INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

BIBLIOTECA PARA A REALIZAÇÃO DE SECURITY TOKEN SERVICES

José Henrique Fernandes de Figueiredo
(Bacharel em Engenharia Informática e de Computadores)

TRABALHO DE PROJECTO PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA INFORMÁTICA E DE COMPUTADORES

Orientador:
Pedro Miguel Henriques dos Santos Félix

Júri:
Manuel Martins Barata
Vítor Jesus Sousa de Almeida
José Luís Falcão Cascalheira

DEZEMBRO DE 2008

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Departamento de Engenharia de Electrónica e Telecomunicações e de Computadores

BIBLIOTECA PARA A REALIZAÇÃO DE SECURITY TOKEN SERVICES

José Henrique Fernandes de Figueiredo
(Bacharel em Engenharia Informática e de Computadores)

TRABALHO DE PROJECTO PARA OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA INFORMÁTICA E DE COMPUTADORES

Orientador:
Pedro Miguel Henriques dos Santos Félix

Júri:
Manuel Martins Barata
Vítor Jesus Sousa de Almeida
José Luís Falcão Cascalheira

DEZEMBRO DE 2008

Resumo

A família de especificações WS-* define um modelo de segurança para *web services*, baseado nos conceitos de *claim*, *security token* e *Security Token Service* (STS). Neste modelo, a informação de segurança dos originadores de mensagens (identidade, privilégios, etc.) é representada através de conjuntos de *claims*, contidos dentro de *security tokens*. A emissão e obtenção destes *security tokens*, por parte dos originadores de mensagens, são realizadas através de protocolos legados ou através de serviços especiais, designados de *Security Token Services*, usando as operações e os protocolos definidos na especificação *WS-Trust*.

O conceito de *Security Token Service* não é usado apenas no contexto dos *web services*. Propostas como o modelo dos *Information Cards*, aplicável no contexto de aplicações *web*, também utilizam este conceito.

Os *Security Token Services* desempenham vários papéis, dependendo da informação presente no *token* emitido. São exemplos o papel de *Identity Provider*, quando os *tokens* emitidos contêm informação de identidade, ou o papel de *Policy Decision Point*, quando os *tokens* emitidos definem autorizações.

Este documento descreve o projecto duma biblioteca software para a realização de *Security Token Services*, tal como definidos na norma *WS-Trust*, destinada à plataforma .NET 3.5.

Propõem-se uma arquitectura flexível e extensível, de forma a suportar novas versões das normas e as diversas variantes que os *Security Token Services* possuem, nomeadamente: o tipo dos *security token* emitidos e das *claims* neles contidas, a inferência das *claims* e os métodos de autenticação das entidades requerentes.

Apresentam-se aspectos de implementação desta arquitectura, nomeadamente a integração com a plataforma WCF, a sua extensibilidade e o suporte a modelos e sistemas externos à norma.

Finalmente, descrevem-se as plataformas de teste implementadas para a validação da biblioteca realizada e os módulos de extensão da biblioteca para: suporte do modelo associado aos *Information Cards*, do modelo *OpenID* e para a integração com o *Authorization Manager*.

Palavras-chave

Security Token Service, *WS-**, *WS-Trust*, WCF, *web services*, sistemas distribuídos, *Information Cards*, *OpenID*, *Authorization Manager*

Abstract

The WS-* family of standards defines a security model for web services, based on the concepts of claims, security tokens and Security Token Services (STS). In this model, the message originator's security information (identity, privileges, etc.) is represented by claim sets, contained within security tokens. The issuance and obtainance of security tokens, by the originators of the messages, is carried out by legacy protocols or by special services, called Security Token Services, using the operations and protocols defined in the WS-Trust standard.

The concept of Security Token Services is not used only in the context of web services. Proposals such as the Information Cards model, applicable to web applications, also use this concept.

Security Token Services play several roles, depending on the information contained in the issued token. Examples include the role of Identity Provider, when the issued tokens contain identity information, or the role of Policy Decision Point, when the issued tokens define authorizations.

This document describes a software library project for the implementation of Security Token Services, as defined in the WS-Trust standard, and targeted for the .NET 3.5 platform.

It proposes a flexible and extensible architecture, in order to support new versions of the standards and the different variants that Security Token Services have, mainly the issued security token type and its contained claims, the claim inference process and the requestor's authentication methods.

This document also presents implementation aspects for this architecture, mainly the integration with the WCF platform, its extensibility and the support of external systems and models.

Finally, it describes the implemented test platforms for validating the library, and also the library's extension modules supporting the Information Card model, the OpenID model and the integration with the Authorization Manager.

Key-words

Security Token Service, WS-*, WS-Trust, WCF, web services, distributed systems, Information Cards, OpenID, Authorization Manager

Agradecimentos

Em primeiro lugar, gostaria de agradecer ao meu orientador, o Professor Pedro Félix, que contribuiu activamente para a realização deste trabalho, apoiando-me incondicionalmente, dando-me diversos ensinamentos e conselhos úteis e mostrando sempre grande disponibilidade e paciência.

Gostaria de agradecer à Weronika toda a sua paciência e compreensão pelo tempo que sacrificámos e pelo apoio constante.

Índice

1 Introdução

1.1 Âmbito e objectivo do documento

Esta dissertação insere-se no âmbito do Trabalho de Projecto do Mestrado em Engenharia Informática e de Computadores (MEIC) do Instituto Superior de Engenharia de Lisboa (ISEL). Apresenta aspectos de desenho e implementação duma biblioteca software para a realização de *Security Token Services*.

1.2 Âmbito e objectivo do trabalho

Na última década, a adopção de arquitecturas e tecnologias distribuídas têm vindo a generalizar-se. A normalização e as características distribuídas dos *web services* levou à sua larga aceitação para a implementação destas arquitecturas. Contudo, os aspectos de segurança não foram considerados nas especificações iniciais dos *web services*.

O conceito de identidade e o controlo de acessos são aspectos fundamentais na implementação da segurança nos sistemas distribuídos. A aplicação das relações de confiança entre sistemas organizacionais distribuídos implica a existência de um modelo único na forma como a identidade é definida, trocada e manipulada. A ausência de um modelo único levou à adopção de soluções distintas e ao consequente isolamento das organizações e dos seus sistemas.

A família de especificações *WS-** [1] define um modelo de segurança para *web services*, baseado nos conceitos de *claim*, *security token* e *Security Token Service* (STS). Neste modelo, a informação de segurança dos originadores de mensagens (identidade, privilégios, etc.) é representada através de conjuntos de *claims*, contidos dentro de *security tokens*. A emissão e obtenção destes *security tokens*, por parte dos originadores de mensagens, são realizadas através de protocolos legados ou através de serviços especiais, designados de *Security Token Services*, usando as operações e os protocolos definidos na especificação *WS-Trust* [1][3].

O conceito de *Security Token Service* não é usado apenas no contexto dos *web services*. Propostas como o modelo dos *Information Cards*, aplicável no contexto de aplicações *web*, também utilizam este conceito.

Os *Security Token Service* desempenham vários papéis, dependendo da informação presente no *token* emitido. São exemplos o papel de *Identity Provider*, quando os *tokens* emitidos contêm informação de identidade, ou o papel de *Policy Decision Point*, quando os *tokens* emitidos definem autorizações.

Este documento descreve o projecto duma biblioteca software (*STSLib*)[1] para a realização de *Security Token Services*, tal como definidos na norma *WS-Trust*, destinada à plataforma .NET 3.5.

[1] Designou-se por *STSLib* a biblioteca software para a realização de *Security Token Services*

Propõem-se uma arquitectura flexível e extensível, de forma a suportar novas versões das normas e as diversas variantes que os *Security Token Services* possuem, nomeadamente: tipo dos *security token* emitidos e das *claims* neles contidas, a inferência das *claims* e os métodos de autenticação das entidades requerentes.

Apresentam-se aspectos de implementação desta arquitectura, nomeadamente a integração com a plataforma WCF, a sua extensibilidade e o suporte a modelos e sistemas externos à norma.

Finalmente, descrevem-se as plataformas de teste implementadas para a validação da biblioteca realizada e os módulos de extensão da biblioteca para: suporte do modelo associado aos *Information Cards*, do modelo *OpenID* e para a integração com o *Authorization Manager*.

1.3 Estrutura do documento

- No capítulo 2 apresentam-se as normas e os modelos associados ao conceito de STS, evidenciando a sua importância na implementação de sistemas distribuídos. Este capítulo apresenta também soluções alternativas ou relacionadas ao conceito de STS.
- No capítulo 3 é apresentada a motivação deste projecto e quais os objectivos pretendidos.
- No capítulo 4 é descrita a arquitectura da *STSLib*, nomeadamente: a arquitectura base, a sua extensibilidade e as justificações para as opções tomadas.
- No capítulo 5 são descritos os aspectos da implementação da arquitectura proposta. Também se detalha a arquitectura implementada para a inclusão de extensões, usada na flexibilização e extensão da *STSLib* para futuras normas e diferentes contextos e aplicações.
- No capítulo 6 apresentam-se as plataformas de teste implementadas para a verificação do bom funcionamento das várias concretizações da *STSLib*, e quais os objectivos específicos de cada uma das verificações realizadas.
- No capítulo 7 são descritos os aspectos da implementação da aplicação *web* de gestão do STS (*STSMan*)[2] e das extensões não consideradas para a implementação base, nomeadamente as extensões para os modelos ISIP [24][25][26] e *OpenID* [27] e a integração com o *Authorization Manager* [30].
- No capítulo 8 apresentam-se as conclusões deste projecto e perspectivam-se direcções para trabalho futuro.
- Em Apêndice é apresentada uma síntese da plataforma *Windows Communication Foundation*, que serve de base à biblioteca realizada.

[2] Designou-se por *STSMan* a aplicação *web* de gestão do STS

2 Contexto e Enquadramento

O presente capítulo tem por objectivo apresentar as normas e os modelos associados ao conceito de *Security Token Service*, evidenciando a sua importância na implementação de sistemas distribuídos. Apresenta também soluções alternativas ou relacionadas ao conceito de STS.

2.1 Normas *WS-** para *web services*

Em 2002, na sequência de trabalho conjunto da IBM e da Microsoft, foi apresentado o documento [4] propondo uma estratégia para a segurança em *web services*. Nesse documento é definido um modelo que suporta, integra e unifica vários modelos, mecanismos e tecnologias de segurança, e foi com base nesse documento que se iniciou o processo de especificação das normas *WS-**.

As normas *WS-** providenciam um conjunto de especificações para a protecção de mensagens SOAP, visando a sua integridade, confidencialidade e a autenticação de origem. Estas especificações, nomeadamente a especificação *WS-Security* [5][7][8][9], definem um modelo de autenticação baseado em *security tokens*, *claims* e assinaturas digitais.

Um *security token* representa um conjunto de *claims*, que são declarações genéricas (nomes, *roles*, etc.) efectuadas por *Identity Providers* (IP). Sendo um conceito abstracto, o *security token* permite múltiplas concretizações, tais como bilhetes *Kerberos* [10] e asserções SAML (*Security Assertion Markup Language*) [11][12][13].

Os *Identity Providers* são entidades que emitem asserções sobre as identidades digitais dos sujeitos, em forma de *security tokens* contendo um conjunto de *claims*, e associam-nas a chaves de verificação. O processo de emissão de um *security token* poderá ser efectuado por um protocolo exterior às normas em discussão, tal como o protocolo *Kerberos*, ou através de um serviço próprio denominado *Security Token Service*, e cuja especificação está definida na norma *WS-Trust*. A autenticidade da associação das chaves de verificação com a mensagem, e do conjunto de *claims* com o seu emissor, é garantida através de assinaturas digitais. Um exemplo concreto de *security tokens* assinados e subscritos por uma autoridade de certificação são certificados X.509 [14], através dos quais se associam nomes X.500 [15] a chaves públicas.

2.1.1 *WS-Policy* e *WS-SecurityPolicy*

A norma *WS-Policy* [16] define um modelo flexível e extensível para a caracterização das capacidades, requisitos e características gerais de entidades pertencentes a sistemas baseados em *web services*. Estas características são definidas como políticas. Uma política é uma colecção de alternativas, uma alternativa é uma colecção de asserções e uma asserção representa um requisito ou uma capacidade.

A norma *WS-SecurityPolicy* [17] define asserções concretas para o domínio da segurança, caracterizando especificamente os tipos de *tokens*, algoritmos criptográficos e quais os mecanismos a usar.
O objectivo destas normas é providenciar aos participantes informação suficiente para a determinação da compatibilidade e da interoperabilidade.

2.1.2 *WS-Trust*

A norma *WS-Trust* estende a norma *WS-Security*, definindo operações para a emissão, renovação e validação de *security tokens*. Para a realização destas operações a norma introduz um novo tipo de *web service* designado por *Security Token Service*.
O modelo (Figura 2.1) apresentado pela norma *WS-Trust* especifica que um *web service* poderá requerer, na recepção de uma mensagem, a prova de um conjunto de *claims*. Este modelo é concretizado através da participação de um STS e da autenticação do requente perante o STS.
Os pedidos ao STS são efectuados através do envio da mensagem *Request Security Token* (RST). O RST contém o tipo de *token* a emitir e as *claims* a processar. A autenticidade dos originadores da mensagem RST é conseguida através da associação de *tokens* a essas mensagens. O STS usa as *claims* do originador da mensagem RST para decidir, com base na política de emissão, quais as *claims* a emitir. O novo *token* é devolvido dentro de uma mensagem *Request Security Token Response* (RSTR).

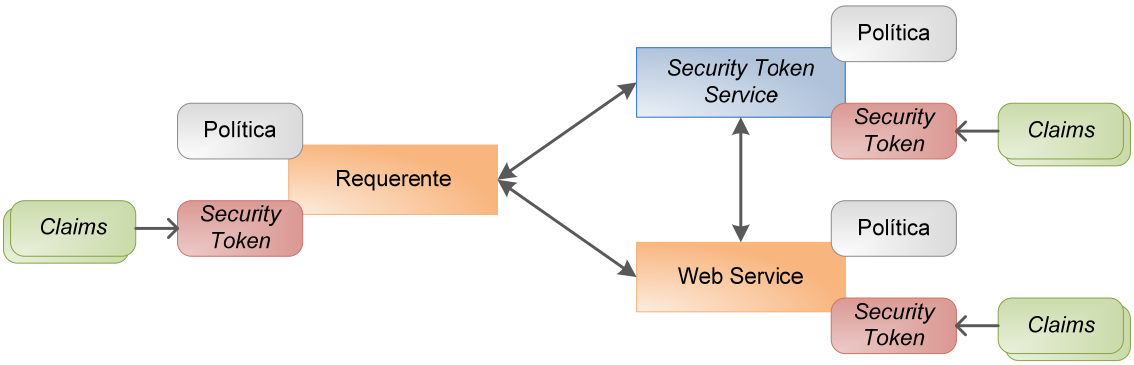

Figura 2.1 – Modelo *WS-Trust*[3]

2.1.3 *WS-Federation*

A norma *WS-Federation* [18] acrescenta às normas *WS-**, e especificamente à norma *WS-Trust*, uma arquitectura para a federação de identidades.
Esta especificação define um conjunto de mecanismos para a interoperabilidade de diferentes domínios de segurança, caracterizando como é que acessos autorizados a recursos localizados num domínio possam ser disponibilizados a entidades cujas identidades são geridas noutro domínio.

[3] Baseado na figura *Web Service Trust Model* dos documentos [1][3]

Um dos objectivos fundamentais desta norma é simplificar o desenvolvimento de serviços federados através da reutilização do modelo definido na norma *WS-Trust*. Com base nestas especificações e no conceito de *Security Token Service*, é possível implementar um conjunto diversificado de soluções e topologias de federação.

Os clientes do tipo *web browser* tipicamente não têm a capacidade de aplicar esquemas criptográficos nem de executar as operações definidas na norma *WS-Trust*. Por esse motivo, esta norma define um protocolo baseado no protocolo HTTP, designado por perfil passivo, para a integração desses clientes.

Uma das possíveis topologias definidas pela norma é apresentada na Figura 2.2.

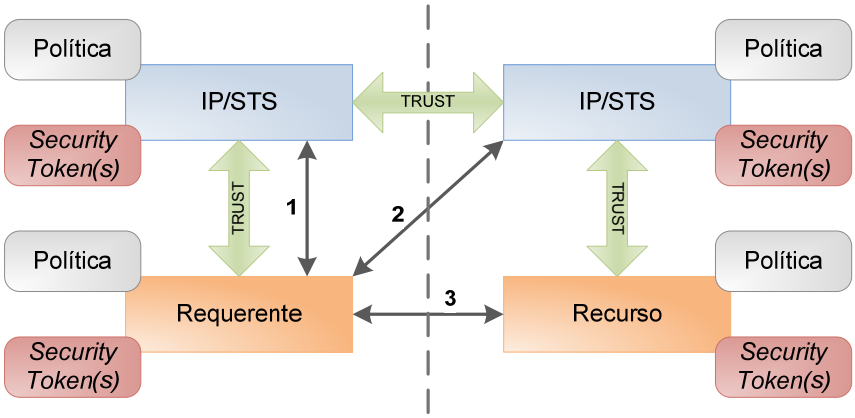

Figura 2.2 – Modelo *WS-Federation* e *WS-Trust*[4]

2.2 *Identity Metasystem*

O conceito de identidade é um conceito fundamental na maioria dos serviços existentes na Internet. A ausência de uma norma especificando um modelo único de identidade, levou a que as organizações adoptassem diferentes soluções na forma como a identidade é definida, trocada e manipulada. Contudo, a adopção de soluções distintas levou ao aparecimento de muitos dos actuais problemas de autenticação existentes na Internet.

Apesar de terem aparecido propostas de sistemas globais para a identidade, nomeadamente o *Passport* [19], nenhuma teve aceitação alargada. Esta ausência de um modelo único e as diferentes visões levou à discussão do tema [20] pelas principais entidades interessadas (Microsoft, IBM, etc.). A discussão focava essencialmente na compreensão dos motivos de sucesso e insucesso das soluções actuais e passadas. As principais conclusões desta discussão foram publicadas num documento intitulado "As leis da identidade" [21]. O documento teve por objectivo elencar quais as propriedades que um sistema de gestão de identidade deve possuir, de forma a maximizar a sua aceitação. Este conjunto de propriedades definiu o modelo designado por *Identity Metasystem* [22].

Uma das leis, "Pluralismo de operadores e tecnologias", implica a coexistência e interacção dos vários modelos actualmente existentes. A concepção de um *Identity Metasystem* não tem como

[4] Baseado na figura *Federation and Trust Model* do documento [18].

objectivo substituir esses modelos, mas sim criar uma camada de integração, abstraindo as especificidades de cada um. Essa integração terá que passar pela definição de conceitos e operações universais sem entrar em detalhes de implementação.
Apresenta-se em seguida os conceitos e termos fundamentais deste modelo.

2.2.1 Identidade digital e *claims*

Uma identidade digital é uma representação digital de um conjunto de *claims* efectuadas por uma entidade, sobre si ou sobre outra entidade. Uma *claim* é uma afirmação sobre algo ou sobre alguém, e cuja veracidade está sujeita a verificação. Alguns exemplos de *claims* sobre pessoas são: "O João nasceu em 1970", "O João é aluno do ISEL", "O João trabalha na IBM", "O dono da chave pública FAE42… é o João". Alguns exemplos de *claims* sobre coisas são: "A chave pública do DEETC é FDA13…", "O DEETC é um departamento do ISEL".

2.2.2 *Identity Providers*, *Relying Parties* e *Subjects*

No *Identity Metasystem* distinguem-se três papéis: *Identity Provider*, *Relying Party* (RP) e *Subject*.
Um IP é uma entidade que emite identidades digitais, na forma de *security tokens* contendo *claims*. O conceito de IP é reflectido no mundo real, por exemplo: um governo é um IP para os seus cidadãos, e como as entidades interessadas reconhecem a autoridade do governo perante os seus cidadãos, então confiam nas *claims* (ex: um passaporte e o seu conteúdo) emitidas pelo governo, sobre os seus cidadãos. Contudo, a aceitação de *claims* não depende só da confiança, depende também do contexto, por exemplo: não é possível acedermos à biblioteca do ISEL só mostrando o passaporte, embora ninguém duvide da sua veracidade.
Um RP é uma entidade que providência um serviço de acesso controlado. Para que o RP autorize o acesso a esse serviço, o requisitante terá que apresentar um comprovativo da sua identidade. Em todos os modelos de autenticação encontramos entidades a representarem o papel de RP, como um serviço de intranet requisitando um bilhete Kerberos ou um serviço de Internet requisitando um certificado X.509.
O papel de *Subject* pode ser atribuído a todas as entidades envolvidas.

