figura 5.7.

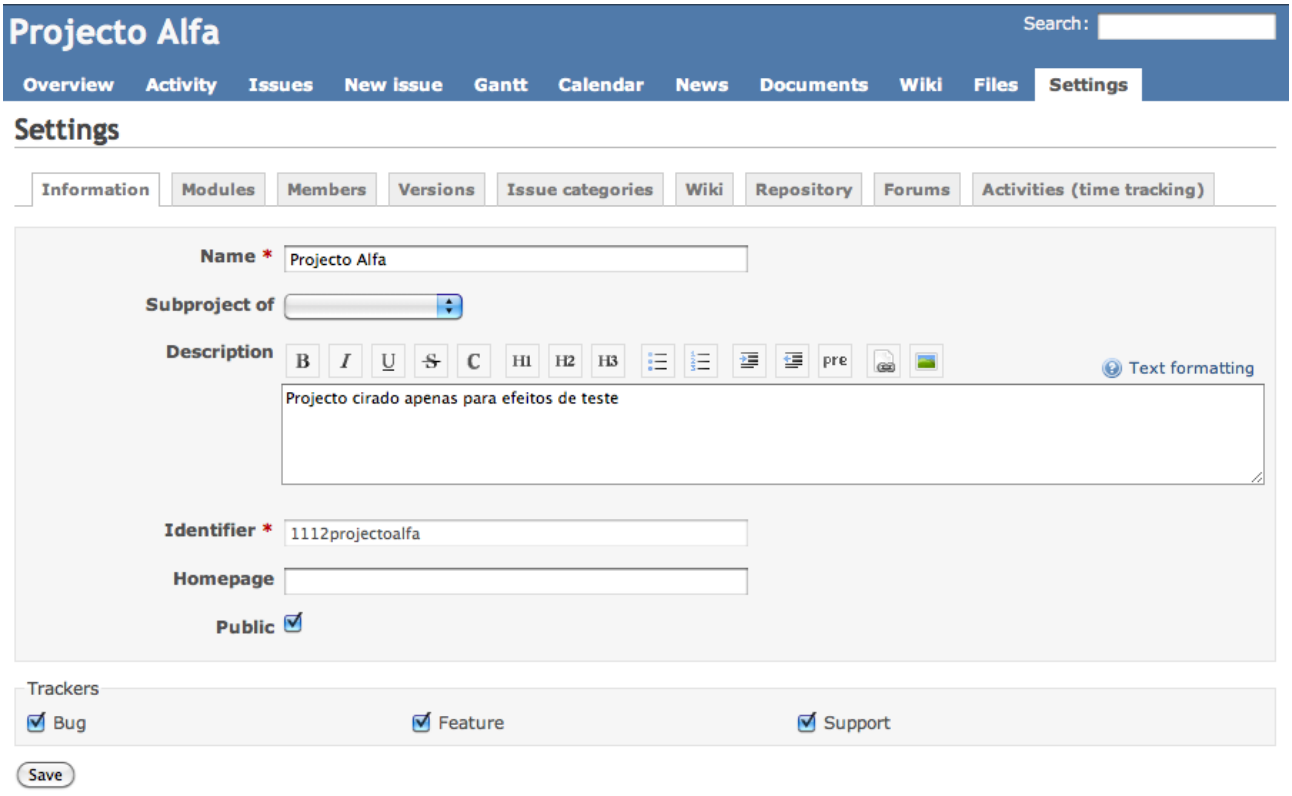


Figura 5.7: Informação base de um projeto no Redmine.

```
1  INFO [2011,10,21 02:46 | Administration : createProject ] name=Projecto Alfa svn_repository=/Projects/Univ.
2 Traneeships/Projecto Alfa redmine=1112ProjectoAlfa description=Projecto cirado apenas para efeitos de teste
3  INFO [2011,10,21 02:46 | LogicalPermissionsSession : create_project ] name=Projecto Alfa svn_repository=/
4 Projects/Univ. Traneeships/Projecto Alfa redmine=1112ProjectoAlfa description=Projecto cirado apenas para efeitos
5 de teste
6  FINE [2011,10,21 02:46 | LogicalPermissionsSession : create_project ] Going to create project
7  INFO [2011,10,21 02:46 | OrganizationSession : createProject ] name=Projecto Alfa svn_repository=/Projects/Univ.
8 Traneeships/Projecto Alfa redmine=1112ProjectoAlfa description=Projecto cirado apenas para efeitos de teste
9  INFO [2011,10,21 02:46 | OrganizationSession : createProject ] p.id=0  p.name=Projecto Alfa
11 INFO [2011,10,21 02:46 | OrganizationSession : createProject ] logicalAccess_policyOrg.id=3
12 logicalAccess_policyOrg.name=Projecto Alfa
13 FINE [2011,10,21 02:46 | LogicalPermissionsSession : create_project ] Project creation returned 186
14 FINEST [2011,10,21 02:46 | LogicalPermissionsSession : find_inherited_logical_permissions ] Getting suborgs of
15 (Role , Org) = (14-CREATION , 3-LogicalAccess Policy Base)
16 FINEST [2011,10,21 02:46 | LogicalPermissionsSession : find_inherited_logical_permissions ] Getting suborgs of
17 (Role , Org) = (14-CREATION , 186-Projecto Alfa)
18 INFO [2011,10,21 02:46 | LogicalPermissionsSession : create_project ] ActionsApplicationsID=3 Action=CREATE
19 PROJECT app=SVN
20 INFO [2011,10,21 02:46 | SoftwareModules.SystemExecs : exec_command ] cp /etc/apache2/svn/svn-projects.access /
21 UbiAccess/svn-projects.access
22 FINEST [2011,10,21 02:46 | SoftwareModules.SVN.SVN : apply_rule ] Action = CREATE PROJECT Organization= Projecto
23 Alfa svn_repository= /Projects/Univ. Traneeships/Projecto Alfa
24 FINEST [2011,10,21 02:46 | SoftwareModules.SVN.SVN : apply_rule ] Linha 318
25 INFO [2011,10,21 02:46 | SoftwareModules.SystemExecs : exec_command ] cp /etc/apache2/svn/svn-projects.access /
26 UbiAccess/svn-projects.access
27 INFO [2011,10,21 02:46 | SoftwareModules.SVN.SVN : create_directory ] svn mkdir -m "new dir for project added" --
28 parents "http://svn2.ubiwhere.com/Projects/Univ. Traneeships/Projecto Alfa" --username ubiaccess --password ******
29 FINEST [2011,10,21 02:46 | SoftwareModules.SVN.svnPermissionsManager : save_to_file ] Output for Access_File
30 [groups]
31 admin = aoliveira,rcosta,nribeiro
32 [...]
33 [/Projects/Univ. Traneeships/Projecto Alfa]
34 @admin = rw
35 INFO [2011,10,21 02:46 | SoftwareModules.SystemExecs : exec_command ] cp /UbiAccess/svn-projects.access /etc/
36 apache2/svn/svn-projects.access
37 INFO [2011,10,21 02:46 | LogicalPermissionsSession : create_project ] ActionsApplicationsID=6 Action=CREATE
38 PROJECT app=Redmine
39 FINEST [2011,10,21 02:46 | SoftwareModules.REDMINE.Redmine : apply_rule ] Action = CREATE PROJECT Organization=
40 Projecto Alfa
41 FINEST [2011,10,21 02:46 | SoftwareModules.REDMINE.Redmine : apply_rule ] Creating Project: name = Projecto Alfa
42 redmine1112ProjectoAlfa
43 FINEST [2011,10,21 02:46 | SoftwareModules.REDMINE.Redmine : apply_rule ] Creating Project: name = Projecto Alfa
44 redmineIdentifier= 1112projectoalfa
```

Figura 5.8: Mensagens de Log durante criação de um projeto.