2.2.3 Claims Transformers

Um dos papéis essenciais do *Identity Metasystem* é o de *Claims Transformer*. Este elemento faz a ponte entre as fronteiras organizacionais, reduzindo as diferenças técnicas entre as várias representações da identidade e aumentando o desacoplamento entre os sistemas.
As transformações a aplicar dividem-se em três tipos. O primeiro tipo visa a transformação do formato do *token*, transformando por exemplo um bilhete *Kerberos* numa asserção SAML. O segundo tipo visa a inferência baseada em políticas (permissões, minimização de informação, etc), por exemplo: as *claims* "é aluno" e "nascido em Dezembro de 1970" poderão ser transformados nas *claims* "pode levantar o livro" e "maior de 18" respectivamente. No terceiro, a transformação visa a tradução de *claims*, traduzindo por exemplo "professor" em "docente".

Esta capacidade de transformação permite a interoperabilidade dos sistemas actualmente existentes e a introdução de novas tecnologias.

2.2.4 *Identity Metasystem* e as normas *WS-**

Para a implementação do *Identity Metasystem* foram identificados pelos seus autores algumas questões essenciais: como representar as identidades através de *claims*; como é que o *Identity Provider*, *Relying Party* e o *Subject* negoceiam entre si a transferência das *claims*; como obter as *claims*, políticas e requisitos; como fazer a ponte entre fronteiras organizacionais usando a transformação de *claims*; como tornar o uso de múltiplos contextos, tecnologias e operadores num uso consistente para o utilizador.

As normas *WS-** anteriormente referidas suportam estes requisitos do *Identity Metasystem*. As especificações disponibilizam os meios necessários para definir as acções e relacionar os vários intervenientes do *Identity Metasystem*. Como exemplo temos o *Security Token Service* que é usado como *Claims Transformer* e as normas *WS-SecurityPolicy* e *WS-MetadataExchange* [23] que são usadas pelos RPs para publicarem os seus requisitos e políticas.

2.3 *Information Card Model*

O modelo *Information Card* (ISIP) [24][25][26] segue e complementa os modelos anteriormente descritos. Neste modelo, os utilizadores gerem um conjunto de identidades digitais, emitidas por diferentes autoridades, e aplicam-nas a diferentes contextos para o acesso a serviços ou aplicações *Web*.

Alguns utilizadores poderão desempenhar o papel do próprio IP para a emissão de *claims* sobre si. Estas identidades digitais auto-emitidas são aplicáveis em muitas aplicações *Web* actualmente existentes. Por exemplo, quando um utilizador quer aceder a um serviço Web para a aquisição de alguma mercadoria, tem que criar uma conta e introduzir dados pessoais. Com a apresentação de uma identidade digital auto-emitida, o utilizador, além de não necessitar memorizar os dados de acesso, indica quais os dados a incluir e provará ser sempre o mesmo em futuras visitas ao serviço.

Para auxiliar o utilizador a seleccionar uma identidade digital num determinado contexto, o modelo introduz o conceito de *Identity Selector*. Um *Identity Selector* é uma aplicação que tem a capacidade de determinar quais as identidades digitais com as características exigidas pelo RP. Disponibiliza também uma *user interface* para que o utilizador possa visualizar, examinar e editar as suas identidades digitais.

Também para ajudar o utilizador a organizar as suas várias identidades digitais, o modelo *Information Card* introduziu o conceito de *information card*. Um *information card* é uma representação de uma identidade digital, que poderá ser visualizada, examinada e editada dentro de uma *user interface*. O modelo definiu dois tipos de *information card*: *self-issued* e *managed*. O *self-issued information card* é a representação da identidade digital auto-emitida, atrás apresentada, e é criado e editado pelo próprio utilizador, através do *Identity Selector*. O

managed information card é gerado por uma terceira entidade, pertencente ou de confiança do RP, e contém apenas o URL do IP/STS que emite o *security token* requerido pelo RP. É importante realçar que um *information card* não é um *security token*, mas sim um meio para a obtenção do *security token*, e que múltiplas identidades digitais emitidas pelo mesmo IP são representadas por *information cards* diferentes.

Quando uma aplicação cliente necessita de um determinado *security token* para satisfazer os requisitos apresentados por um serviço, invoca o *Identity Selector* para que o utilizador possa escolher qual dos *information cards* a usar. Após a escolha do *information card*, e através da norma *WS-Trust*, o *Identity Selector* obtém o *security token* do IP/STS referenciado no *information card*.

O modelo *Information Card* estende a norma *WS-Trust*, adicionando algumas restrições e extensões às mensagens RST e RSTR.

2.4 *OpenID*

A especificação *OpenID* [27] define uma *framework* descentralizada para a manipulação de identidades digitais. Nesta especificação a identidade digital é representada pelo URL de *blogs*, páginas pessoais, etc. e tira partido de tecnologias *Web* já existentes (HTTP/HTTPS).

A especificação descreve o processo que garante a autenticidade de um identificador pertencente a um utilizador, sem haver a necessidade do envio de credenciais para o RP. Sendo uma *framework* descentralizada, não existe nenhuma autoridade central para a aprovação ou registo de RPs ou IPs.

O *OpenID* sofre de vulnerabilidades de segurança, nomeadamente ataques do tipo *phishing* [28]. Por esse motivo foi definida uma variante à especificação principal, designada por *OpenID Information Cards* [29]. Esta especificação baseia-se nos modelos anteriormente descritos e acrescenta o *OpenID token*, contendo como *claim* o identificador *OpenID*. Também neste contexto é usado o conceito de *Security Token Service*, como elemento emissor de *security tokens*. Neste caso, os *security token* contêm informação proveniente das mensagens do protocolo *OpenID*.

2.5 Trabalho relacionado

Esta secção apresenta soluções alternativas ou relacionadas com o conceito de *Security Token Services*.

2.5.1 ADFS

Os serviços *Active Directory Federation Services* (ADFS), disponíveis a partir do Windows Server 2003 R2, permitem que organizações partilhem entre si a identidade digital e os direitos de acesso dos seus utilizadores, providenciando autenticação e autorização distribuída. Estes serviços utilizam o perfil passivo descrito na norma *WS-Federation*.

Havendo a necessidade simultânea de expor os serviços a clientes externos e de obter dados internos, sensíveis e necessariamente protegidos, os serviços ADFS foram desenhados numa

arquitectura 2-Tier. O *Federation Service Proxy* (FS-P), localizado numa zona DMZ exposta ao exterior, é uma aplicação *web front-end* que serve de intermediário entre clientes passivos e o *Federation Service* (FS). Na segunda *tier* está localizado o FS, que é um STS usado para a emissão de *security tokens* contendo informação fornecida pelo *Active Directory* (AD). Os FS-Ps podem manter relações de confiança com um ou mais FSs. A inclusão de outros intervenientes, como o AD ou o *Authorization Manager* (*AzMan*) [30], resulta numa terceira *tier*.

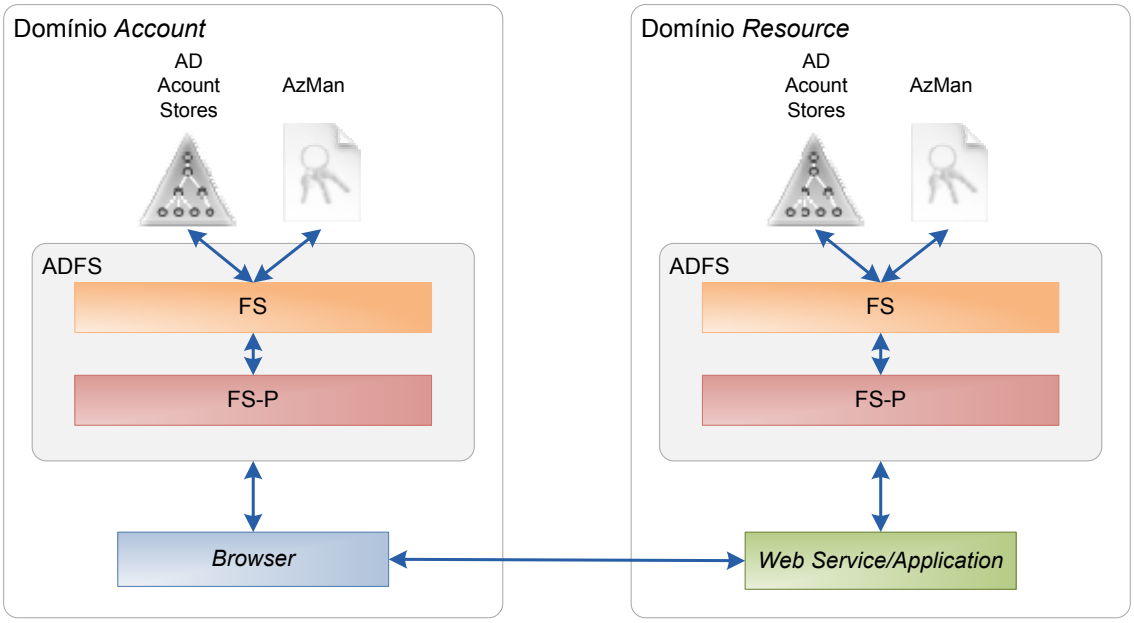

Figura 2.3 – Arquitectura ADFS

O FS e o FS-P desempenham papéis diferentes em função do domínio onde se encontram (Figura 2.3). Numa transacção típica, o FS do domínio aonde pertence o cliente, designado de *Account* FS, autentica o utilizador perante o AD, obtém os atributos do utilizador do AD, gera uma colecção de *claims* necessárias no pedido de acesso e emite um *security token* que inclui estas *claims*. O FS do domínio do recurso, designado de *Resource* FS, valida a autenticidade do *token*, processa-o e fornece as *claims* contidas a uma aplicação apropriada para a toma de decisões de autorização. O *Account* FS-P providência uma *user interface* ao cliente para o pedido das credenciais de autenticação e reencaminha-as ao FS associado. O *Resource* FS-P aceita *tokens* de contas federadas, emitidas pelo *Account* FS, e reencaminha-os para o FS associado.

Os *security tokens* obedecem à sintaxe definida na norma SAML 1.1, e como o meio de transporte usado é o protocolo HTTPS, não há a necessidade de cifra das mensagens. Por omissão os *tokens* são assinados com uma chave privada RSA [31] e verificados pela chave pública disponível através de certificados X.509. Opcionalmente, os *tokens* emitidos pelo *Resource* FS são assinadas com a chave de sessão do *Kerberos*.

É necessário configurar as políticas de confiança (*Trust Policy*) para que reflictam as relações de confiança com as organizações associadas. Estas configurações consistem na identificação das organizações, associação de *claims* ou grupos de *claims* e mapeamentos. Os mapeamentos dependem da origem, do destino, do sujeito e do recurso.

2.5.2 WCF SDK *samples*

A Microsoft disponibilizou uma implementação [32] para exemplificar a aplicação das normas *WS-Federation* e *WS-Trust* sobre a plataforma *Windows Communication Foundation* (WCF)[5].
Nesta implementação existe um conjunto de classes e interfaces que servem de base aos vários serviços STS implementados. A arquitectura implementada visou essencialmente a aplicação considerada no exemplo, tornando-se limitativa e pouco fléxivel, e possibilitando apenas um pequeno número de aplicações.
Embora esta implementação seja limitada e pouco fléxivel, foi com base no estudo das suas caracteristicas e limitações que se deu início ao desenho e implementação da *STSLib*.

2.5.3 Projecto Higgins

O projecto Higgins [33] é uma *framework* que surge da aliança entre a IBM, Novell e Parity Communications, para criar uma alternativa *open source* às soluções apresentadas pela Microsoft. O projecto é implementado sobre a plataforma Java e é mantido na comunidade da plataforma Eclipse. É uma infra-estrutura que suporta a maioria dos protocolos usados para a identidade digital, incluindo *WS-Trust*, *OpenID*, SAML e LDAP (*Lightweight Directory Access Protocol*).
O projecto Higgins disponibiliza um conjunto de componentes base para a criação de *Identity Providers*, *Security Token Services* e *Relying Parties*, e a sua arquitectura é construída para permitir a máxima extensibilidade e versatilidade. Através de *plugins*, configuráveis e dependentes dos contextos, poderá adaptar soluções já existentes ou implementar novas soluções.
O STS do projecto Higgins implementa a norma *WS-Trust*. É constituído por um conjunto base de classes e interfaces, vários *plugins* e uma aplicação *web* que simplifica o registo e a gestão das identidades digitais.
Para a ligação do STS a repositórios de identidade, sistemas de controlo de acessos ou outros, o projecto Higgins disponibiliza um *Identity Attribute Service* e vários *Context Providers*.
Para o STS foram disponibilizados dois conjuntos de *plugins*. O primeiro conjunto, designado por *bindings*, são responsáveis pelo carregamento e configuração do STS, e por um processamento prévio e/ou posterior à passagem da mensagem RST ao segundo conjunto de *plugins*. O segundo conjunto é responsável pelo processamento dos *security tokens* e das suas *claims*.
Na maioria dos casos e através da sua configuração, o STS sabe que extensões processam que mensagens, mas em determinados casos o STS terá que fazer *polling* para determinar qual usar.
Todos os componentes atrás indicados são fornecidos como parte de uma implementação de referência, e qualquer um deles poderá ser substituído por uma implementação alternativa, desde que implementem as interfaces apropriadas e as funcionalidades operacionais.

---

[5] As características principais da plataforma WCF são apresentadas no Apêndice A.

3 Objectivos do projecto

No capítulo anterior apresentou-se a importância do conceito de STS, em diferentes contextos e aplicações. No *Identity Metasystem*, todo o mecanismo de interoperabilidade e compatibilidade dos actuais modelos de segurança e entre as organizações, passa por um *Claims Transformer*, cujo papel é desempenhado pelo STS. Das normas *WS-**, a norma *WS-Trust* é a base de todo o modelo de autenticação para os *web services*, através do seu componente principal, o STS.

Dos vários estudos teóricos e exemplos estudados e apresentados no capítulo anterior, conclui-se que o STS desempenha diferentes papéis, dependentes do contexto, da arquitectura e do seu objectivo. O papel primordial será o de *Identity Provider*, onde o IP asserta *claims* de identidade, necessárias para a autenticação do sujeito perante o RP. O IP/STS está associado a repositórios de identidade e poderá estar localizado no domínio do cliente, do recurso ou ainda numa terceira entidade.

Outro papel que um STS desempenha é o de *Policy Decision Point*, onde, associado a um sistema de controlo de acessos como o RBAC (*Role Base Access Control*) [34], toma decisões de autorização nos pedidos de acessos aos recursos associados. Nestes casos o STS emite um *security token* com uma única *claim* assertando se o sujeito está autorizado ou não.

Existem topologias onde não há uma relação de confiança directa entre os dois STS dos domínios do cliente e do recurso. Nestas topologias a relação de confiança poderá passar por um terceiro STS localizado numa terceira entidade. Aqui o papel desempenhado é o de elo de ligação numa cadeia de confiança, onde participam 3 ou mais STS.

Embora já tenham aparecido varias soluções, como as apresentadas no capítulo anterior, que possibilitam aplicar as normas e os conceitos anteriormente apresentados, a maioria apenas cobre uma pequena área das aplicações possíveis. O projecto Higgins visa os mesmos objectivos que os nossos, mas aplicável à plataforma Java. Para a plataforma .Net ainda não existe[6] nenhuma implementação que visasse cobrir todas as aplicações anteriormente apresentadas. Este projecto visa especificamente colmatar esta lacuna através da implementação de uma biblioteca software suportada sobre a plataforma .Net 3.5, para a construção de *Security Token Services*.

A arquitectura da *STSLib* tem que ser suficientemente flexível de forma que as várias concretizações do STS desempenhem os vários papéis apresentados, em diferentes contextos e aplicações. A arquitectura também tem que ser extensível de forma a incluir novas implementações e suportar novas versões das normas aplicadas.

A *STSLib* (Figura 3.1) divide-se em duas partes: *framework* base e extensões. A *framework* base define um núcleo constituído por um conjunto de classes e interfaces que servirão de

[6] Á data de escrita deste relatório foi apresentado pela Microsoft a versão beta do Zermatt. O Zermatt que é uma *framework* para a realização de *Security Token Services* sobre a plataforma .Net.

base às várias soluções a implementar, em conjunto com um número mínimo de módulos de extensão, para que seja possível a configuração de um STS com um conjunto de mínimo de funcionalidades. A opção de implementar o conjunto de mínimo de funcionalidades através de módulos de extensão visou a extensibilidade para novas versões das normas aplicadas.

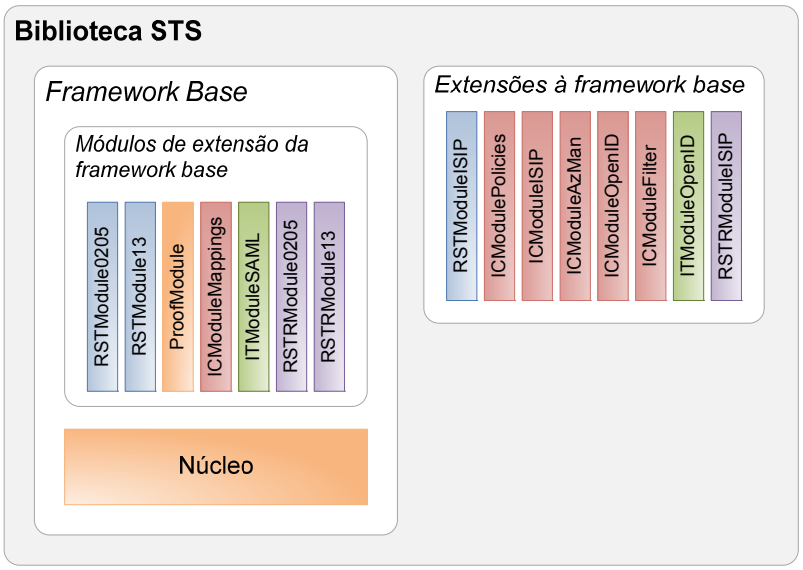

Figura 3.1 – Biblioteca STS

As extensões à *framework* base visam o suporte da *STSLib* para as extensões à norma *WS-Trust* e a integração de outros modelos ou sistemas externas, como por exemplo ISIP, *OpenID* e o *Authorization Manager*.
Em seguida, detalham-se os requisitos dos diferentes componentes.

3.1.1 Requisitos da Framework base

- Suporte para o *issue binding* de diversas versões da norma *WS-Trust*.
- Suporte para extensões às mensagens RST e RSTR (ex. ISIP).
- Suporte para diversos métodos de autenticação do requerente.
- Suporte para diversas políticas de emissão de *tokens*.
- Políticas de emissão mantidas em ficheiros de configuração ou em base de dados.

3.1.2 Extensões para diversos cenários de aplicação

- Suporte para as extensões definidas no ISIP.
- Suporte para a utilização do *Authorization Manager*.
- Suporte para as especificações *OpenID*.

3.1.3 Outros requisitos exteriores à biblioteca

- Aplicação *web* para a gestão das políticas de emissão do STS.
- Geradores de *Information Cards* ISIP e *OpenID*.

4 Arquitectura

O presente capítulo apresenta a arquitectura da *STSLib*, nomeadamente: a arquitectura base, a sua extensibilidade e as justificações para as opções tomadas.

4.1 Arquitectura global

Na Figura 4.1 apresenta-se a arquitectura de alto nível adoptada e implementada para a concretização do STS em diferentes contextos e aplicações. A arquitectura, restrita apenas à *STSLib*, é constituída por um núcleo e um conjunto de módulos de extensão. O núcleo do *STSLib* assenta sobre a plataforma WCF 3.5[7] e é constituído por um *pipeline* – cujas origens, características e funcionalidades são apresentadas na secção seguinte – e por uma arquitectura de integração de módulos de extensão para a flexibilização e a extensão da *STSLib* para futuras normas e diferentes contextos e aplicações. Os módulos de extensão predefinidos e incluídos na *STSLib* implementam os aspectos necessários para a concretização do STS nos vários papéis mencionados neste documento. Incluiu-se na arquitectura uma aplicação *web* (*STSMan*) para a configuração numa base de dados as políticas de emissão do STS.

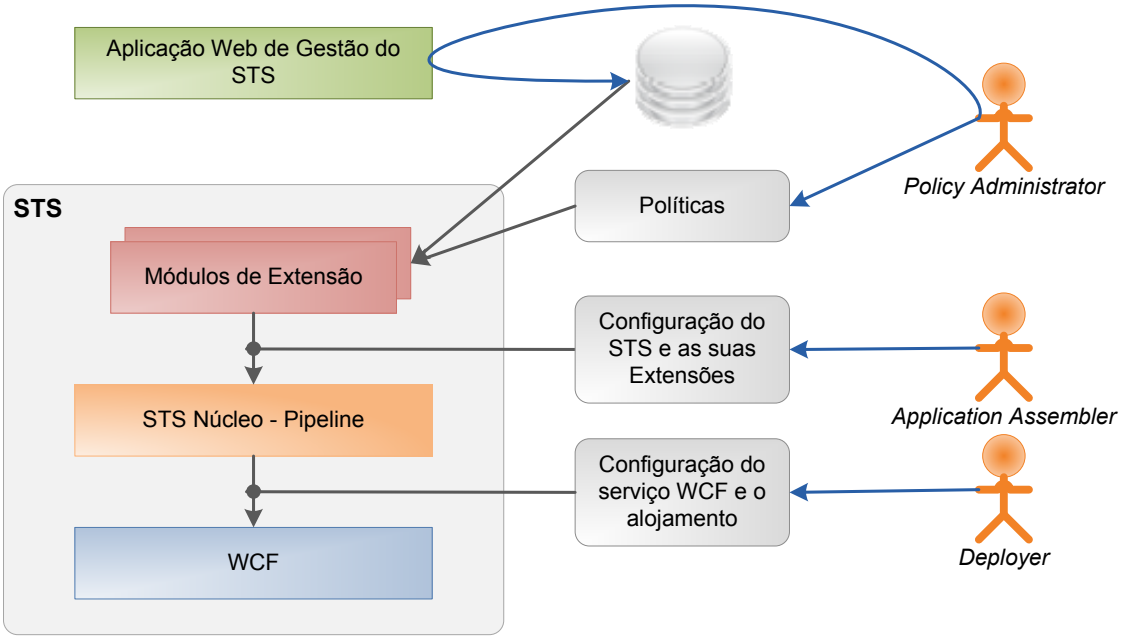

Figura 4.1 – Arquitectura global do STS

Na instanciação de um STS, para um determinado papel, existem diferentes configurações a realizar nas diferentes fases da sua montagem, disponibilização e operação. Os objectivos das diferentes configurações são claramente distintas, por isso definiu-se três intervenientes

[7] A versão é especifica da *framework* .Net e não da plataforma WCF.

distintos para cada conjunto de configurações. Na Figura 4.1 e na Figura 4.2 apresentam-se esses intervenientes e as fases necessários para a configuração e instanciação de um STS. Na primeira figura realça-se os pontos de configuração na arquitectura de uma instância do STS e os intervenientes que os configuram. Na segunda figura é apresentada a sequência de configurações necessárias para a instanciação de um STS.

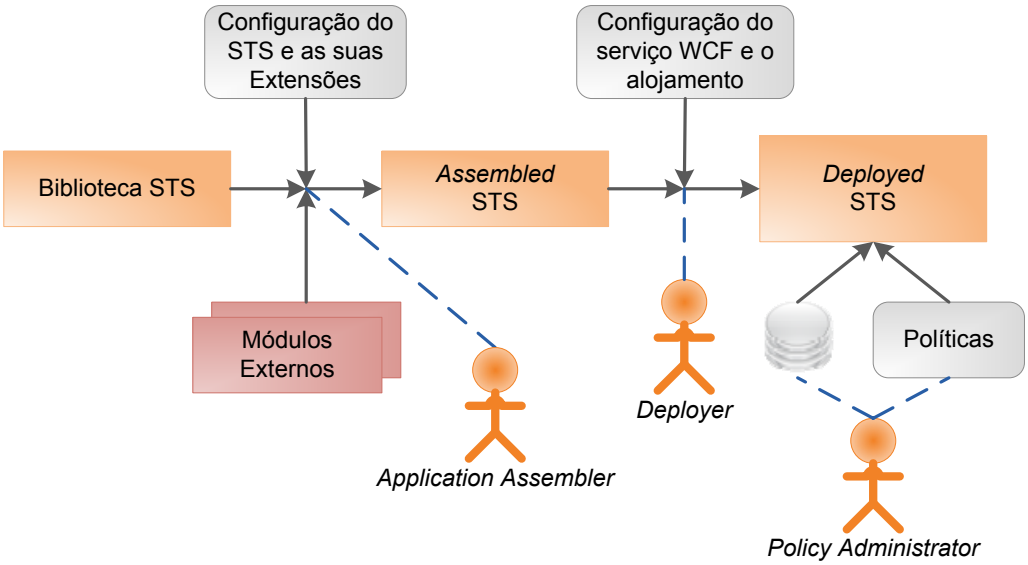

Figura 4.2 – Fases de configuração e inicialização de um STS

Os vários intervenientes e as suas funções[8] são as seguintes:

- ***Application Assembler:*** É o primeiro a intervir na configuração do STS e tem como objectivos a adição de módulos de extensão, da *STSLib* ou externos, e a configuração necessária para a integração funcional desses módulos no núcleo do STS, para o contexto e a aplicação pretendida. Como exemplo, e para uma concretização de um STS que suporte as extensões ISIP, o *Application Assembler* terá que incluir, além dos módulos mínimos que implementam a norma *WS-Trust*, os módulos que implementam as extensões ISIP.
- ***Deployer:*** Após a configuração de uma montagem do STS, pelo *Application Assembler*, para uma determinada aplicação, o *Deployer* disponibiliza o STS, aplicando uma configuração específica para a plataforma WCF e outros parâmetros necessários e dependentes do alojamento do serviço STS, como por exemplo o alojamento do serviço no IIS.
- ***Policy Administrator:*** A função do *Policy Administrator* é definir as políticas de emissão do STS. As políticas de emissão podem ser as definições dos *Relying Parties*, com o qual o STS tem uma relação de confiança, e os mapeamentos das *claims* a emitir. A necessidade de configurar políticas de emissão depende do papel do STS e

[8] Baseado nos *roles* J2EE [35]

podem ser configuradas em ficheiros de configuração ou numa base de dados. Estas configurações podem ser definidas antes ou depois da instanciação do STS.

4.2 Arquitectura do Núcleo - *Pipeline*

Para a concretização dos objectivos do projecto, o núcleo da arquitectura apresentada, além de ter que ser suficientemente flexível e extensível para permitir diversas aplicações e os vários papeis que um STS poderá desempenhar, terá que disponibilizar um serviço *web* com uma operação de emissão de *security tokens*. A operação implementada terá que suportar a acção de *issue binding*, tal como especificado na norma *WS-Trust*.

Para a concretização da operação de emissão considerou-se a implementação de um *pipeline* de processamento, tendo como fonte inspiradora as arquitecturas dos processadores HTTP, como o IIS e o ASP.NET. Nestas arquitecturas, os pedidos recebidos são reencaminhados através de um conjunto de módulos HTTP. Cada um dos módulos poderá inspeccionar o pedido, e dependente das decisões tomadas, alterar o fluxo interno desse pedido.

No núcleo implementado, o *pipeline* de emissão recebe à entrada uma mensagem *Request Security Token* (RST), tal como especificado na norma *WS-Trust*, e devolve ao requerente uma mensagem *Request Security Token Response* (RSTR), contendo o *security token* pedido.

Para a concretização do *pipeline* de emissão identificaram-se e isolaram-se os procedimentos da operação de emissão. Estes procedimentos dividem-se em três conjuntos. O primeiro conjunto é constituído por procedimentos baseados na norma *WS-Trust*, como por exemplo a interpretação da mensagem RST, a construção da mensagem RSTR e a geração do *token* de prova de posse (*proof token*). O segundo conjunto é constituído por procedimentos especificados em extensões à norma *WS-Trust*, como por exemplo as especificações ISIP, e outras especificações necessárias para a concretização da operação de emissão e não contempladas na norma *WS-Trust*, como por exemplo a especificação SAML, que define a estrutura de um dos tipos possíveis do *security token* a emitir. O terceiro e último conjunto é constituído por procedimentos não especificados em nenhuma norma, mas necessários para a concretização da operação de emissão, como por exemplo a inferência e a origem das *claims* a inserir no *security token*.

Após a identificação de todos os procedimentos necessários para as diferentes concretizações, procurou-se identificar qual a ordem de processamento ideal e quais os procedimentos que irão participar activamente dentro do *pipeline* de emissão, para cada papel do STS. A escolha dos procedimentos a incluir no processamento da emissão do *security token* terá que ser flexível e automático, dependente mais da informação contida no pedido do que da configuração do *Application Assembler*. A concretização deste objectivo leva à possível inclusão de todos os procedimentos possíveis no *pipeline*, por parte do *Application Assembler*, e à participação activa apenas dos procedimentos necessários com base na informação contida no pedido.

O processamento do *pipeline* foi dividido em cinco fases, e em cada fase agrupou-se os procedimentos do mesmo tipo. As fases e os seus procedimentos são seguintes:

- **A interpretação da mensagem RST**. O conjunto de procedimentos desta fase tem o objectivo de interpretar a informação na mensagem RST, em formato XML, e reencaminha-la para as fases seguintes do *pipeline* como uma instância dum modelo de classes com propriedades tipificadas que representam os elementos constituintes da mensagem. Este conjunto de procedimentos realiza-se dentro de um *sub-pipeline*. Este novo *pipeline* processa a mensagem RST, entregando-a aos procedimentos configurados e especializados para esta fase.
- **A geração do *token* de prova de posse (*proof token*)**. Esta fase tem como objectivo a geração do *token* de prova de posse, tal como especificado na norma *WS-Trust*. Esta fase recebe da fase anterior o objecto RST representante da mensagem RST, que contém a informação necessária para a geração do *token*. Este *token*, em conjunto com o objecto RST, é passado às fases seguintes do *pipeline* de emissão. Esta fase terá que incluir sempre e apenas um único procedimento, o especificado na norma *WS-Trust*.
- **O processamento das *claims* a incluir no *security token* a emitir**. Este conjunto de procedimentos tem o objectivo de gerar as *claims* pedidas e especificadas no objecto RST, e que serão posteriormente inseridas no *security token* a emitir. O STS infere, a partir das *claims* oriundas da autenticação do requerente, da política de emissão e das *claims* requeridas, quais as *claims* a gerar. Estes procedimentos não estão especificados em nenhuma norma e estão muito dependentes dos papéis desempenhados pelo STS. Á semelhança dos procedimentos da primeira fase, estes procedimentos também são realizados num *sub-pipeline*. Este novo *pipeline* processa as *claims* com base nas entradas anteriormente referidas, entregando-as aos vários procedimentos configurados, e obtendo como resultado final o conjunto de *claims* requeridas.
- **A geração do *security token* a emitir**. Este conjunto de procedimentos tem o objectivo de gerar o *security token* com o formato especificado no objecto RST. Poderá coexistir vários procedimentos especializados em diferentes tipos de formatos. A selecção de qual o procedimento a usar é feita automaticamente com base na informação contida no objecto RST. Para a geração do *security token*, estes procedimentos recebem das fases anteriores do *pipeline* de emissão, o *proof token* e o conjunto de *claims* a emitir. Após a geração, o *security token* é inserido num objecto RSTR, que é uma instância dum modelo de classes com propriedades tipificadas que representam os elementos constituintes de uma mensagem RSTR, e que é depois enviado para a última fase do *pipeline*.
- **A geração da mensagem RSTR**. Nesta fase, os procedimentos têm por objectivo transformar o objecto RSTR numa mensagem RSTR, em formato XML. Este conjunto de procedimentos, à semelhança do conjunto de procedimentos da primeira e terceira

fase, realiza-se dentro de um *sub-pipeline*. Este *pipeline* processa o objecto RSTR, entregando-o aos procedimentos configurados e especializados nas especificações desta fase.

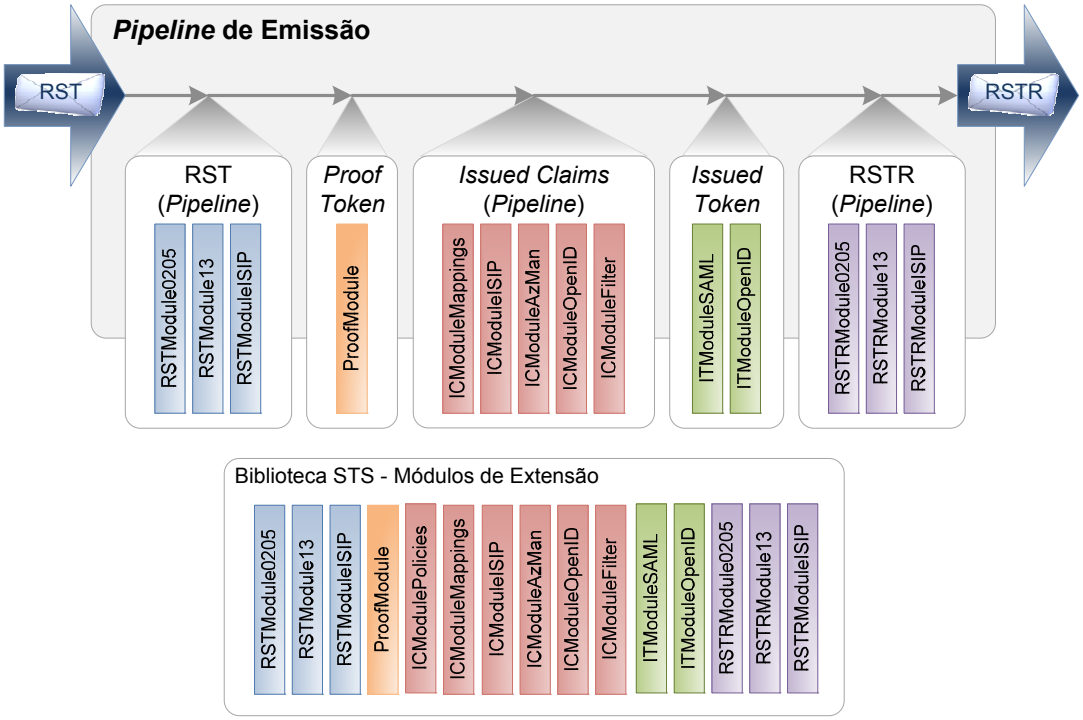

Figura 4.3 – *Pipeline* de emissão e os módulos de extensão

Para a concretização das cinco fases do *pipeline* foram definidos cinco pontos de extensão no processamento do núcleo e cada um dos procedimentos especializados são concretizadas por módulos de extensão. Na Figura 4.3 apresenta-se a arquitectura do *pipeline* atrás referenciado e o conjunto de módulos de extensão implementado e incluído na *STSLib*.

A inclusão e a participação activa dos módulos de extensão no *pipeline* são efectuadas por dois processos distintos. O *Application Assembler* poderá incluir no *pipeline* todas os módulos implementadas para todos os ponto de extensão excepto o terceiro. A escolha de quais os módulos que participam activamente em cada pedido é efectuado automaticamente através da informação contida no pedido RST. Os módulos a incluir no terceiro ponto de extensão estão directamente dependentes dos papeis a desempenhar pelo STS e com quais sistemas exteriores o STS terá que interagir para a concretização dos objectivos inerentes aos papeis. Como exemplo temos a obtenção de informação de identidade de um directório ou de uma autorização de um *Policy Decision Point* (PDP). Alguns dos módulos implementados para este ponto de extensão têm a capacidade de verificar dinamicamente a necessidade da sua participação no processo, tal como os módulos implementados para os outros pontos de extensão, possibilitando assim uma configuração única para os diferentes papéis a desempenhar pelo STS. Contudo, a não possibilidade da coexistência de alguns módulos neste ponto de extensão, dependendo da sua implementação, leva a que o *Application*

Assembler desempenhe um papel necessariamente mais activo na sua configuração. Disponibilizou-se processos de configuração distintos, dependentes dos meios pretendidos e dos módulos incluídos, para que a definição das políticas de emissão seja flexível e configurável pelos serviços (*Policy Administrator*) que pretendam usar o STS.

5 Implementação base

O presente capítulo tem por objectivo apresentar aspectos da implementação da arquitectura apresentada no capítulo anterior. Também se detalha a arquitectura implementada para a inclusão de extensões, usada na flexibilização e extensão da *STSLib* para futuras normas e diferentes contextos e aplicações.

5.1 Contrato de serviço WCF para o STS

A implementação de um serviço WCF que reflicta a operação de emissão de *security tokens*, especificada na norma *WS-Trust*, passa pela definição de um contrato de serviço WCF que caracterize essa mesma operação. O contrato contém uma operação WCF que recebe uma mensagem de pedido RST, e caso suceda, devolve uma mensagem de resposta RSTR.
As propriedades `Action` e `ReplyAction`, associadas à operação WCF, reflectem o conteúdo que o elemento `Action` da mensagem SOAP terá que conter no pedido e na sua resposta. A norma *WS-Trust* especifica um conjunto de acções *WS-Addressing* [36] (URI) para o preenchimento destes elementos. Contudo, esse conjunto difere para cada versão da norma *WS-Trust*:

- *WS-Trust* de Fevereiro de 2005:
  Mensagem RST, `Action =`
  `"http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Issue"`
  Mensagem RSTR, `Action =`
  `"http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Issue"`
- *WS-Trust* de Março de 2007 (Versão 1.3):
  Mensagem RST, `Action =`
  `"http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/Issue"`
  Mensagem RSTR, `Action =`
  `"http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Issue"`
  Mensagem RSTRC, `Action =`
  `"http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTRC/IssueFinal"`

Neste projecto considerou-se as duas versões actualmente existentes e atrás apresentadas.
Como o conteúdo das propriedades atrás referenciadas terá que ser diferente para cada versão da norma, e para que o serviço WCF possa responder a pedidos de qualquer uma das versões escolhidas, o serviço terá que disponibilizar contratos WCF específicos para cada versão (Figura 5.1). Assim sendo, e sempre que seja necessário o suporte de uma futura versão da norma, terá que se implementar uma nova interface com as propriedades correspondentes à nova norma. As restantes implicações da inclusão de uma nova versão da norma serão realçadas nas várias secções deste capítulo.

```
[ServiceContract]
public interface ISecurityTokenService0502
{
  [OperationContract(
    Action = Constants.WSTrust.Actions0502.Issue,
    ReplyAction = Constants.WSTrust.Actions0502.IssueReply)]
  Message ProcessRequestSecurityToken(Message message);
}

[ServiceContract]
public interface ISecurityTokenService13
{
  [OperationContract(
    Action = Constants.WSTrust.Actions13.Issue,
    ReplyAction = Constants.WSTrust.Actions13.IssueFinalReply)]
  Message ProcessRequestSecurityToken(Message message);
}
```

Figura 5.1 – Contrato WCF

5.2 Implementação do contrato de serviço WCF para o STS

Para a implementação do serviço WCF criou-se uma classe abstracta (Figura 5.2) que implementa os vários contratos anteriormente definidos. Optou-se por definir uma única classe que implementa os vários contratos em vez de uma classe por contrato, assim minimizando a replicação de código com a inclusão de novas versões. Para a distinção entre pedidos de diferentes versões é analisada o elemento `Action` da mensagem SOAP recebida, e dependendo do seu conteúdo, são afectadas duas propriedades da classe. Uma propriedade é afectada com o URI da versão da norma, a outra com o URI que será necessário inserir no elemento `Action` da mensagem SOAP a devolver. A primeira propriedade é usada nas várias fases da emissão para distinguir quais os módulos de extensão a usar, dependentes da versão da norma.

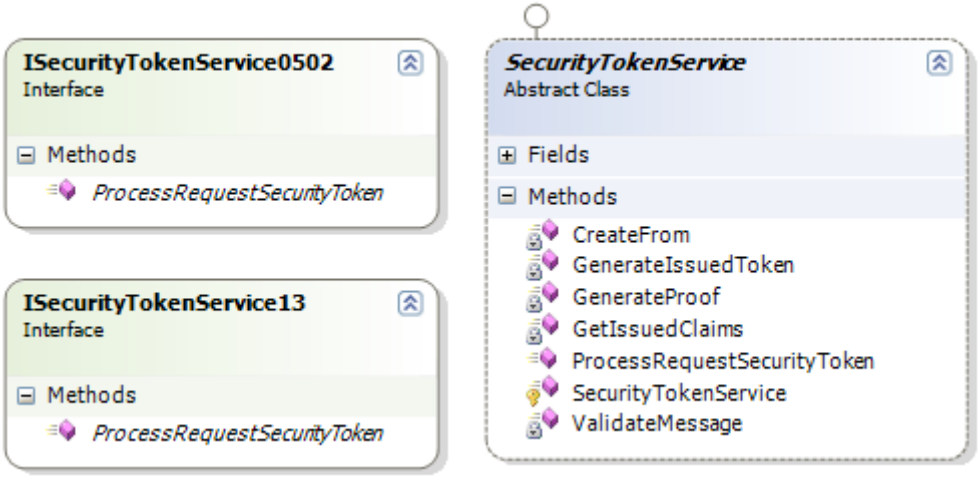

Figura 5.2 – Implementação do serviço WCF

O processo para a emissão de *security tokens* reflecte a arquitectura descrita no capítulo anterior, onde se apresentou o *pipeline* de emissão com as seguintes fases:

- Recepção e processamento do **RequestSecurityToken**.
- Criação de um *proof token* a partir de informação e especificações contidas no RST.