Associar Utilizador

Após ser criado o utilizador e o projeto, esse utilizador foi associado ao projeto com o cargo *Developer*. Assim sendo, foi invocado o método *assignUser(colaboradorTest, Developer, Projecto Alfa)*. O resultado desta operação é a associação do utilizador ao projeto no SVN à respetiva diretoria, como ilustrado na Figura 5.9.

```
1   INFO [2011,10,21 02:59 | Logical.Administration : assignUser ] Parameters: User=colaboradorTest, Role=Developer,
2  Organization=Projecto Alfa
3  FINEST [2011,10,21 02:59 | LogicalPermissionsSession : find_inherited_logical_permissions ] Geting suborgs of
4  (Role , Org) = (5-Developer , 186-Projecto Alfa)
5   INFO [2011,10,21 02:59 | LogicalPermissionsSession : assign_user_to_project ] ActionsApplicationsID=1 Action=READ
6  app=SVN
7   INFO [2011,10,21 02:59 | SoftwareModules.SystemExecs : exec_command ] cp /etc/apache2/svn/svn-projects.access /
8  UbiAccess/svn-projects.access
9  FINEST [2011,10,21 02:59 | SoftwareModules.SVN.SVN : apply_rule ] Action = READ Organization= Projecto Alfa
11 svn_repository= /Projects/Univ. Traneeships/Projecto Alfa
12 FINEST [2011,10,21 02:59 | SoftwareModules.SVN.svnPermissionsManager : save_to_file ] Output for Access_File
13 [groups]
14 admin = aoliveira,rcosta,nribeiro
15 [...]
16 [/Projects/Univ. Traneeships/Projecto Alfa]
17 @admin = rw
18 colaboradorTest = r
19  INFO [2011,10,21 02:59 | SoftwareModules.SystemExecs : exec_command ] cp /UbiAccess/svn-projects.access /etc/
20 apache2/svn/svn-projects.access
21  INFO [2011,10,21 02:59 | Data.DataSessionLogical.LogicalPermissionsSession : assign_user_to_project ]
22 ActionsApplicationsID=2 Action=WRITE app=SVN
23  INFO [2011,10,21 02:59 | SoftwareModules.SystemExecs : exec_command ] cp /etc/apache2/svn/svn-projects.access /
24 UbiAccess/svn-projects.access
25 FINEST [2011,10,21 02:59 | SoftwareModules.SVN.SVN : apply_rule ] Action = WRITE Organization= Projecto Alfa
26 svn_repository= /Projects/Univ. Traneeships/Projecto Alfa
27 FINEST [2011,10,21 02:59 | SoftwareModules.SVN.svnPermissionsManager : save_to_file ] Output for Access_File
28 [groups]
29 admin = aoliveira,rcosta,nribeiro
30 [...]
31 [/Projects/Univ. Traneeships/Projecto Alfa]
32 @admin = rw
33 colaboradorTest = rw
34  INFO [2011,10,21 02:59 | SoftwareModules.SystemExecs : exec_command ] cp /UbiAccess/svn-projects.access /etc/
35 apache2/svn/svn-projects.access
36
```

Figura 5.9: Mensagens de Log durante associação de um utilizador a um Projeto.

5.4 Sumário

Neste capítulo foi apresentada a metodologia usada nos testes realizados à aplicação desenvolvida e avaliados os resultados obtidos. Estes testes incidiram mais no módulo de controlo de acessos físicos abordando o seu desempenho numa perspectiva geral. Tendo em conta os objetivos iniciais do trabalho e o real cenário de utilização do *Ubiaccess* face aos testes a que foi submetido, os resultados obtidos podem ser considerados bons. Tal como foi abordado ao longo da descrição dos testes, foram identificados alguns aspetos e pontos de melhoria do desempenho do sistema a ter em conta quando este seja aplicado a cenários de maior escala com milhares de utilizadores.

Capítulo 6

Conclusões

Após a apresentação e explicação de todo o trabalho desenvolvido, neste capítulo serão agora abordadas as conclusões finais do trabalho realizado. Será feita uma síntese das principais etapas do trabalho, seguindo-se uma reflexão sobre as principais contribuições resultantes e possíveis tópicos de trabalho futuro.

6.1 Resumo do Trabalho Desenvolvido

De forma a ter uma base de conhecimento para a implementação do sistema de controlo de acessos, foram estudados vários modelos de controlo de acessos já existentes. Após este estudo, face ao âmbito e objetivos deste trabalho, inspirado num dos modelos já existentes foi criado e apresentado nesta dissertação um novo modelo de controlo de acessos. De seguida, foi efetuado um estudo das tecnologias de autenticação física. A tecnologia adotada foi o RFID, pois permite autenticar o utilizador sem haver contacto físico, nem linha de vista com leitor o RFID, sendo possível colocá-lo dentro das instalações da empresa.

O desenho e implementação da solução foram efetuados com a estrutura mais modular e independente possível, de forma a que no futuro facilmente possam ser desenvolvidos novos módulos, aumentando a compatibilidade do *Ubiaccess* com outras aplicações/tecnologias. A arquitetura e implementação do *Ubiaccess* no módulo de controlo de acessos lógicos, foi especialmente orientada a aplicações que careçam da implementação de protocolos de controlos de acesso.

No módulo de controlo de acessos físicos, de forma a reduzir o risco dos casos em que um cartão RFID pode ser perdido ou roubado, foi utilizado um mecanismo complementar de auten-

ticação caso o utilizador seja o primeiro a aceder às instalações. Este mecanismo consiste no facto do sistema automaticamente fazer uma chamada para o utilizador, para ele introduzir o seu código pessoal de acesso.

6.2 Principais Contribuições

As contribuições deste trabalho dividem-se em três pontos chave:

- **novo modelo de acessos** – foi criado e apresentado um novo modelo de controlo de acessos, que se distingue dos anteriores por associar as permissões ao par cargo-organização. Desta forma, permite às organizações e departamentos darem privilégios diferentes ao mesmo cargo de acordo com a sua política interna. Outra grande vantagem deste modelo, dada a sua estrutura, é o facto deste ter a capacidade de representar e simular outros modelos de acesso, como por exemplo o *Role Based Access Control* e *Group Based Acess Control*, entre outros. Assim, pode ser utilizado para unificar e convergir vários modelos numa só política de acessos.

- **acessos físicos** – a combinação do uso da tecnologia RFID mais um código de acesso através de telemóvel na autenticação de utilizadores é uma nova abordagem. Desta forma, ao colocar o leitor RFID do lado de dentro das instalações, conseguiu-se conceber um mecanismo de autenticação anti-vandalismo, para além dos equipamentos estarem resguardados de condições climatéricas adversas.

- **acessos lógicos** – neste trabalho foi apresentada uma abordagem para o controlo de acessos lógicos diferente e contrária às soluções existentes. Uma vez que as aplicações SVN e Redmine, entre outras utilizadas pela empresa, não implementam qualquer protocolo nem mecanismo de forma a consultarem a política de acessos de um sistema externo (*Policy Decision Point*), foi adotada a metodologia inversa. Ou seja, a aplicação de controlo de acessos desenvolvida neste trabalho reconfigura automaticamente as outras aplicações e serviços da empresa de forma a que estas espelhem uma política global e coesa.

Em resultado da relevância das novas abordagens deste trabalho, foi publicado o artigo *A Unifying Role and Organization Based Access Control* [15], sendo o mesmo apresentado na conferência sobre redes de computadores CRC2010.

6.3 Trabalho Futuro

Em relação ao modelo de controlo de acessos, uma possível melhoria será o suporte de recomendações e a possibilidade de se conceder permissões caso estas recomendações estabelecidas na política tenham sido cumpridas. Quanto ao módulo de acessos físicos, tal como foi identificado no capítulo dos testes, poderá haver necessidade de se efetuar algumas melhorias de escalabilidade para cenários de grande dimensão. Outro tópico é tornar este módulo capaz de suportar também a autenticação por biometrias.

No módulo de acessos lógicos, ainda existem algumas melhorias a ser efetuadas - nomeadamente no módulo do Redmine - devido a limitações da sua atual API em relação às funcionalidades da aplicação. Seria bastante enriquecedor para a aplicação implementar o suporte de um protocolo como o eXtensible Access Control Markup Language (XACML), pois permitiria uma mais fácil comunicação com outros sistemas.

Bibliografia

[1] R. S. Sandhu, "Lattice-Based Access Control Models," *Computer*, vol. 26, pp. 9–19, 1993.

[2] R. C. David F. Ferraiolo, D. Richard Kuhn, *Role-Based Access Control*. Artech House, second ed., 2007.

[3] Thomas, Sandhu, and R. K. Thomas, "Task-based Authorization Controls (TBAC): A family of Models for Active and Enterprise-oriented Authorization management," in *Proceedings of the IFIP WG11.3 Workshop on Database Security, Lake Tahoe*, pp. 166–181, 1997.

[4] R. Sandhu, D. Ferraiolo, and R. Kuhn, "The NIST Model for Role-Based Access Control: Towards A Unified Standard," in *Proceedings of the fifth ACM workshop on Role-based access control*, pp. 47–63, 2000.

[5] D. Ferraiolo and R. Kuhn, "Role-Based Access Control," in *15th NIST-NCSC National Computer Security Conference*, pp. 554–563, 1992.

[6] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[7] A. A. E. Kalam, S. Benferhat, A. Miège, R. E. Baida, F. Cuppens, C. Saurel, P. Balbiani, Y. Deswarte, and G. Trouessin, "Organization Based Access Control," in *POLICY '03: Proceedings of the 4^{th} IEEE International Workshop on Policies for Distributed Systems and Networks*, (Washington, DC, USA), pp. 120–131, IEEE Computer Society, 2003.

[8] T. Moses, "eXtensible Access Control Markup Language (XACML)," tech. rep., OASIS, 2005.

[9] "eXtensible Access Control Markup Language. Documents Repository." `http://www.oasis-open.org/committees/xacml/`, [online August 2011].

[10] "XML Access Control Language (XACL)." `http://xml.coverpages.org/xacl.html`, [online August 2011].

[11] D. L. R. Jothy Rosenberg, *Securing Web Services with WS-Security*. Sams Publishing, May 2004.

[12] "Open PREMIS." `http://www.openpermis.org/`, [online August 2011].

[13] C. . (APQ), "Sistemas de Gestão da Qualidade," tech. rep., Instituto Português da Qualidade, November 2008.