- Processamento das *claims* recebidas e preparação das *claims* para a emissão com base na política do STS.
- Criação do *security token* a partir das *claims* a emitir, do *proof token* e de especificações contidas no RST.
- Processamento e envio do **RequestSecurityTokenResponse**.

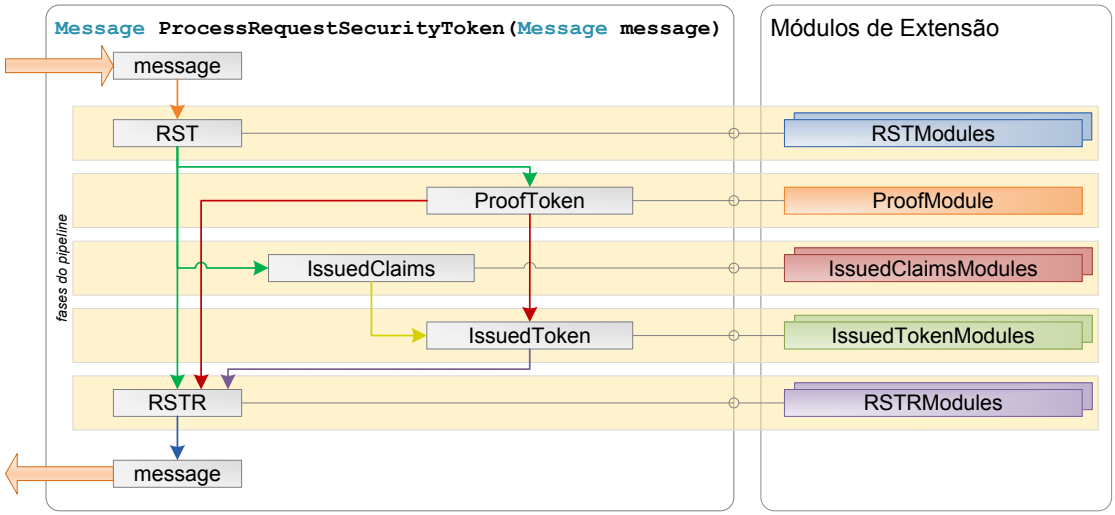

Figura 5.3 – Processo de emissão, as cinco fases do *pipeline* de emissão e os módulos de extensão

O processo do *pipeline* inicia-se com a recepção de uma mensagem SOAP contendo um elemento `RequestSecurityToken` no seu corpo. A primeira operação consiste na transformação do `RequestSecurityToken`, em formato XML, numa instância dum modelo de classes com propriedades tipificadas que representam os elementos constituintes da mensagem. Este processo de transformação por si só também possui o seu próprio *pipeline,* como se descreve na secção **Recepção de um RequestSecurityToken**. Após a transformação, o objecto RST é usado na criação de um objecto RSTR, e ambos são passados às fases seguintes do *pipeline*.

A fase seguinte consiste na criação de um *proof token*, tendo como base a informação contida no objecto RST recebido. Dependendo do *proof token* pedido, o *proof token* resultante é anexado ao objecto RSTR e também passado às fases seguintes do *pipeline*. Este processo é descrito pormenorizadamente na secção **Geração do Proof Token**.

A próxima fase visa o processamento das *claims* pedidas e especificadas no objecto RST. Este processo infere quais as *claims* a emitir com base nas *claims* recebidas e na política de emissão do STS. O conjunto de *claims* resultante é passado às fases seguintes do *pipeline*. Esta fase é descrita em detalhe mais adiante na secção **Processamento de Claims**.

A fase seguinte recebe o *proof token* e o conjunto de *claims* anteriormente processados no *pipeline*, e usa-os para a geração do *security token*. O *security token* gerado é depois anexado

ao RSTR para a posterior emissão na última fase no *pipeline*, como veremos mais adiante na secção **Geração do *Security Token***.

Na última fase do *pipeline*, o objecto RSTR, já contendo os dados necessários, é transformado num elemento `RequestSecurityTokenResponse` e é inserido numa mensagem SOAP para a devolução ao requente. Este processo é apresentado em detalhe mais adiante na secção **Envio do RequestSecurityTokenResponse**.

5.3 Arquitectura de extensões

Após a caracterização dos dados de entrada e de saída de cada ponto de extensão atrás apresentado, e com a necessidade de flexibilizar os procedimentos a incluir em cada ponto de extensão, através de módulos de extensão, e configuráveis pelo *Application Assembler*, implementou-se a arquitectura apresentada na Figura 5.5.

Para cada ponto de extensão apresentado no capítulo anterior implementou-se uma interface (Figura 5.4) caracterizando o módulo e os parâmetros necessários para a implementação de um módulo de extensão. Cada módulo terá que implementar uma das interfaces disponíveis para que possa ser incluída no processamento do *pipeline* de emissão.

```
public interface IRSTProcessor
{
  void ProcessMessage(RST rst, XElement xRst);
}

public interface IProofProcessor
{
  SecurityKeyIdentifier GenerateProof(RST rst, RSTR rstr);
}

public interface IIssuedClaimsProcessor
{
  void GetIssuedClaims(RST rst, Collection<Claim> claims);
}

public interface IIssuedTokenProcessor
{
  void GenerateIssuedToken(string stsName, SecurityKeyIdentifier proof,
    Collection<Claim> claims, RSTR rstr);
}

public interface IRSTRProcessor
{
  void ProcessRstr(RSTR rstr, ref XDocument xRstr);
}
```

Figura 5.4 – Interfaces para os módulos de extensão

A classe `AddInTool` é usada pelo STS para o carregamento dos módulos de extensão configurados pelo *Application Assembler*. Esta classe disponibiliza cinco propriedades estáticas, uma para cada tipo de extensão (fase do *pipeline*). A propriedade `ProofProcessorModule` contém uma instância `de IProofProcessor`, e as propriedades `RstProcessorModules`, `IssuedClaimsProcessorModules`, `IssuedTokenProcessorModules` e `RstrProcessorModules` são contentores de instâncias `de IRSTProcessor`, `IIssuedClaimsProcessor`,

`IIssuedTokenProcessor` e `IRSTRProcessor`, respectivamente. A classe `AddInTool` possui também o método estático `LoadAddIns`, que é chamado sempre que o serviço do STS é iniciado, e cujo objectivo é carregar as propriedades atrás referenciadas com instâncias dos módulos de extensão configurados pelo *Application Assembler*.

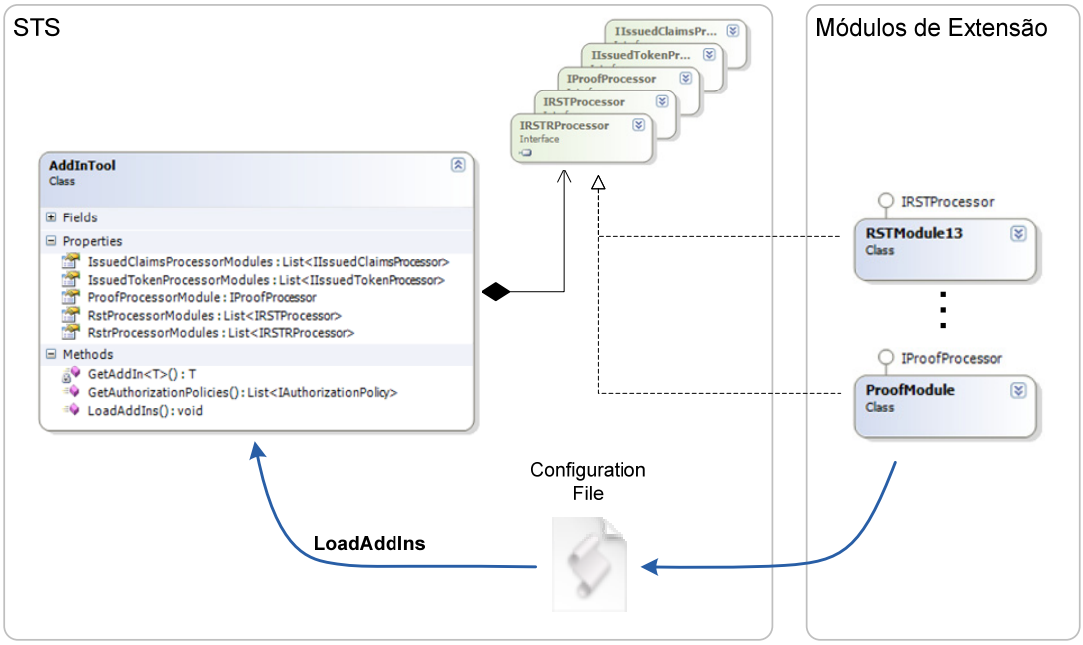

Figura 5.5 – Arquitectura de extensões

Para a inclusão dos módulos de extensões na configuração do STS, o *Application Assembler* adiciona ao ficheiro de configuração do serviço os nomes dos módulos de extensão (classes implementadas) e respectivas bibliotecas. Para isso houve a necessidade de definir um conjunto de classes (Figura 5.6) representativa da configuração do STS contida no ficheiro de configuração (Figura 5.7). Esta arquitectura especifica qual a estrutura, os elementos e os atributos permitidos no ficheiro de configuração. Outros aspectos da configuração são apresentados em pormenor na secção **Configuração do STS**.

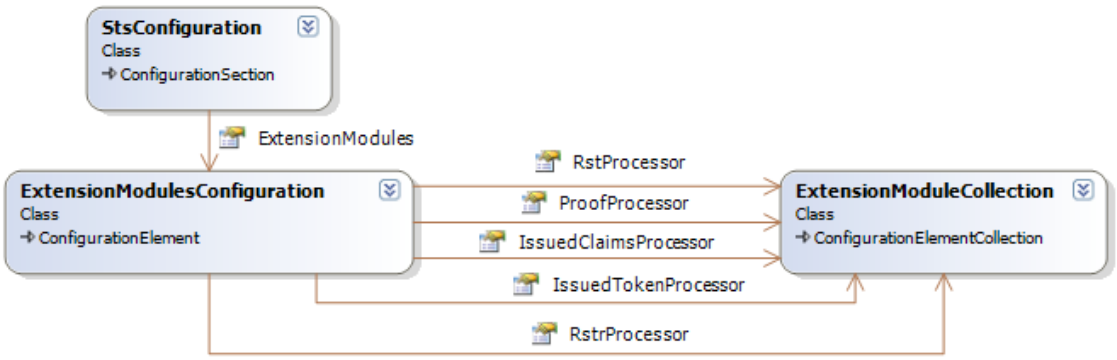

Figura 5.6 – Arquitectura de configuração de extensões

```
<extensionModules path="C:\...\">
  <rstProcessor>
    <add type="RSTModule0502" lib="RstModules.dll"/>
    <add type="RSTModule13" lib="RstModules.dll"/>
    ...
  </rstProcessor>
  <proofProcessor>...</proofProcessor>
  <issuedClaimsProcessor>...</issuedClaimsProcessor>
  <issuedTokenProcessor>...</issuedTokenProcessor>
  <rstrProcessor>...</rstrProcessor>
</extensionModules>
```

Figura 5.7 – Exemplo de uma configuração das extensões

5.4 O *Pipeline* e os seus módulos de extensão

5.4.1 Recepção de um *RequestSecurityToken*

A Figura 5.8 apresenta uma amostra de uma mensagem SOAP com um elemento `RequestSecurityToken`. O elemento `Header` da mensagem, além de dados de segurança, contém um elemento `Action`, que é usado para identificar a versão da norma *WS-Trust*, como anteriormente referido.

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope" ...>
  <s:Header>
    <a:Action ...>http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Issue</a:Action>
    ...
  </s:Header>
  <s:Body u:Id=" 3">
    <t:RequestSecurityToken xmlns:t="http://schemas.xmlsoap.org/ws/2005/02/trust">
      <t:RequestType>http://schemas.xmlsoap.org/ws/2005/02/trust/Issue</t:RequestType>
      <wsp:AppliesTo xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
        <EndpointReference xmlns="http://www.w3.org/2005/08/addressing">
          <Address>http://www.jf rp.com/RP/service.svc/test2</Address>
          <Identity xmlns="http://schemas.xmlsoap.org/ws/2006/02/addressingidentity">
            <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
              <X509Data>
                <X509Certificate>MIIB7DCC...</X509Certificate>
              </X509Data>
            </KeyInfo>
          </Identity>
        </EndpointReference>
      </wsp:AppliesTo>
      <t:Entropy>
        <t:BinarySecret ...>cfSn4a...</t:BinarySecret>
      </t:Entropy>
      <t:TokenType>http://...SAMLV1.1</t:TokenType>
      <t:KeyType>http://.../SymmetricKey</t:KeyType>
      <t:Claims>
        <wsid:ClaimType Uri= "http://.../claims/EmailAddress" ...></wsid:ClaimType>
      </t:Claims>
      ...
    </t:RequestSecurityToken>
  </s:Body>
</s:Envelope>
```

Figura 5.8 – Amostra de uma mensagem SOAP com um elemento RST

Na recepção de uma mensagem SOAP/RST, o elemento `RequestSecurityToken` é retirado do elemento `Body` da mensagem e é passado ao método `CreateFrom` (Figura 5.9) da classe `SecurityTokenService` (Figura 5.2), para o processamento e devolução de uma instância de um objecto RST.

```
private RST CreateFrom(XmlReader xr, XNamespace namespaceUri)
{
  XElement xRst = XElement.ReadFrom(xr) as XElement;
  RST rst = new RST(namespaceUri);
  foreach (IRSTProcessor rstp in AddInTool.RstProcessorAddIns)
    rstp.ProcessMessage(rst, xRst);
  return rst;
}
```

Figura 5.9 – Processamento do RST

Antes de dar inicio ao processamento, é instanciado um novo objecto RST, passando como parâmetro de iniciação a versão da norma, previamente processada.

A classe RST (Figura 5.10) contém um conjunto de propriedades tipificadas que representam os elementos constituintes da mensagem RST, comuns às diferentes versões da norma *WS-Trust*. A classe RST também disponibiliza o contentor `Extensions` para a adição de componentes contempladas nas extensões à norma *WS-Trust*, como por exemplo a extensão ISIP. O contentor recebe como indexador o nome do novo elemento, e recebe como valor o conteúdo desse novo elemento.

Para a classe RST optou-se por uma solução fortemente tipificada, ao contrário de outras implementações (ex: SharpSTS [37]). As variações entre ambas as normas contempladas, e da futura norma 1.4, são mínimas e não afectam a estrutura da classe. Contudo, caso haja inclusão de informação não contemplada, o uso do contentor `Extensions` evita a necessidade de uma nova estrutura.

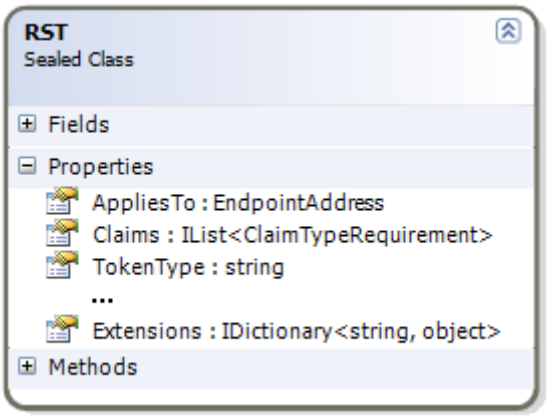

Figura 5.10 – Classe RST

O objecto RST instanciado e a mensagem são, em seguida, introduzidos num novo *pipeline* de processamento. Este *pipeline* é definido no elemento `rstProcessor` do ficheiro de configuração, através da inclusão dos módulos de extensão, na ordem desejada de processamento.

O objectivo dos módulos de extensão RST é transformar a informação descrita em XML, especifica das várias normas XML e WS-*, nas propriedades existentes na classe RST, ou, caso não exista, inclui-la no contentor `Extensions`.

No *pipeline* poderão existir um ou mais módulos que implementam diferentes versões da norma. A decisão de qual deles processará o pedido RST é decidido dentro da operação do módulo, com base na informação inicialmente passada ao construtor do objecto RST.
O processo termina com a devolução de uma instância do objecto RST ao *pipeline* de emissão, contendo toda a informação necessária para as fases seguintes.

5.4.2 Instanciação da classe RSTR

Antes de passar às fases seguintes do *pipeline* de emissão e para a melhoria no desempenho do seu processamento, o objecto RSTR (Figura 5.11), que será usado no final para a geração da mensagem *RequestSecurityTokenRequest*, é instanciado recebendo como parâmetro o objecto RST previamente criado e processado. Ambos os objectos são usados nas fases seguintes do *pipeline* de emissão, o RST como fonte de informação e o RSTR como destino de informação.
Os detalhes do RSTR serão apresentados na última fase do *pipeline* de emissão.

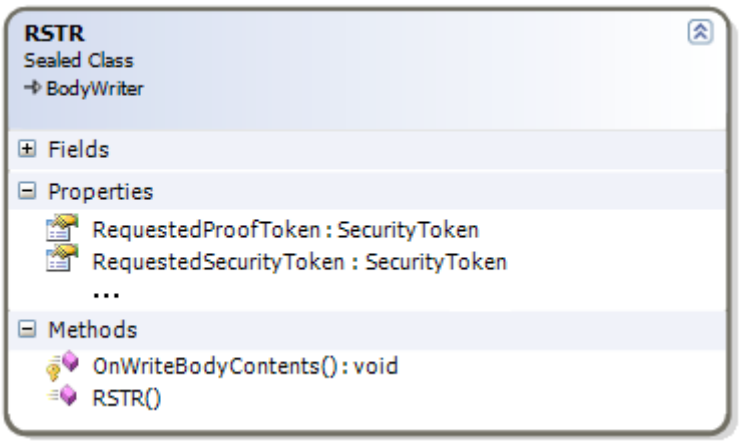

Figura 5.11 – Classe RSTR

5.4.3 Geração do *proof token*

Como especificado na norma *WS-Trust*, um *security token* emitido pelo STS poderá conter um *proof token* simétrico, assimétrico ou nenhum. O elemento `KeyType` da mensagem RST, define qual o tipo de *proof token* a inserir no *security token*.
O *pipeline* de emissão contém um ponto de extensão para incluir um único módulo de extensão para o processamento do *proof token* pedido. Para o processamento, e como definido na sua interface (Figura 5.4), o módulo recebe um objecto RST, contendo a informação necessária para a criação do *proof token*, e um objecto RSTR, que poderá receber informação consequente da criação do *proof token*.

5.4.3.1 *Proof* simétrico

A norma *WS-Trust* especifica que sempre que seja requisitado um *proof token* simétrico, o cliente terá que fornecer uma chave provisória no pedido RST. Esta chave provisória é usada

na geração da chave de sessão a incluir no *proof token*. A chave de sessão é a chave secreta usada na protecção das mensagens trocadas entre o cliente e o RP. No RST a chave provisória está contida no elemento `Entropy`, que por sua vez está contido num elemento `BinarySecret`, como exemplificado na Figura 5.8. A dimensão da chave deverá ser igual à especificada na política do RP, e caso não seja especificada, o cliente deverá requisitar uma com pelo menos 256 bits e submeter ao STS uma chave provisória de igual dimensão.

A norma especifica vários cenários onde o STS combina a chave provisória com outra chave fornecida pelo próprio STS, ou poderá ser aproveitado apenas uma das chaves, do cliente ou do STS, para a chave de sessão.

Sempre que é especificado uma chave simétrica, o módulo de extensão implementado, fornecido para o processamento do *proof token*, gera uma nova chave aleatória com a dimensão especificada no RST, e caso o cliente forneça a chave provisória, combina-a com a chave gerada para a obtenção da chave final de sessão. Após a obtenção da chave de sessão, a chave é anexa ao RSTR de duas formas distintas (Figura 5.12). As duas formas distintas asseguram que a chave de sessão é obtida apenas pelos dois destinatários. Para a obtenção por parte do requerente, a chave é simplesmente anexa ao RSTR, pois apenas o requerente poderá ver o conteúdo do RSTR. Para a obtenção por parte do RP a chave é cifrada assimetricamente com a chave pública do RP, contida num certificado X.509 previamente configurado no serviço do STS, e é inserida no *security token* destinado ao RP. Desta forma ambos os intervenientes, cliente e RP, têm a garantia da autenticidade das mensagens trocadas entre si, evitando assim ataques to tipo *man in the middle* [38].

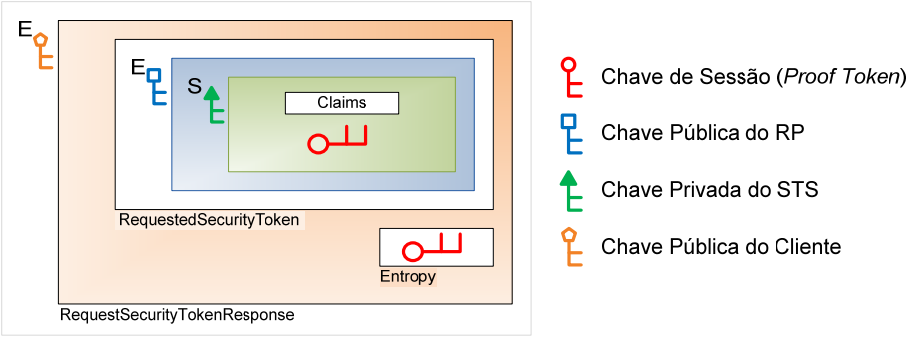

Figura 5.12 – Inserção do *Proof Token* simétrico no elemento RSTR[9]

E denota cifra, **S** denota assinatura

5.4.3.2 *Proof* assimétrico

Sempre que seja requisitado um *proof token* assimétrico, o requerente tem que gerar um par de chaves assimétricas RSA com pelo menos 1024 bits de dimensão, e a chave pública do par gerado é inserida no elemento `UseKey` do RST (Figura 5.13).

Neste caso, a tarefa do módulo de extensão é apenas preparar a chave pública recebida no RST para posterior inserção no *security token*. Não há a necessidade de inclui-la no RSTR,

[9] Adaptado de [39]

fora do *security token*, como indicado para o *proof token* simétrico, porque o requerente já tem na sua posse a chave privada do par gerado para poder cifrar as mensagens assimetricamente entre o cliente e o RP.

```
<t:RequestSecurityToken xmlns:t="http://schemas.xmlsoap.org/ws/2005/02/trust">
 ...
 <t:KeySize>1024</t:KeySize>
 <t:KeyType>http://schemas.xmlsoap.org/ws/2005/02/trust/PublicKey</t:KeyType>
 <UseKey xmlns="http://schemas.xmlsoap.org/ws/2005/02/trust">
   <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
     <KeyValue>
       <RSAKeyValue>
         <Modulus>9MefLZzp...</Modulus>
         <Exponent>AQAB</Exponent>
       </RSAKeyValue>
     </KeyValue>
   </KeyInfo>
 </UseKey>
</t:RequestSecurityToken>
```

Figura 5.13 – Chave pública para o *proof token* assimétrico

Até à data os *bindings* predefinidos da plataforma WCF, necessários do lado do STS, não suportam o *proof token* assimétrico e nem há forma de especificar o suporte no ficheiro de configuração. Assim sendo, houve a necessidade de alterar o comportamento do *binding* através do código (Figura 5.14) [40]. Esta alteração passa pela instanciação de um objecto `RsaSecurityTokenParameters` (`System.ServiceModel.Security.Tokens`) [41], usado para a parametrização de *bindings* que necessitam da cifra assimétrica RSA.

```
static void ConfigureAsymmetricBinding(ServiceEndpoint stsEndpoint)
{
 BindingElementCollection bindingElements = stsEndpoint.Binding.CreateBindingElements();
 SecurityBindingElement sbe = bindingElements.Find<SecurityBindingElement>();
 RsaSecurityTokenParameters rsaParams = new RsaSecurityTokenParameters();
 rsaParams.InclusionMode = SecurityTokenInclusionMode.Never;
 rsaParams.RequireDerivedKeys = false;
 SupportingTokenParameters requirements = new SupportingTokenParameters();
 requirements.Endorsing.Add(rsaParams);
 sbe.OptionalOperationSupportingTokenParameters.Add(Constants.WSTrust.Actions0502.Issue,
  requirements);
 stsEndpoint.Binding = new CustomBinding(bindingElements);
}
```

Figura 5.14 – Configuração do *binding* do STS para suportar *proof tokens* assimétricos

5.4.4 Processamento de *claims*

A operação de processamento das *claims* requeridas no RST, de entre as várias operações atrás identificadas, é a operação mais complexa de generalizar e de flexibilizar para todas as possibilidades aplicacionais. Para a implementação de um sistema que produza o conjunto de *claims* requeridas pelo RP teremos que considerar as seguintes entradas no sistema:

- Um conjunto de *claims* oriundo da autenticação do cliente perante o STS. A autenticação é resultado da apresentação de um *security token* por parte do cliente, podendo ser um *token* Kerberos, X.509, um par nome/palavra-chave, etc.
- Um conjunto de políticas do STS para a transformação de *claims* de entrada em *claims* de saída. Este conjunto de políticas poderá visar permissões de acesso, a minimização

de informação, tradução, etc. Este processo poderá ser um processo iterativo, com sucessivas transformações dependendo do tipo de *claim* ou do seu conteúdo.

- O conjunto de *claims* requeridas por parte do RP e contidas no RST.
- Informação proveniente da integração com sistemas como o *Active Directory* e o *Authorization Manager*.

Na Figura 5.15 apresenta-se a arquitectura implementada para o processamento de *claims*. O objectivo desta arquitectura não é o uso de todas as funcionalidades em simultâneo, mas sim combiná-las para diferentes aplicações ou contextos. As possíveis combinações dos módulos de extensão ficam ao critério do *Application Assembler* e das necessidades do sistema.

Nas seguintes secções apresentam-se os vários mecanismos e os diferentes módulos de extensão implementados para o processamento de *claims*. Alguns destes módulos (`ICModuleISIP`, `ICModuleAzMan` e `ICModuleOpenID`) são descritos nas secções ***Information Card Model***, ***Authorization Manager*** e ***OpenID*** do capítulo **Extensões à implementação base**.

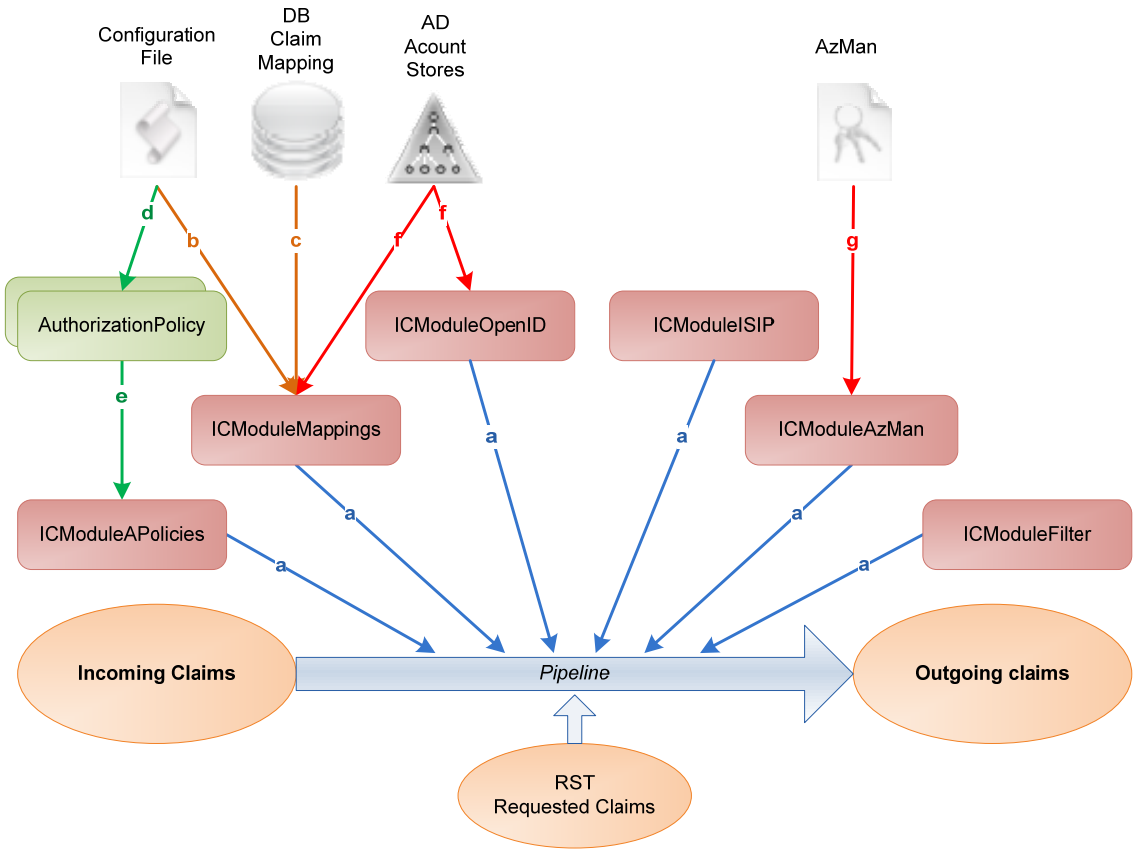

Figura 5.15 – Arquitectura do sistema de processamento de *claims*
(ICModule = IssuedClaimsModule)

O processamento dos módulos de extensão pertencentes a esta fase do *pipeline* de emissão é efectuado dentro de um novo *pipeline*. Os módulos de extensão e a sua ordem de processamento são definidos pelo *Application Assembler* (Figura 5.15 **a**).

Caso o *Application Assembler* inclua no *pipeline* o módulo `IssuedClaimsModuleMappings`, terá que também definir a origem das políticas de mapeamento de *claims* (Figura 5.15 **b**, **c**), cujas políticas são definidas pelo *Policy Administrators*. Este módulo poderá ainda obter informação do *Active Directory* ou de outro repositório de identidades digitais (Figura 5.15 **f**). Este módulo é descrito em detalhe nas secções **Politicas de mapeamento no ficheiro de configuração** e **Politicas de mapeamento na base de dados de gestão do STS**.
Se o *Application Assembler* incluir no *pipeline* o módulo `IssuedClaimsModulePolicies`, terá que definir também, através do ficheiro de configuração (Figura 5.15 **d**), quais os *Authorization Policies* a incluir no processamento do módulo (Figura 5.15 **e**). Este módulo é descrito em detalhe na secção **Uso de IAuthorizationPolicy**.

Nesta fase do *pipeline* de emissão, a maioria dos procedimentos não são especificados ou baseados em normas. Assim sendo, alguns dos módulos de extensão, implementados para o processamento de *claims*, servem apenas de exemplo de uma possível implementação, e demonstram a integração do *STSLib* com outros sistemas, como por exemplo o *Active Directory* e o *AzMan*.

5.4.4.1 Preparação para o processamento de *claims*

A operação de processamento de *claims* inicia-se com a criação um novo contentor de *claims* vazio (Figura 5.16). Esse contentor em conjunto com o objecto RST é passado ao *pipeline* de processamento *claims*. Caso o processamento termina com um contentor vazio, uma excepção é enviada ao requerente.

```
private Collection<Claim> GetIssuedClaims(RST rst)
{
  Collection<Claim> claims = new Collection<Claim>();
  foreach (IIssuedClaimsProcessor icp in AddInTool.IssuedClaimsProcessorAddIns)
    icp.GetIssuedClaims(rst, claims);
  if (claims.Count == 0)
    throw new FaultException("STS Exception:The processed issued claims set is empty!");
  return claims;
}
```

Figura 5.16 – Preparação das *claims* para processamento

5.4.4.2 Politicas de mapeamento no ficheiro de configuração

Para a configuração de politicas de mapeamento pelo *Policy Administrator* adicionou-se ao ficheiro de configuração um novo elemento que visa a definição dos mapeamentos desejados (Figura 5.17). Esta configuração permite definir configurações independentes para cada RP. Desta forma pode-se construir politicas não só dependentes das *claims* de entrada e de saída mas também dependentes do contexto. Esta configuração também possibilita a transformação do tipo de *claims* e do seu conteúdo.
Para exemplificar o uso destes mapeamento disponibilizou-se um módulo de extensão (Figura 5.18) que, depois de obter as *claims* de autenticação, processa as *claims* recebidos em conjunto com as *claims* a emitir e as politicas configuradas para o RP destinatário. O módulo

disponibiliza ainda dois métodos auxiliares, cuja implementação está dependente de cada aplicação do STS. O primeiro método `ProcessClaim`, recebe uma combinação das *claims* de entrada, de saída e um mapeamento. Esse método terá que validar essa combinação, e caso a politica o defina, insere uma ou mais *claims* no contentor de retorno. O segundo método serve de auxílio ao primeiro método caso haja necessidade da transformação de uma *claim*. A obtenção ou transformação de *claims* poderá implicar a integração ou a interrogação a sistemas como o *Active Directory*.

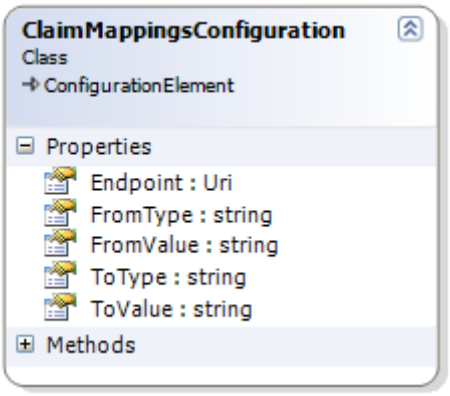

Figura 5.17 – ClaimMapping no ficheiro de configuração

```
public class IssuedClaimsModuleMappings : IIssuedClaimsProcessor
{
  public void GetIssuedClaims(RST rst, Collection<Claim> claims)
  {
    ReadOnlyCollection<ClaimSet> claimSets =
      OperationContext.Current.ServiceSecurityContext.AuthorizationContext.ClaimSets;
    foreach (ClaimSet claimSet in claimSets)
      foreach (Claim claim in claimSet)
        foreach (ClaimMapping cm in
            StsConfiguration.Current.ClaimMappings.GetMappingByEndpoint(rst.AppliesTo))
          foreach (ClaimTypeRequirement ctr in rst.Claims)
            ProcessClaim(claims, claim, cm, ctr);
  }
  private void ProcessClaim(Collection<Claim> claims,
    Claim claim, ClaimMapping cm, ClaimTypeRequirement ctr) {...}
  private Claim TransformClaim(Claim claim) {...}
}
```

Figura 5.18 – Módulo de extensão IssuedClaimsModuleMappings

5.4.4.3 Politicas de mapeamento na base de dados de gestão do STS

Além da configuração de politicas de mapeamento através de ficheiros de configuração, implementou-se também a possibilidade da definição de políticas através da base de dados relacional apresentada na Figura 5.19.

Para a gestão dos dados contidos na base de dados implementou-se o *STSMan* referenciado no capítulo anterior. O *STSMan* visa dar aos *Policy Administrators* a capacidade de definir diferentes configurações online, retirando a necessidade da configuração local inerente aos ficheiros de configuração. O *STSMan* é descrito em detalhe na secção **Aplicação *web* de gestão do STS** do capítulo **Extensões à implementação base**.

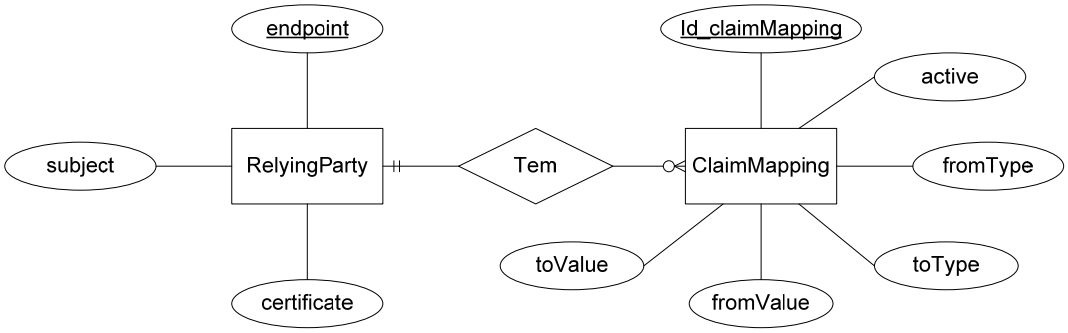

Figura 5.19 – Base de dados para a especificação de Relying Parties e as respectivas políticas de mapeamento de *claims*

Os mapeamentos definidos na base de dados destinam-se ao mesmo módulo de extensão descrito na secção anterior. O módulo de extensão não distingue a origem dos mapeamentos, base de dados ou ficheiro de configuração. A distinção é efectuada dependente da configuração definida pelo *Application Assembler*.

A possibilidade da especificação de diferentes RPs e os respectivos certificados e políticas de mapeamentos de *claims*, pelos *Policy Administrators* do STS e através da base de dados, melhora a versatilidade e a simplicidade de configuração, além de evitar a necessidade de instalar os certificados dos RPs no repositório de certificados do sistema destino do STS.

5.4.4.4 Uso de IAuthorizationPolicy

A plataforma .Net dispõe de um conjunto de classes e interfaces para o processamento de *claims*. Definem-se políticas de autorização através da implementação da interface `IAuthorizationPolicy` (`System.IdentityModel.Policy`). Esta implementação visa a adição ou o mapeamento de um conjunto de *claims* noutro, após a avaliação do conjunto actual. Na Figura 5.20 apresenta-se um exemplo dessa implementação. Neste exemplo as *claims* oriundos da autenticação, mais uma *claim* do tipo e-mail, são adicionadas ao conjunto actual (`evaluationContext`).

```
public class CustomAuthorizationPolicy : IAuthorizationPolicy
{
  public bool Evaluate(EvaluationContext evaluationContext, ref object state)
  {
    IList<Claim> claims = new List<Claim>();
    ReadOnlyCollection<ClaimSet> claimSets =
      OperationContext.Current.ServiceSecurityContext.AuthorizationContext.ClaimSets;

    foreach (ClaimSet cs in claimSets)
      foreach (Claim c in cs.FindClaims(ClaimTypes.Name, Rights.PossessProperty))
        claims.Add(c);

    claims.Add(new Claim(ClaimTypes.Email, "jhfigueiredo@netvisao.pt",
      Rights.PossessProperty));
    evaluationContext.AddClaimSet(this, new DefaultClaimSet(this.Issuer, claims));
    return true;
  }
  ...
}
```

Figura 5.20 – Implementação de um IAuthorizationPolicy

As várias implementações da interface `IAuthorizationPolicy` são adicionadas à classe `AuthorizationContext` (`System.IdentityModel.Policy`) para avaliação. Do resultado dessa avaliação obtém-se um conjunto de *claims*. A classe `AuthorizationContext` chama o método `Evaluate` para cada uma das implementações de `IAuthorizationPolicy`. Este método determina se novas *claims* deverão ser adicionadas ao conjunto actual, e é chamado várias vezes, para cada política, sempre que sejam adicionadas novas *claims* de outras políticas.

Na Figura 5.21 apresenta-se o módulo de extensão que processa as `IAuthorizationPolicy` implementadas. As diversas implementações são obtidas e instanciadas através do método `GetAutorizationPolicies` da classe `AddInTool`, já referida, e são devolvidas à classe `AuthorizationContext` para avaliação. Após a avaliação, o conjunto de *claims* resultante é adicionado às *claims* a devolver ao *pipeline*.

```
class IssuedClaimsModuleAPolicies : IIssuedClaimsProcessor
{
  public void GetIssuedClaims(RST rst, Collection<Claim> claims)
  {
    AuthorizationContext context =
      AuthorizationContext.CreateDefaultAuthorizationContext(
        AddInTool.GetAuthorizationPolicies());

    foreach (ClaimSet claimset in context.ClaimSets)
      foreach (Claim claim in claimset)
        claims.Add(claim);
  }
}
```

Figura 5.21 – Módulo de extensão IssuedClaimsModuleAPolicies

Para a inclusão de implementações de `IAuthorizationPolicy` na configuração do STS, adicionou-se ao ficheiro de configuração o novo elemento `claimPolicies` (Figura 5.22).