[14] C. . (IPQ), "Gestão da Investigação, Desenvolvimento e Inovação (IDI)," tech. rep., Instituto Português da Qualidade, January 2007.

[15] J. Novais, N. Ribeiro, and P. Sousa, "A Unifying Role and Organization Based Access Control," in *Proceedings of CRC'2010 - 10ª Conferência sobre Redes de Computadores*, pp. 119–124, 2010.

[16] "ANSI ® INCITS 359-2004; American National Standard for Information Technology - Role Based Access Control," February 2004.

[17] Z. Zhang, X. Zhang, and R. Sandhu, "ROBAC: Scalable Role and Organization Based Access Control Models," *in International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pp. 1–9, 2006.

[18] C. do Cidadão, "O novo documento de identificação dos cidadãos portugueses. (nota informativa-1)," Março 2007.

[19] "Microsoft CryptoAPI." `http://msdn.microsoft.com/en-us/library/aa380255(v=VS.85).aspx`, [online August 2011].

[20] "PKCS Cryptographic Token Interface Standard," tech. rep., RSA Laboratories, June 2004.

[21] K. E. Mayes and K. Markantonakis, *Smart Cards, Tokens, Security and Applications*. Springer, 2008.

[22] "An overview for Companies Seeking to use RFID Technology to Connect their IT Systems Directly to the "Real" World; RFID White Paper Technology, Systems, and Applications." BITKOM - German Association for Information Technology, 2005.

[23] S. E. Sarma, S. A. Weis, and D. W. Engels, "RFID Systems and Security and Privacy Implications," in *Workshop on Cryptographic Hardware and Embedded Systems*, pp. 454–470, Springer, 2002.

[24] A. Juels, D. Molnar, and D. Wagner, "Security and Privacy Issues in E-passports," in *First International Conference on Security and Privacy for Emerging Areas in Communications Networks*, pp. 74–88, 2005.

[25] E. Inc, "Tag Class Defenitions," tech. rep., EPCglobal Inc, November 2007.

[26] J. R. W. Stephen B. Miles, Sanjay E. Sarma, *RFID Technology and Applications*. Cambridge University Press, second ed., 2008.

[27] M. Indovina, U. Uludag, R. Snelick, A. Mink, and A. Jain, "Multimodal Biometric Authentication Methods: A COTS Approach," in *Proceedings of MMUA 2003, Workshop on Multimodal User Authentication*, pp. 99–106, 2003.

[28] N. Clarke, *Transparent User Aunthentication. Biometrics, RFID and Behavioural Profiling*. Springer, 2011.

[29] D. Berlin and G. Rooney, *Practical Subversion*. Berkely, CA, USA: Apress, second ed., 2006.

[30] "Redmine - Project Management Web Application." `http://www.redmine.org/projects/redmine/wiki`, [online August 2011].

[31] E. J. Sermersheim, "Lightweight Directory Access Protocol (LDAP): The Protocol." RFC4511, August 2006.

[32] "OpenLDAP." `http://www.openldap.org/`, [online July 2011].

[33] A. Goncalves, *Beginning Java EE 6 Platform with GlassFish 3*. Apress, 2008.

[34] "GlassFish." http://glassfish.java.net/downloads/3.0.1-final.html, [online May 2011].

[35] "Oracle's Support for Open Source and Open Standards." `http://www.oracle.com/us/technologies/open-source/index.html`, [online August 2011].

[36] "MySQL." `http://www.mysql.com/why-mysql/benchmarks/`, [online May 2011].

[37] D. J. Blezard and J. Marceau, "One user, one password: Integrating unix accounts and active directory," in *Proceedings of the 30th annual ACM SIGUCCS Conference on User services, SIGUCCS '02*, pp. 5–8, ACM, 2002.

[38] "Linux-HA." `http://www.linux-ha.org/wiki/Main_Page`, [online May 2011].

[39] "Pacemaker." `http://www.clusterlabs.org/`, [online May 2011].

[40] S. P. H. Schulzrinne, "RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals." RFC2833, 2000.

[41] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol." RFC3261, 2002.

[42] L. Veltri, "MjSip." `http://www.mjsip.org/`, [online May 2010].

[43] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications." RFC3550, 2003.

[44] "Redmine & Chiliproject Java API." `http://code.google.com/p/redmine-java-api/`, [online August 2011].

[45] "cURL." `http://curl.haxx.se/`, [online October 2011].

[46] E. H. Halili, *Apache JMeter. A practical beginner's guide to automated testing and performance measurement for your websites*.
Packt Publishing, 2008.

[47] "Apache JMeter." `http://jakarta.apache.org/jmeter/`, [online October 2011].