```
<claimPolicies>
  <add type="CustomAuthorizationPolicy" lib="ClaimPolicyAddIns.dll"/>
</claimPolicies>
```

Figura 5.22 – Configuração de AuthorizationPolicies

5.4.4.5 Filtragem

O processamento das *claims* através do uso de `IAuthorizationPolicy`, como descrito na secção anterior, resulta por vezes num conjunto de *claims* contendo *claims* além daquelas pedidas no RST. Para filtrar o conjunto de *claims* fornecidos ao *pipeline* de emissão, implementou-se o módulo de extensão `IssuedClaimsModuleFilter` (Figura 5.23). Este módulo filtra o conjunto de *claims* recebidas, devolvendo apenas as *claims* requeridas.

```
class IssuedClaimsModuleFilter : IIssuedClaimsProcessor
{
  public void GetIssuedClaims(RST rst, Collection<Claim> claims)
  {
    Collection<Claim> finalClaims = new Collection<Claim>();
    foreach (Claim claim in claims)
      foreach (ClaimTypeRequirement ctr in rst.Claims)
```

```
        if (string.Compare(claim.ClaimType, ctr.ClaimType, true) == 0)
        {
          finalClaims.Add(claim);
          break;
        }
        claims = finalClaims;
  }
}
```

Figura 5.23 – Módulo de extensão IssuedClaimsModuleFilter

5.4.5 Geração do *Security Token*

No último ponto de extensão (Figura 5.24), o *proof token*, o conjunto de *claims* requeridas e a instância do RSTR são passadas a cada um dos módulos de extensão configurados, mas apenas um dos módulos irá gerar o *security token* pedido. Como referido no capítulo anterior, os módulos implementados para este ponto de extensão têm como objectivo a geração do *security token* com o formato especificado no objecto RST. Para este ponto de extensão foram implementados dois módulos, uma para gerar *security tokens* SAML e outra para gerar *security tokens* OpenID, este último será apresentado no capítulo **Extensões à implementação base**.

```
private void GenerateIssuedToken(string stsName, SecurityKeyIdentifier proof,
    Collection<Claim> claims, RSTR rstr)
{
  foreach (IIssuedTokenProcessor itp in AddInTool.IssuedTokenProcessorAddIns)
    itp.GenerateIssuedToken(stsName, proof, claims, rstr);
  ...
}
```

Figura 5.24 – Processamento do *Security Token*

5.4.5.1 *Security tokens* SAML

O módulo de extensão dedicado à geração de um *security token* SAML (Figura 5.25), em conjunto com os dados recebidos atrás referidos, verifica primeiro se o tipo de *token* requerido é do tipo SAML, se sim, o módulo prossegue com a geração do *token*.

```
public class IssuedTokenModuleSAML : IIssuedTokenProcessor
{
  public void GenerateIssuedToken(string stsName, SecurityKeyIdentifier proof,
      Collection<Claim> claims, RSTR rstr)
  {
    if (rstr.RST.TokenType != Constants.SAML.NamespaceUri)
      return;
    ...
    SamlAssertion samlAssertion = new SamlAssertion(...);
    X509SecurityToken issuerToken = StsConfiguration.Current.GetStsToken();
    ...
    samlAssertion.SigningCredentials =
      new SigningCredentials(issuerToken.SecurityKeys[0],…);
    SamlSecurityToken samlToken = new SamlSecurityToken(samlAssertion);
    rstr.RequestedSecurityToken = samlToken;
    ...
  }
}
```

Figura 5.25 – Módulo de extensão IssuedTokenModuleSAML

Para a geração do *security token* SAML é instanciado um objecto do tipo `SamlAssertion` a partir das *claims* e do *proof token* recebidos. De seguida é obtido o *token* que representa o

certificado X509 do STS, a partir da configuração definida pelo *Deployer* no ficheiro de configuração, e que irá servir para assinar o objecto anteriormente instanciado. Para finalizar, o *token* do tipo `SamlSecurityToken` é gerado e inserido no objecto RSTR.

Na Figura 5.26 é apresentada uma amostra, em formato XML, de um *security token* SAML, onde se identificam as duas *claims* contidas e a assinatura do STS.

```
<saml:Assertion MajorVersion="1" MinorVersion="1" AssertionID=" 67..." Issuer="MySTS"
    IssueInstant="2008-04-30T19:45:15.364Z"
    xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">
  <saml:Conditions
      NotBefore="2008-04-30T19:45:15.364Z" NotOnOrAfter="2008-04-30T20:45:15.364Z"/>
  <saml:AttributeStatement>
    <saml:Subject>
      <saml:SubjectConfirmation>
        <saml:ConfirmationMethod>…holder-of-key</saml:ConfirmationMethod>
      </saml:SubjectConfirmation>
    </saml:Subject>
    <saml:Attribute AttributeName="emailaddress"
        AttributeNamespace="http://schemas.xmlsoap.org/ws/2005/05/identity/claims">
      <saml:AttributeValue>jhfigueiredo@netvisao.pt</saml:AttributeValue>
    </saml:Attribute>
    <saml:Attribute AttributeName="privatepersonalidentifier"
        AttributeNamespace="http://schemas.xmlsoap.org/ws/2005/05/identity/claims">
      <saml:AttributeValue>im/PF6R…</saml:AttributeValue>
    </saml:Attribute>
    </saml:AttributeStatement>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>...</SignedInfo>
    <SignatureValue>q6ZS...</SignatureValue>
    <KeyInfo>...</KeyInfo>
  </Signature>
</saml:Assertion>
```

Figura 5.26 – Amostra do *Security Token* SAML

5.4.6 Geração da mensagem RequestSecurityTokenResponse

A última fase do *pipeline* de emissão consiste na transformação do objecto RSTR para o formato XML, de forma a inclui-lo no corpo da mensagem SOAP a devolver ao requerente. Nas mensagens, o cabeçalho é *buffered* enquanto o corpo é *streamed*, por isso a classe passada ao construtor da mensagem terá que saber como escrever o corpo da mensagem quando lhe for pedido. Para isso, a classe `RSTR` (Figura 5.11) deriva da classe `BodyWriter`. Esta classe possui o método `OnWriteBodyContents`, que terá que ser redefinido pela classe derivada e chamada sempre que seja necessário escrever o corpo da mensagem, através de um `XmlWriter`.

Esta fase é processada através de um ponto de extensão inserido no método `OnWriteBodyContents` atrás referenciado (Figura 5.27). O processamento do método inicia-se com a instanciação da classe `XDocument`. Esta instância é passado em conjunto com o objecto RSTR recebido da fase anterior do *pipeline*, a cada um dos módulos de extensão configurados pelo *Application Assembler* para esta fase. Cada um dos módulos irá construir ou preencher a estrutura XML correspondente ao objecto RSTR, na instância `XDocument`. Após a obtenção do `XDocument` totalmente processado, o seu conteúdo é escrito no corpo da mensagem, através de um `XmlWriter`.

```
protected override void OnWriteBodyContents(XmlDictionaryWriter writer)
{
  XDocument xRstr = new XDocument();
  foreach (IRSTRProcessor rstrp in AddInTool.RstrProcessorModules)
    rstrp.ProcessRstr(this, ref xRstr);
  xRstr.Root.WriteTo(writer);
}
```

Figura 5.27 – Processamento da mensagem RSTR

Na Figura 5.28 apresenta-se o módulo de extensão `RSTRModule0502`, usado na construção da mensagem RSTR como especificado na norma *WS-Trust* de Fevereiro de 2005. O módulo `RSTRModule13`, implementado para a versão 1.3 da norma, difere do apresentado essencialmente pela necessidade de inserir o elemento `RequestSecurityTokenResponse`, já definido na norma anterior, dentro de um elemento `RequestSecurityTokenResponseCollection`.

```
public class RSTRModule0502 : IRSTRProcessor
{
  string  wst = Constants.WSTrust.NamespaceUri0502;
  WSSecurityTokenSerializer  ser = new WSSecurityTokenSerializer();

  public void ProcessRstr(RSTR rstr, ref XDocument xRstr)
  {
    if (rstr.RST.NamespaceUri != Constants.WSTrust.NamespaceUri0502)
      return;
    MemoryStream ms = new System.IO.MemoryStream();
    XmlDictionaryWriter writer =
      XmlDictionaryWriter.CreateDictionaryWriter(XmlWriter.Create(ms));
    EncryptedWSSecurityTokenSerializer eser =
      new EncryptedWSSecurityTokenSerializer(rstr.RST);

    // <wst:RequestSecurityTokenResponse>
    writer.WriteStartElement("wst",
      Constants.WSTrust.Elements.RequestSecurityTokenResponse, _wst);
    ...
    if (rstr.RequestedSecurityToken != null)
    {   // <wst:RequestedSecurityToken>
      writer.WriteStartElement("wst",
        Constants.WSTrust.Elements.RequestedSecurityToken,  wst);
      eser.WriteToken(writer, rstr.RequestedSecurityToken);
      writer.WriteEndElement();
    }   // </wst:RequestedSecurityToken>
    ...
    if (rstr.RequestedProofToken != null)
    {   // <wst:RequestedProofToken>
      writer.WriteStartElement("wst",
        Constants.WSTrust.Elements.RequestedProofToken, _wst);
       ser.WriteToken(writer, rstr.RequestedProofToken);
      writer.WriteEndElement();
    }   // </wst:RequestedSecurityToken>
    ...
    writer.WriteEndElement();   // </wst:RequestSecurityTokenResponse>

    writer.Flush();
    ms.Position = 0;
    XmlReader xr = XmlReader.Create(ms);
    xRstr = XDocument.Load(xr);
  }
}
```

Figura 5.28 – Módulo de extensão RSTRModule0502

Para serializar alguns dos elementos para XML, como por exemplo o *proof token*, foi necessário instanciar um objecto da classe `WSSecurityTokenSerializer` (`System.ServiceModel.Security`), que possui métodos especializados nestas tarefas. Contudo, este objecto não possui a capacidade de cifrar a informação serializada, que é necessário sempre que seja especificado no RST. Para isso, implementou-se um novo tipo de classe, `EncryptedWSSecurityTokenSerializer` (Figura 5.29), derivado da classe `WSSecurityTokenSerializer`.

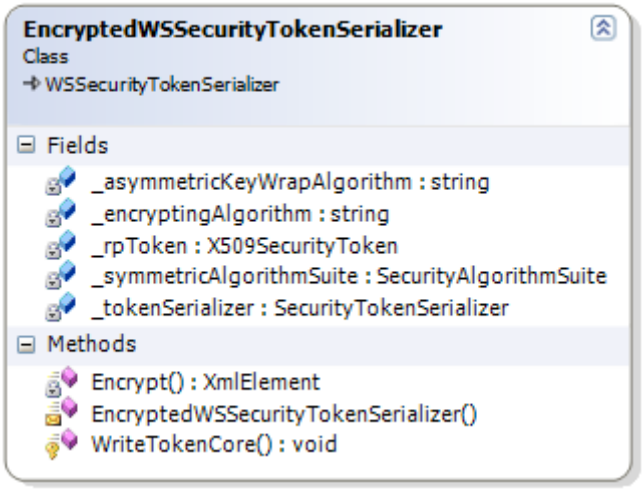

Figura 5.29 – EncryptedWSSecurityTokenSerializer

A instância desta nova classe é construída com informação de cifra, proveniente do pedido RST, e com o certificado X509 do RP. Este certificado contem a chave pública do RP, através do qual esta classe cifra o *security token*.

Na Figura 5.30 apresenta-se uma amostra de uma mensagem SOAP contendo no corpo uma resposta RSTR. Nesta amostra evidencia-se a presença do *security token* (`RequestedSecurityToken`) cifrado e do *proof token* (`Entropy`).

```
<s:Envelope xmlns:a="http://www.w3.org/2005/08/addressing"
    xmlns:s="http://www.w3.org/2003/05/soap-envelope">
 <s:Header>
   <a:Action s:mustUnderstand="1">
       http://schemas.xmlsoap.org/ws/2005/02/trust/RSTR/Issue</a:Action>
   <a:RelatesTo>urn:uuid:69c567eb-c1a3-4369-ab7f-f121e93bf7f4</a:RelatesTo>
 </s:Header>
 <s:Body>
   <wst:RequestSecurityTokenResponse
       xmlns:wst="http://schemas.xmlsoap.org/ws/2005/02/trust">
     <wst:TokenType>
         http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1
     </wst:TokenType>
     <wst:RequestedSecurityToken>
       <enc:EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
           xmlns:enc="http://www.w3.org/2001/04/xmlenc#">...</enc:EncryptedData>
     </wst:RequestedSecurityToken>
     <wst:RequestedAttachedReference>...</wst:RequestedAttachedReference>
     <wst:RequestedUnattachedReference>...</wst:RequestedUnattachedReference>
     <wsp:AppliesTo
         xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">...</wsp:AppliesTo>
     <wst:RequestedProofToken>
       <wst:ComputedKey>
```

```
            http://schemas.xmlsoap.org/ws/2005/02/trust/CK/PSHA1</wst:ComputedKey>
      </wst:RequestedProofToken>
      <wst:Entropy>
         <t:BinarySecret u:Id="uuid…" xmlns:u="…" xmlns:t= "…">ZWT...</t:BinarySecret>
      </wst:Entropy>
   </wst:RequestSecurityTokenResponse>
 </s:Body>
</s:Envelope>
```

Figura 5.30 – Amostra de uma mensagem SOAP com um RSTR

5.5 Configuração do STS

Nesta secção apresenta-se a forma de configuração dos serviços STS, detalhando aspectos de configuração para diferentes aplicações e papeis do STS. Apresenta-se em primeiro lugar o uso exclusivo de ficheiros de configuração, apresentando especificamente as opções tomadas e as suas justificações, e quais as configurações possíveis. Em segundo lugar apresenta-se as configurações para as diferentes autenticações por parte do cliente perante o STS. A configuração das políticas de emissão, através da base de dados e do *STSMan*, será detalhada na secção **Aplicação *web* de gestão do STS** do capítulo **Extensões à implementação base**.

Os ficheiros de configuração dos serviços WCF dão a capacidade aos vários intervenientes – *Application Assembler*, *Deployer* e *Policy Administrator* – de configurar os parâmetros de funcionamento desses serviços após a sua implementação. Informação mais detalhada sobre os ficheiros de configuração e dos seus elementos WCF são apresentados no **Apêndice A – *Windows Communications Foundations***.
A configuração de um serviço STS está dividida em duas secções distintas: parâmetros WCF e parâmetros gerais.

5.5.1 Parâmetros WCF

O uso de ficheiros de configuração para a definição dos parâmetros do serviço WCF, visa a definição de diferentes configurações para diferentes cenários, pelo *Deployer*, e sem a necessidade de alterar a implementação. Na Figura 5.31 apresenta-se uma amostra de uma configuração dos parâmetros WCF de um serviço STS, onde se realça os seguintes aspectos:

- ***Endpoints***: Este serviço WCF expõe três *endpoints*. Dois deles, o `wst0502` e o `wst13`, representam os dois contratos implementados e apresentados na secção **Contrato de serviço WCF para o STS**. A definição de dois contratos individuais e a necessidade de *bindings* distintos implica a definição de *endpoints* distintos. O *endpoint* `mex` é usado pelo cliente para a obtenção da metadata do serviço WCF através da norma *WS-MetadataExchange*.
- ***Bindings***: A plataforma WCF permite a definição de *custom bindings* com as propriedades pretendidas, contudo, para a maioria das aplicações, a plataforma já disponibiliza um conjunto de *bindings* predefinidos, dois dos quais reflectindo as diferenças entre as duas versões consideradas da norma WS-Trust. Por exemplo: para

a versão de Fevereiro de 2005 temos disponível o *binding* `wsHttpBinding` e para a versão 1.3 temos disponível o *binding* `ws2007HttpBinding`. Ambos os *bindings* referenciados usam o protocolo HTTP como o meio de transporte e providenciam segurança ao nível da mensagem, e neste caso especifico requerem uma autenticação *Username*/*Password*.

- ***Behaviors***: Este serviço tem associado um certificado X509, usado na autenticação do serviço e na segurança do transporte das mensagens.

```
<configuration>
  ...
  <system.serviceModel>

  <services>
    <service name="SecurityTokenServiceNS.STSImplSample"
             behaviorConfiguration="STSImplSampleBehavior">
      <endpoint address="wst0502"
                binding="wsHttpBinding"
                bindingConfiguration="usernameBinding"
                contract="SecurityTokenServiceNS.ISecurityTokenService0502"/>
      <endpoint address="wst13"
                binding="ws2007HttpBinding"
                bindingConfiguration="usernameBinding2007"
                contract="SecurityTokenServiceNS.ISecurityTokenService13"/>
      <endpoint address="mex"
                binding="mexHttpsBinding"
                contract="IMetadataExchange"/>
    </service>
  </services>

  <bindings>
    <wsHttpBinding>
      <binding name="usernameBinding">
        <security mode="Message">
          <message clientCredentialType="UserName"/>
        </security>
      </binding>
    </wsHttpBinding>
    <ws2007HttpBinding>
      <binding name="usernameBinding2007">
        <security mode="Message">
          <message clientCredentialType="UserName"/>
        </security>
      </binding>
    </ws2007HttpBinding>
  </bindings>

  <behaviors>
    <serviceBehaviors>
      <behavior name="STSImplSampleBehavior">
        <serviceCredentials>
          <serviceCertificate findValue="CN=www.jf_sts.com"
                              storeLocation="LocalMachine"
                              storeName="My"
                              x509FindType="FindBySubjectDistinguishedName"/>
        </serviceCredentials>
        <serviceMetadata httpsGetEnabled="True"/>
        <serviceDebug includeExceptionDetailInFaults="true"/>
      </behavior>
    </serviceBehaviors>
  </behaviors>

  </system.serviceModel>
</configuration>
```

Figura 5.31 – Configuração dos parâmetros WCF do STS

A autenticação do cliente perante o serviço STS poderá ainda ser configurado com os seguintes parâmetros:

- Username/Password: `<message clientCredentialType="UserName"/>`
- Certificado X509: `<message clientCredentialType=" Certificate"/>`
- Windows: `<message clientCredentialType="Windows"/>`

Cada uma destas autenticações necessita de diferentes configurações do lado da aplicação do cliente, como por exemplo a definição de um certificado X509 do cliente caso a autenticação seja *Certificate*.

5.5.2 Parâmetros Gerais

Na arquitectura da biblioteca apresentada no capítulo anterior apresentaram-se origens distintas das configurações efectuadas pelos vários intervenientes na configuração do STS. Na implementação da arquitectura, e de forma a aproveitar a necessidade do ficheiro de configuração para a configuração dos parâmetros WCF do serviço, integrou-se no mesmo ficheiro também as configurações da responsabilidade do *Application Assembler* e as do *Policy Administrator*.

Para a configuração dos parâmetros gerais do STS, da responsabilidade do *Application Assembler*, e das políticas de emissão, da responsabilidade do *Policy Administrator*, através do ficheiro de configuração, implementou-se a classe `StsConfiguration` (Figura 5.32) e um conjunto de classes associadas (`ExtensionModulesConfiguration`, `ServiceCertificateCollection`, etc). Estas classes definem quais os elementos possíveis, e os seus conteúdos, a incluir no ficheiro de configuração. Estas classes também possuem métodos para facilitar a obtenção de alguns dos dados contidos.

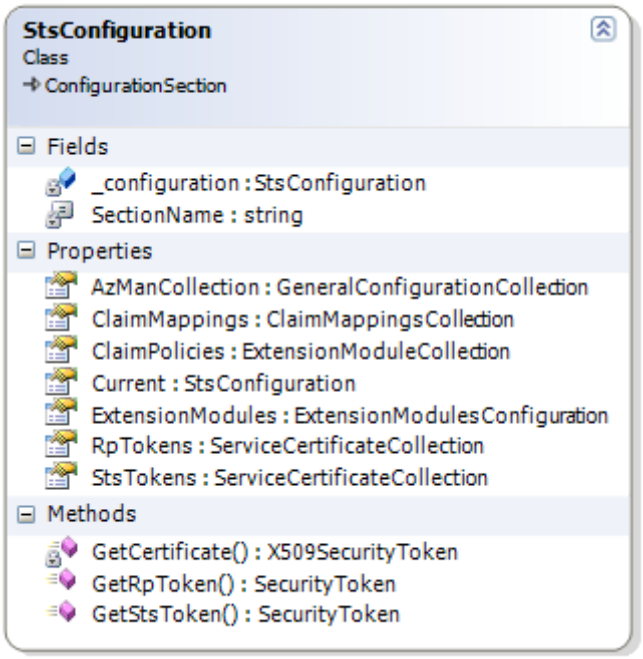

Figura 5.32 – Classe StsConfiguration

Caso o *Application Assembler* defina que a origem das políticas de emissão seja o ficheiro de configuração, o *Policy Administrator* terá que configurá-las no ficheiro, antes ou depois da inicialização do STS, nos elementos (propriedades da classe `StsConfiguration`) `RPTokens` e `ClaimMappings`. Caso o *Application Assembler* defina que a origem das políticas de emissão seja a base de dados anteriormente referenciado, o *Policy Administrator* terá que configurá-las na base de dados. As restantes configurações são da exclusiva responsabilidade do *Application Assembler*.

Na Figura 5.33 apresenta-se uma amostra de uma configuração dos parâmetros gerais de um serviço STS. Nesta amostra realça-se os seguintes aspectos:

- **Certificados**: No elemento `stsTokens` define-se o certificado associado ao STS e no elemento `rpTokens`, caso o *Application Assembler* o defina (`<rpTokens db="false">`), define-se os vários certificados associados aos RPs. Ambos estes elementos são usados nas diversas operações da emissão do *security token*.
- **Módulos de extensão**: A definição deste elemento (`extensionModules`) e do seu conteúdo (`rstProcessor`, `proofProcessor`, `etc.`) foram apresentados na secção **Arquitectura de extensões**.
- **Políticas de emissão de *claims***: Caso seja incluído no elemento `issuedClaimsProcessor` o módulo de extensão `IssuedClaimsModuleAPolicies`, o *Application Assembler* tem que incluir no elemento `claimPolicies` os módulos `AuthorizationPolicy` desejados (ver a secção **Uso de IAuthorizationPolicy**). Caso seja incluido o módulo de extensão `IssuedClaimsModuleMappings`, e caso o *Application Assembler* defina `<claimMappings db="false">`, o *Policy Administrator* terá que incluir no elemento `claimMappings` os mapeamentos de *claims* necessários (ver a secção **Politicas de mapeamento no ficheiro de configuração**).
- ***Authorization Manager***: Caso seja incluído no elemento `issuedClaimsProcessor` o módulo de extensão `IssuedClaimsModuleAzMan`, o elemento `azMan` tem que conter os dados necessários exigidos pelo módulo (ver a secção ***Authorization Manager* (AzMan)** do capítulo **Extensões à implementação base**).

```
<configuration>

 <configSections>
   <section name="stsConfiguration"
            type="SecurityTokenServiceNS.StsConfiguration, Configuration"/>
 </configSections>

 <stsConfiguration>
   <stsTokens>
     <add uri="http://schemas.xmlsoap.org/ws/2005/02/trust/RST/Issue"
          storeLocation="LocalMachine"
          storeName="My"
```

```
          findValue="CN=www.jf sts.com"
          findType="FindBySubjectDistinguishedName"/>
    </stsTokens>
    <rpTokens db="false">
      <add uri="http://www.jf_rp.com/RP/service.svc/test2"
          storeLocation="LocalMachine"
          storeName="OtherPeople"
          findValue="CN=www.jf rp.com"
          findType="FindBySubjectDistinguishedName"/>
    </rpTokens>
    <extensionModules path="C:\STS AddIns\">
      <rstProcessor>
        <add type="RSTModule0502" lib="RstAddIns.dll"/>
        <add type="RSTModule13" lib="RstAddIns.dll"/>
      </rstProcessor>
      <proofProcessor>
        <add type="ProofModule" lib="ProofAddIns.dll"/>
      </proofProcessor>
      <issuedClaimsProcessor>
        <add type="IssuedClaimsModuleAPolicies" lib="IssuedClaimsAddIns.dll"/>
        <add type="IssuedClaimsModuleFilter" lib="IssuedClaimsAddIns.dll"/>
      </issuedClaimsProcessor>
      <issuedTokenProcessor>
        <add type="IssuedTokenModuleSAML" lib="IssuedTokenAddIns.dll"/>
      </issuedTokenProcessor>
      <rstrProcessor>
        <add type="RSTRModule0502" lib="RstrAddIns.dll"/>
        <add type="RSTRModule13" lib="RstrAddIns.dll"/>
      </rstrProcessor>
    </extensionModules>
    <claimPolicies>
      <add type="CustomAuthorizationPolicy" lib="ClaimPolicyAddIns.dll"/>
    </claimPolicies>
    <claimMappings/>
    <azMan>
      <add key="path" value="C:\...\RBAC.xml"></add>
      <add key="appName" value="STS1"></add>
    </azMan>
  </stsConfiguration>
   ...
</configuration>
```

Figura 5.33 – Configuração dos parâmetros gerais do STS

6 Testes à implementação base - *Relying Party* e Sujeito (cliente)

Para a verificação da correcção da *STSLib* foram implementados vários serviços e aplicações, desempenhando os papéis de *Relying Party* e Sujeito. Alguns aspectos ligados aos contextos apresentados no capítulo seguinte, serão pormenorizados nesse capítulo.

A Figura 6.1 apresenta a 1ª plataforma de testes implementada para dar suporte ao desenvolvimento da *STSLib*. Esta plataforma é constituída por um *web service* WCF (*Relying Party*) e uma aplicação *Windows* (Cliente). Também na Figura 6.1 estão representadas as operações necessárias para os pedidos efectuados com a intervenção STS. Detalhes desta plataforma são apresentadas na secção seguinte.

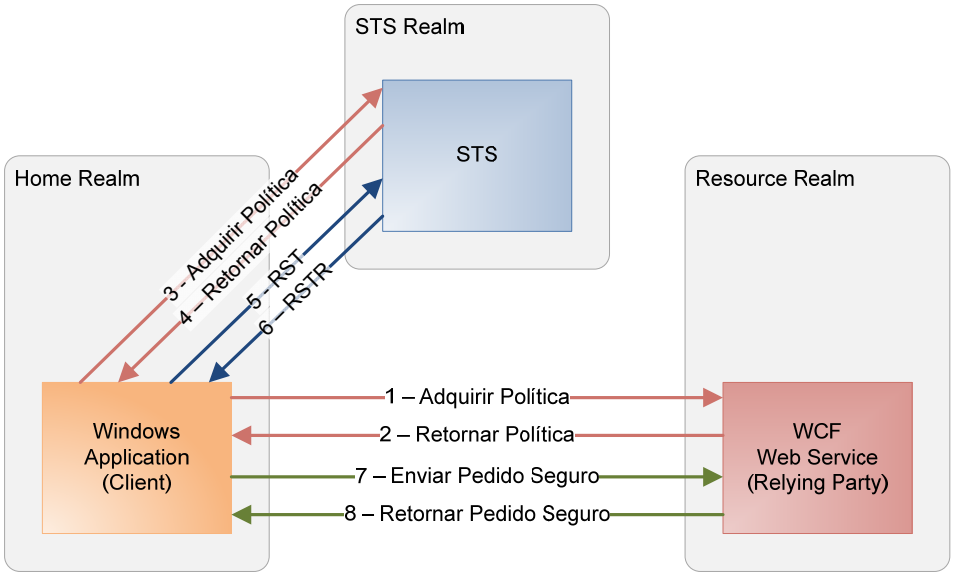

Figura 6.1 – 1ª Plataforma de testes

A Figura 6.2 apresenta a 2ª plataforma de testes, que é constituído pelo *STSMan*, um *browser* e um *Identity Selector* (Cardspace), que foi apresentado na secção ***Information Card Model*** do capítulo **Contexto e Enquadramento**. Embora a motivação principal para a implementação do *STSMan* seja a melhoria no processo de definição de políticas de emissão, por parte dos *Policy Administrators*, aproveitou-se a integração do *STSMan* com o modelo ISIP para usa-la como uma segunda plataforma de testes. O *browser* em conjunto com o *Identity Selector* também desempenha o papel de Sujeito quando o *login* é efectuado usando um *managed information card*. Este processo e os seus aspectos são apresentados na secção ***Information Card Model*** do capítulo **Extensões à implementação base**.

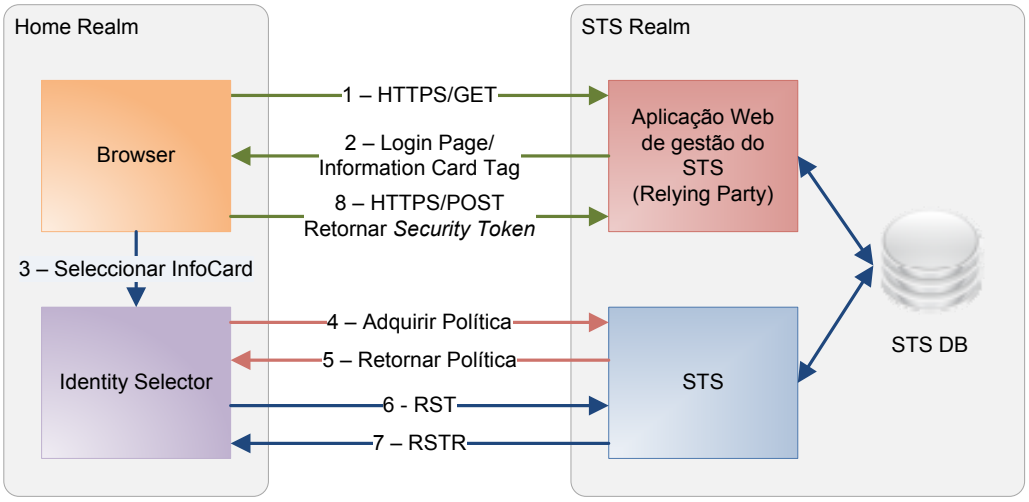

Figura 6.2 – 2ª Plataforma de testes

6.1 *Relying Party*

Implementou-se um serviço WCF para o papel de *Relying Party*, cujo objectivo é a verificação dos diferentes papéis e configurações do serviço STS. O serviço instancia a classe `RPServiceTest1` apresentada na Figura 6.3, e que implementa as interfaces WCF (`IRPServiceTest1`, `IRPServiceTest2`, …). Cada uma das interfaces disponibiliza uma operação WCF.

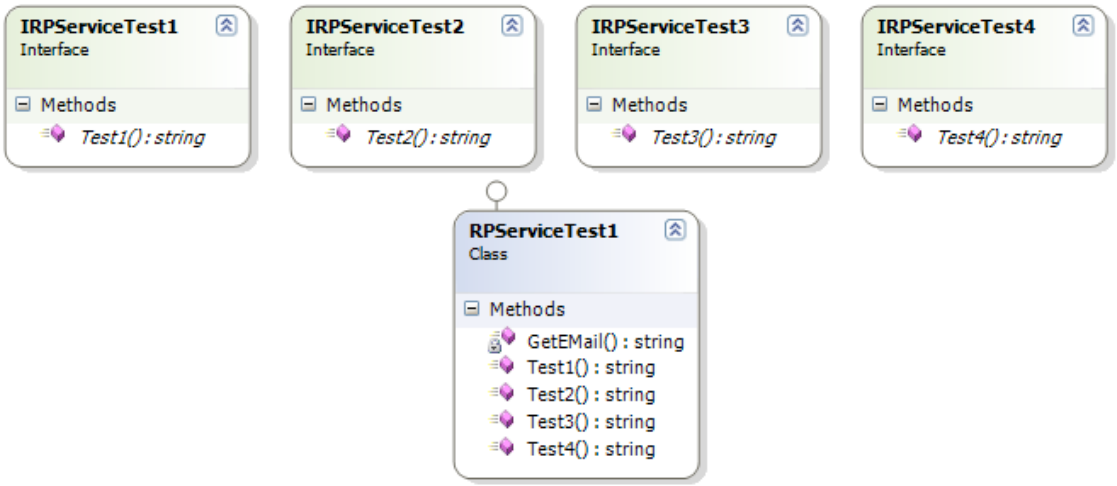

Figura 6.3 – Serviço WCF (*Relying Party*)

Na Figura 6.4 é apresentado o ficheiro de configuração do serviço WCF atrás apresentado. A configuração referente às interfaces `IRPServiceTest3` e `IRPServiceTest4` foi omitida nesta fase e será apresentada na secção ***Authorization Manager*** do capítulo **Extensões à implementação base**.

Na corrente configuração realça-se os seguintes aspectos:

- ***Endpoints***: O serviço expõe um *endpoint* para a interface WCF `IRPServiceTest1`, configurado com o *binding* predefinido `wsHttpBinding` sem algum tipo de segurança. O serviço expõe um segundo *endpoint*, para a interface WCF `IRPServiceTest2`, configurado com o *binding* predefinido `wsFederationHttpBinding`. O *binding* deste segundo *endpoint* tem segurança ao nível da mensagem e requer a recepção de um *security token* SAML, contendo dois *claims* emitidos pelo STS referênciado pelo Uri `https://www.jf_sts.com/STSImplSample/STSImplSample.svc`.
- ***Behaviours***: Na recepção de um pedido em conjunto com um *security token*, a mensagem e o seu conteúdo é validada pela classe `ClaimProcessing`, derivada da classe `ServiceAuthorizationManager`, através do método redefinido `CheckAccess`. Esta definição está configurada no elemento `serviceAuthorization`. Está também configurado no elemento `behaviours` o certificado do STS.

```
<configuration>
 ...
 <system.serviceModel>
   <services>
     <service
       name="RelyingPartyService.RPServiceTest1"
       behaviorConfiguration="RelyingPartyService.RPServiceTest1Behavior" >
       <endpoint address="test1"
         binding="wsHttpBinding"
         bindingConfiguration="IRPServiceTest1Binding"
         contract="RelyingPartyService.IRPServiceTest1" />
       <endpoint address="test2"
         binding="wsFederationHttpBinding"
         bindingConfiguration="IRPServiceTest2Binding"
         contract="RelyingPartyService.IRPServiceTest2" />
       <endpoint address="mex"
         binding="mexHttpBinding"
         contract="IMetadataExchange" />
       <host>
         <baseAddresses>
           <add baseAddress="http://www.jf rp.com/RP/service.svc/" />
         </baseAddresses>
       </host>
     </service>
   </services>
   <bindings>
     <wsHttpBinding>
       <binding name='IRPServiceTest1Binding'>
         <security mode='None' />
       </binding>
     </wsHttpBinding>
     <wsFederationHttpBinding>
       <binding name='IRPServiceTest2Binding'>
         <security mode="Message">
           <message issuedTokenType="http://...#SAMLV1.1" issuedKeyType="SymmetricKey">
             <claimTypeRequirements>
               <add claimType="http://.../claims/name" />
               <add claimType="http://.../claims/emailaddress" />
             </claimTypeRequirements>
               <issuerMetadata
                   address='https://www.jf_sts.com/STSImplSample/STSImplSample.svc/mex'>
                 <identity>
                 <certificateReference
                   findValue="CN=www.jf sts.com"
                   storeLocation="LocalMachine"
                   storeName="OtherPeople"
                   x509FindType="FindBySubjectDistinguishedName"/>
```

```
            </identity>
          </issuerMetadata>
        </message>
      </security>
    </binding>
  </wsFederationHttpBinding>
 </bindings>
 <behaviors>
  <serviceBehaviors>
   <behavior name="RelyingPartyService.RPServiceTest1Behavior">
    <serviceAuthorization serviceAuthorizationManagerType='…ClaimProcessing,…' />
    <serviceMetadata httpGetEnabled="true" />
    <serviceDebug includeExceptionDetailInFaults="true" />
    <serviceCredentials>
     <serviceCertificate
      findValue="CN=www.jf rp.com"
      storeLocation="LocalMachine"
      storeName="My"
      x509FindType="FindBySubjectDistinguishedName" />
     <issuedTokenAuthentication …>
      <knownCertificates>
       <add
        findValue="CN=www.jf_sts.com"
        storeLocation="LocalMachine"
        storeName="OtherPeople"
        x509FindType="FindBySubjectDistinguishedName" />
      </knownCertificates>
     </issuedTokenAuthentication>
    </serviceCredentials>
   </behavior>
  </serviceBehaviors>
 </behaviors>
</system.serviceModel>
</configuration>
```

Figura 6.4 – Configuração do serviço WCF (*Relying Party*)

O *STSMan* também desempenha o papel de *Relying Party* quando o *login* é efectuado usando um *managed information card*. Este processo e os seus aspectos são apresentados na secção ***Information Card Model*** do capítulo seguinte.

6.2 Sujeito (cliente)

Para o papel de sujeito foi implementada a aplicação Windows com a interface com o utilizador apresentada na Figura 6.5. Esta aplicação disponibiliza quatro acções (botões), uma para cada interface/*endpoint* exposta pelo *Relying Party* e apresentada na secção anterior. A terceira e a quarta acção e as suas descrições são apresentadas na secção ***Authorization Manager*** do capítulo seguinte.

A primeira acção invoca a operação WCF da interface `IRPServiceTest1` sem restrições algumas. A segunda acção invoca a operação WCF da interface `IRPServiceTest2`. Esta acção requer a introdução prévia do par *Username*/*Password*, necessário para a autenticação da aplicação cliente perante o STS. Ao espoletar a acção, a aplicação cliente, antes de invocar a operação do RP, requisita ao STS um *security token*. Caso suceda, a aplicação depois invoca a operação em conjunto com o *security token* recebido do STS. Caso o RP valide o *security token* recebido, devolve à aplicação cliente o resultado da operação, neste caso devolve uma *string* contendo a *claim* do tipo e-mail, contida no *security token*.

O ficheiro de configuração da aplicação cliente, na configuração dedicada à tecnologia WCF, reflecte a mesma configuração do RP (Figura 6.4).

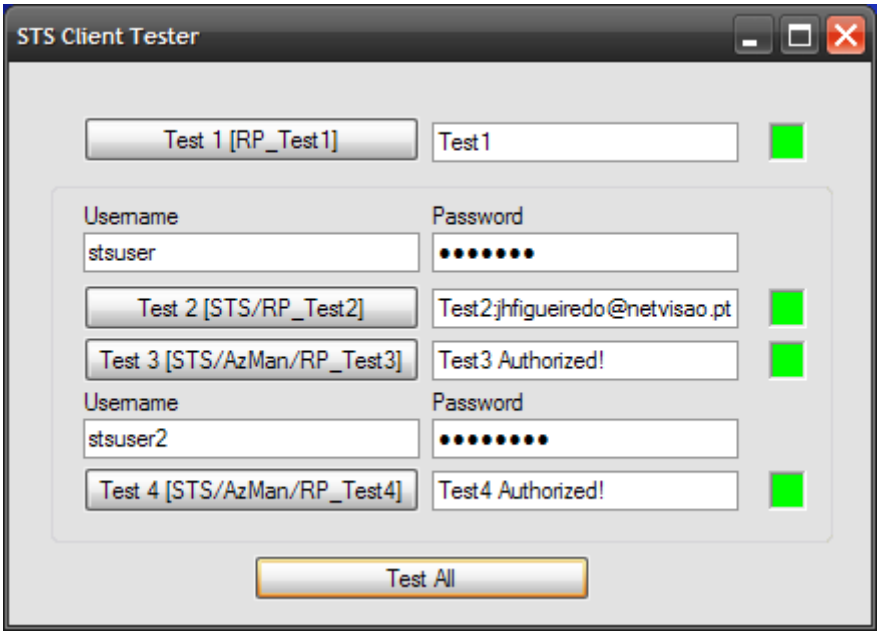

Figura 6.5 – Interface da aplicação cliente (Sujeito) do serviço WCF (*Relying Party*)

6.3 Verificação das diferentes autenticações suportadas

Para suportar as diferentes autenticações especificadas e para que a aplicação cliente possa aceder às operações disponibilizadas pelo *Relying Party*, efectuou-se as configurações WCF referenciadas na secção **Parâmetros WCF** do capítulo anterior e apresentadas na Tabela 6.1. A tabela apresenta, para cada autenticação e para cada participante, os contratos expostos nos *endpoints*, qual o tipo de *binding* escolhido, e caso necessite da participação do STS (wsFederationHttpBinding), qual o *binding* escolhido (*Issuer Binding*) e qual a tipo de autenticação (*Issuer Binding Credential Type*).

Autenticação	Sujeito	*Endpoint contract*	*Binding*	*Issuer Binding*	*Issuer Binding Credential Type*
Username	STS	ISecurityTokenService	wsHttpBinding		UserName
	RP	IRPServiceTest1	wsHttpBinding		
		IRPServiceTest2	wsFederationHttpBinding		
	Cliente	IRPServiceTest1	wsHttpBinding		
		IRPServiceTest2	wsFederationHttpBinding	wsHttpBinding	UserName
X.509	STS	ISecurityTokenService	wsHttpBinding		Certificate
	RP	IRPServiceTest1	wsHttpBinding		
		IRPServiceTest2	wsFederationHttpBinding		
	Cliente	IRPServiceTest1	wsHttpBinding		
		IRPServiceTest2	wsFederationHttpBinding	wsHttpBinding	Certificate
Windows	STS	ISecurityTokenService	wsHttpBinding		Windows
	RP	IRPServiceTest1	wsHttpBinding		
		IRPServiceTest2	wsFederationHttpBinding		
	Cliente	IRPServiceTest1	wsHttpBinding		
		IRPServiceTest2	wsFederationHttpBinding	wsHttpBinding	Windows

Tabela 6.1 – Configurações WCF para as diferentes autenticações suportadas

7 Extensões à implementação base

O presente capítulo apresentada aspectos da implementação do *STSMan* e das extensões não consideradas para a implementação base, nomeadamente as extensões para os modelos ISIP e *OpenID* e a integração com o *Authorization Manager*.

7.1 Aplicação *web* de gestão do STS

Na secção **Politicas de mapeamento na base de dados de gestão do STS** do capítulo **Implementação base** fez-se uma breve apresentação do *STSMan*. Esta aplicação visa dar aos *Policy Administrators* do STS a capacidade de definir online diferentes políticas de emissão, retirando a necessidade da configuração local inerente aos ficheiros de configuração.
Esta aplicação foi implementado sobre a tecnologia ASP.NET e AJAX, e está suportado sobre uma base de dados SQL Server cujo esquema reflecte a base de dados relacional apresentada na Figura 5.19, em conjunto com mais alguns dados adicionais referentes à identificação dos utilizadores do *STSMan*.

7.1.1 Criação de novos utilizadores e login no serviço

A página de entrada (Figura 7.1) contém duas opções de login e uma opção de criação de um novo utilizador. A primeira opção de login consiste na introdução de um nome e de uma palavra-chave e a segunda opção consiste na apresentação de um *managed information card*. Este segundo processo é detalhado na secção ***Information Card Model***.

Figura 7.1 – Página de entrada da aplicação *web* de gestão do STS

7.1.2 Gestão através do *STSMan*

O utilizador do *STSMan*, após o login com sucesso, tem à sua disposição várias áreas de configuração (Figura 7.3):

- ***User Information*** (Figura 7.2): Esta área permite a alteração de dados do utilizador (ex: palavra-chave), a obtenção e o registo de um *managed information card* (detalhado na secção ***Information Card Model***).

Figura 7.2 – A aplicação *web* de gestão do STS e a área para a alteração dos dados do utilizador e a obtenção e registo de um *managed information card*

- ***Relying Parties*** (Figura 7.3): Nesta área o utilizador regista os RPs que gere e que requerem os serviços do STS. Para isso o utilizador terá que introduzir o *endpoint* (URI) do RP e carregar o certificado X509 a ele associado. Este conjunto de pares, *endpoints*/certificados, substitui o conjunto de entradas no elemento `rpTokens` do ficheiro de configuração descrito na secção **Parâmetros Gerais** do capítulo **Implementação base**. Para que o STS use esta informação específica contida na base de dados em vez do ficheiro de configuração, o *Application Assembler* terá que configurar o atributo `db` do elemento `rpTokens` para *true* (Figura 5.33).

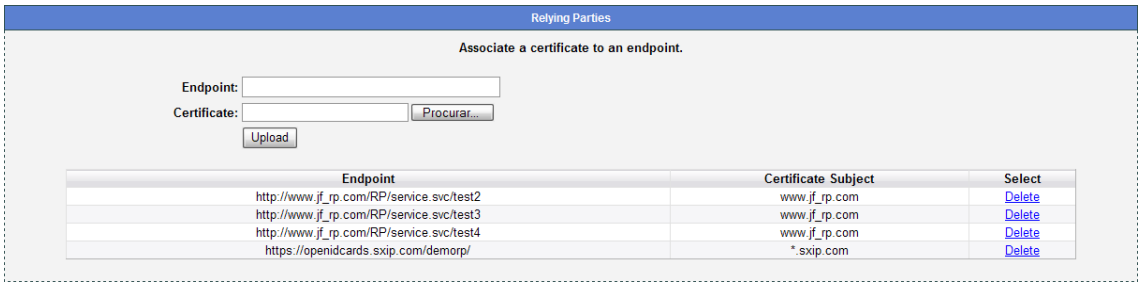

Figura 7.3 – Área para a definição de *Relying Parties*

- ***Claim Mapping*** (Figura 7.4): Alguns detalhes em relação ao conteúdo e à utilização desta área foram apresentados na secção **Politicas de mapeamento na base de dados de gestão do STS** do capítulo **Implementação base**. É relevante acrescentar que cada mapeamento de *claims* configurado é associado a um dos RPs configurados na área *Relying Parties* e para que o STS use esta informação específica contida na base de dados em vez do ficheiro de configuração, o *Application Assembler* terá que configurar o atributo `db` do elemento `claimMappings` para *true*.

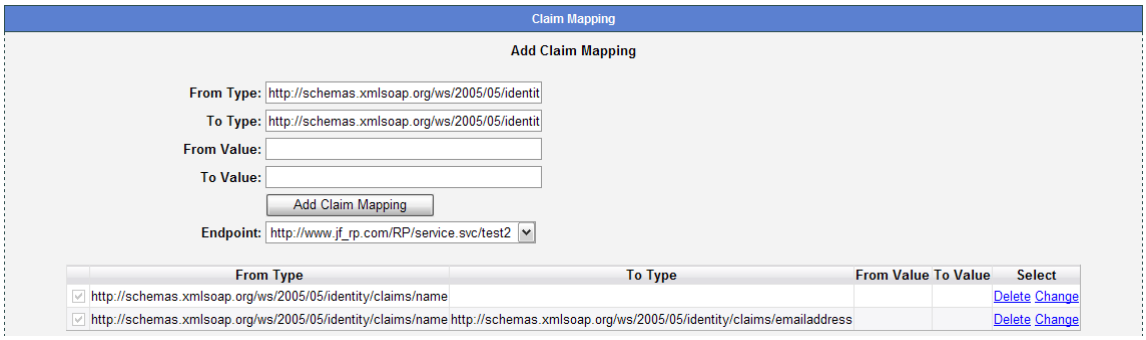

Figura 7.4 – Área para a definição de mapeamentos de *claims*

- ***OpenID*** **(**Figura 7.5): Descrito na secção **OpenID**.

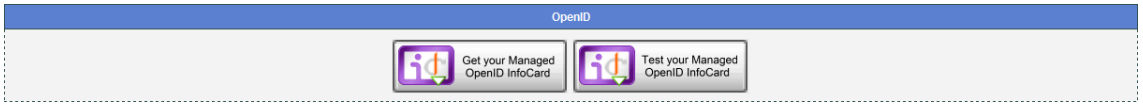

Figura 7.5 – Área para a obtenção e uso de um *OpenID managed information card*

7.2 *Information Card Model*

O modelo *Information Card* (ISIP) foi apresentado na secção ***Information Card Model*** do capítulo **Contexto e Enquadramento** e os vários aspectos desta integração são apresentados nas secções seguintes.

7.2.1 Integração do modelo ISIP na *STSLib*

Para a integração do modelo ISIP numa das possíveis configurações que o STS pode ter, foi necessário acrescentar novos módulos de extensões para o processamento da informação necessária, especificamente os módulos `RSTModuleISIP e IssuedClaimsModuleISIP`.

7.2.1.1 Recepção de um RST com extensões ISIP

Na Figura 7.6 apresenta-se uma amostra de um pedido RST contendo informação específica do modelo ISIP, nomeadamente a informação contida nos elementos `InformationCardReference`, `ClientPseudonym` e `RequestDisplayToken`.

```
<wst:RequestSecurityToken Context="ProcessRequestSecurityToken" xmlns:wst="…/200512">
 <wst:RequestType>http://…/ws-sx/ws-trust/200512/Issue</wst:RequestType>
 <wsid:InformationCardReference xmlns:wsid="http://…/ws/2005/05/identity">
   <wsid:CardId>http://www.jf sts.com/card/randomnnumber123</wsid:CardId>
   <wsid:CardVersion>1</wsid:CardVersion>
 </wsid:InformationCardReference>
 <wst:Claims Dialect="http://schemas.xmlsoap.org/ws/2005/05/identity">
   <wsid:ClaimType Uri="http://…/claims/privatepersonalidentifier" xmlns:wsid="…" />
   <wsid:ClaimType Uri="http://…/claims/givenname" xmlns:wsid="…" />
   <wsid:ClaimType Uri="http://…/claims/emailaddress" xmlns:wsid="…" />
 </wst:Claims>
 <wst:KeyType>http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer</wst:KeyType>
 <wsp2004:AppliesTo xmlns:wsp2004="http://schemas.xmlsoap.org/ws/2004/09/policy">
   <EndpointReference xmlns="http://www.w3.org/2005/08/addressing">
     <Address>https://www.jf sts.com/StsManager/StsLogin.aspx</Address>
     <Identity xmlns="http://…/ws/2006/02/addressingidentity">...</Identity>
   </EndpointReference>
 </wsp2004:AppliesTo>
 <ClientPseudonym xmlns="http://schemas.xmlsoap.org/ws/2005/05/identity">
   <PPID>im/PF6RBmRd2QWF31rgrSS3TrpPRY/2awZXDp4gjnQo=</PPID>
 </ClientPseudonym>
 <wst:TokenType>http://…/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1</wst:TokenType>
 <wsid:RequestDisplayToken xml:lang="pt" xmlns:wsid="http://…/ws/2005/05/identity" />
</wst:RequestSecurityToken>
```

Figura 7.6 – Amostra de um RST com extensões ISIP

Como os elementos da norma ISIP são extensões à norma *WS-Trust*, não foram contemplados no conjunto de propriedades tipificadas da classe RST. Para a extracção desta informação implementou-se o módulo de extensão `RSTModuleISIP`, cujo objectivo é transformar a informação descrita em XML e inclui-la no contentor `Extensions` da classe RST apresentado na secção **Recepção de um RequestSecurityToken**.

O módulo de extensão `RSTModuleISIP` é incluído no primeiro ponto de extensão do *pipeline* de emissão (processamento do RST) e poderá coexistir com os restantes módulos de extensão deste ponto de extensão, mesmo para pedidos que não sejam especificos do modelo ISIP. O módulo só processará a informação ISIP, caso exista.

7.2.1.2 Processamento da *claim* PPID

O *security token* gerado pelo STS, para um pedido referente ao modelo ISIP, terá que conter, entre outros, o identificador único do *information card* usado para espoletar o pedido RST, designado por *private personal identifier* (PPID).

Para cada *information card*, o *Identity Selector* gera uma chave mestre e um Card ID. A chave mestre consiste em 32 bytes de dados aleatórios e o Card ID é um GUID. O PPID é gerado a partir do Card ID em conjunto com algumas propriedades do certificado do RP, assim criando um identificador único para a relação do *information card* com o *Relying Party*. O PPID gerado é normalmente usado pelo RP para indexar um utilizador do seu recurso, armazenado numa base de dados. Para isso, após a autenticação com sucesso do utilizador perante o STS, através do browser e do *Identity Selector*, o STS terá que incluir o PPID recebido no RST, como uma *claim* no *security token* requisitado pelo RP.

Para o objectivo atrás descrito e para completar a integração da *STSLib* com o modelo ISIP, foi implementado o módulo de extensão `IssuedClaimsModuleISIP` (Figura 7.7).

```
public class IssuedClaimsModuleISIP : IIssuedClaimsProcessor
{
  public void GetIssuedClaims(RST rst, Collection<Claim> claims)
  {
    if (rst.Extensions.ContainsKey(Constants.WSIdentity.Elements.PPID))
      claims.Add(new Claim(ClaimTypes.PPID,
          rst.Extensions[Constants.WSIdentity.Elements.PPID], Rights.PossessProperty));
  }
}
```

Figura 7.7 – Módulo de extensão IssuedClaimsModuleISIP

O módulo obtém o PPID armazenado no contentor `Extentions` do objecto RST e adiciona-o como uma *claim* à colecção de *claims* a emitir. O módulo é incluído no terceiro ponto de extensão do *pipeline* de emissão (processamento das *claims*) e poderá coexistir com os restantes módulos de extensão, mesmo para pedidos que não sejam especificos do modelo ISIP. O módulo só processará a informação ISIP caso exista.

7.2.1.3 Configuração do STS

Para a obtenção da metadata do STS através da norma *WS-MetadataExchange*, e por não ser possível garantir a segurança ao nível da mensagem, o *Identity Selector* requer uma ligação segura ao nível do transporte, através de HTTPS. Para isso, houve a necessidade de usar o *binding* `mexHttpsBinding` no *endpoint* `mex`, tal como apresentado na Figura 5.31, e implementar uma instância do STS para o IIS com HTTPS.

7.2.2 Integração do modelo ISIP no *STSMan*

Para a validação da integração do modelo ISIP na *STSLib* optou-se por evoluir o *STSMan* de forma a suportar o modelo e ISIP e em simultânea validar a implementação da *STSLib*. Este processo passa pela geração, registo e uso de *managed information cards* (*InfoCard*)[10].

7.2.2.1 Geração de *managed information cards*

Um *InfoCard* é um cartão digital emitido por uma entidade terceira. As *claims* associadas ao *InfoCard* não são guardadas localmente. Quando se selecciona um *InfoCard,* o *Identity Selector* requisita um *security token*, contendo as *claims* necessárias, ao STS, cujo URI está embebido no *InfoCard*.

Para a inclusão do modelo ISIP no *STSMan* foi necessário implementar primeiro um gerador de *InfoCards*. Para isso foram implementadas as classes `ManagedCardWriter` e `ManagedInformationCard` (Figura 7.8). A classe estática `ManagedCardWriter` disponibiliza o método `CreateCard`, que ao ser invocado devolve um *InfoCard* em formato XML. A classe `ManagedInformationCard` contém um conjunto de propriedades tipificadas

[10] Embora a designação *InfoCard* não especifica se se trata de um *information card self-issued* ou *managed*, por motivos de simplificação e uma melhor leitura considera-se que o termo *InfoCard* refere-se sempre a um *managed information card*.

que representam os elementos constituintes de um *InfoCard* e disponibiliza o método `SerializeAndSign`. Este método recebe o certificado do serviço, serializa as propriedades da classe e assina com a chave privada contida no certificado.

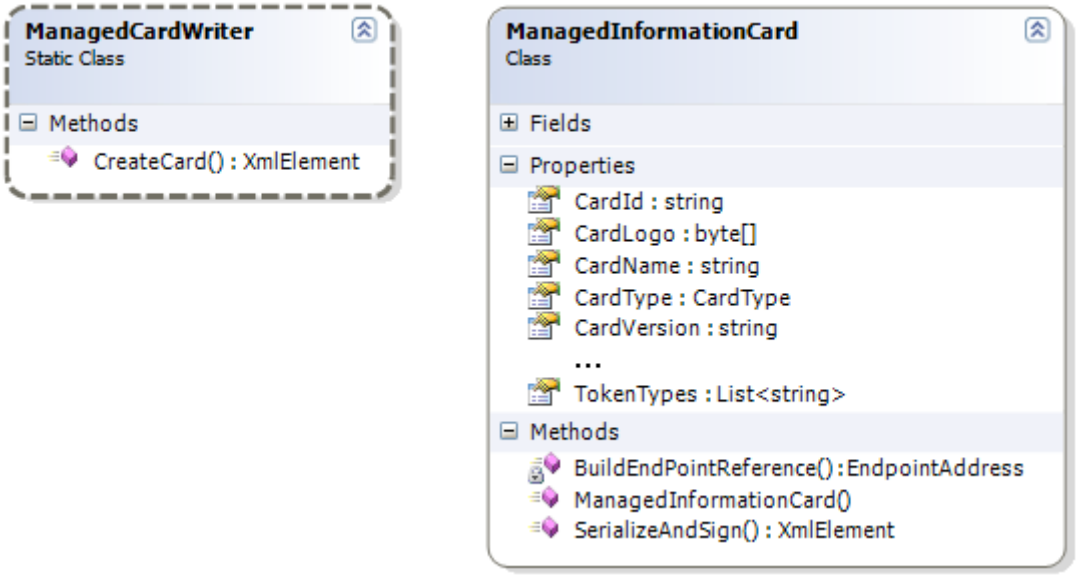

Figura 7.8 – Classes usadas para a geração de um *managed information card*

O formato e o conteúdo do *InfoCard* gerado são configurados e definidos no ficheiro de configuração do *STSMan*. Através da implementação das classes apresentadas na Figura 7.9 definiu-se uma arquitectura de classes representativa da configuração do *InfoCard* contida no ficheiro de configuração.

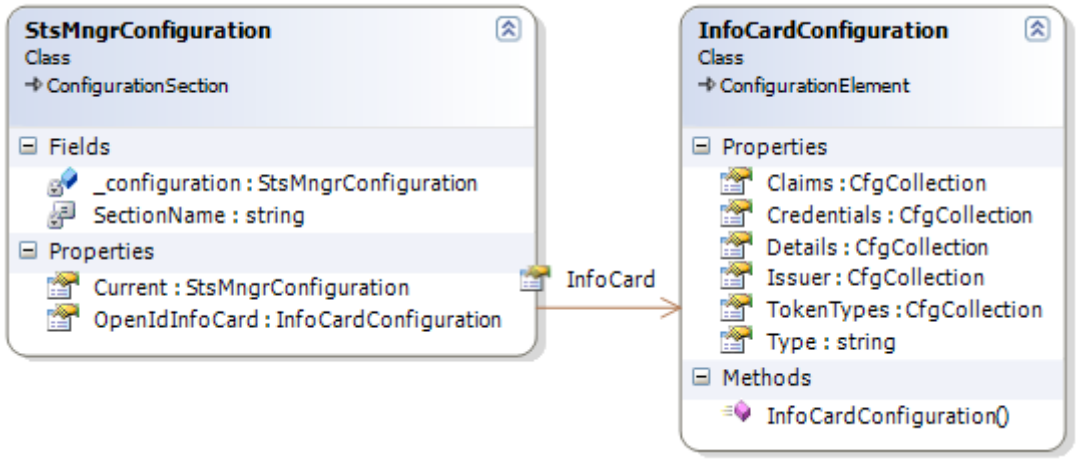

Figura 7.9 – Configuração para a geração de um *managed information card*

Para que o utilizador do *STSMan* obtenha um *InfoCard*, terá primeiro que efectuar o *login* inserindo o nome e a palavra-chave. Após o *login* com sucesso, o utilizador terá à sua

disposição, na área *User Information*, um botão para a obtenção de um *InfoCard* (Figura 7.2). Ao receber o InfoCard o *Identity Selector* questiona o utilizador se o quer guardar.

7.2.2.2 Registo de *managed information cards*

Numa aplicação *web* (RP), a adopção do modelo ISIP obriga a existência de uma relação de confiança entre o RP e um STS. Esta relação leva à não necessidade de manter muita informação acerca dos utilizadores na base de dados do RP, bastando somente um identificador único do utilizador neste RP.

Como foi atrás referenciado, o *Identity Selector* gera um PPID único para cada *InfoCard*. Para que o PPID seja usado na identificação do utilizador no RP, o utilizador terá que primeiro apresentar o *infocard* que deseja registar, após o qual, e com a participação do STS, o RP recebe um *security token* contendo o PPID. Para isso, a página HTML do RP terá que conter um objecto igual ou semelhante ao apresentado na Figura 7.10. Neste exemplo pode-se identificar o URI do STS, o tipo de *security token* e a *claim* necessária. A existência deste código na página, mais um código para a activação do processo de requisição, leva ao aparecimento do *Identity Selector* (Figura 7.11) para a selecção de um *InfoCard*.

```
<object type="application/x-informationcard" name="_xmlToken">
  <param name="issuer" value= "http://www.jf_sts.com/…/STSImplSample.svc/wst13"/>
  <param name="tokenType" value="http://…/oasis-wss-saml-token-profile-1.1#SAMLV1.1" />
  <param name="requiredClaims" value="http://…/claims/privatepersonalidentifier" />
</object>
```

Figura 7.10 – HTML para requisição de um *managed information card*

Após a escolha do *InfoCard* e, caso o *InfoCard* o exija, a introdução de dados de autenticação, o *Identity Selector* requisita um *security token* ao STS, contendo o *claim* com o PPID associado. Caso o pedido ao STS se realize com sucesso, e após a recepção do *security token* do *Identity Selector*, o RP extrai a *claim* PPID e actualiza a base de dados para o utilizador actual.

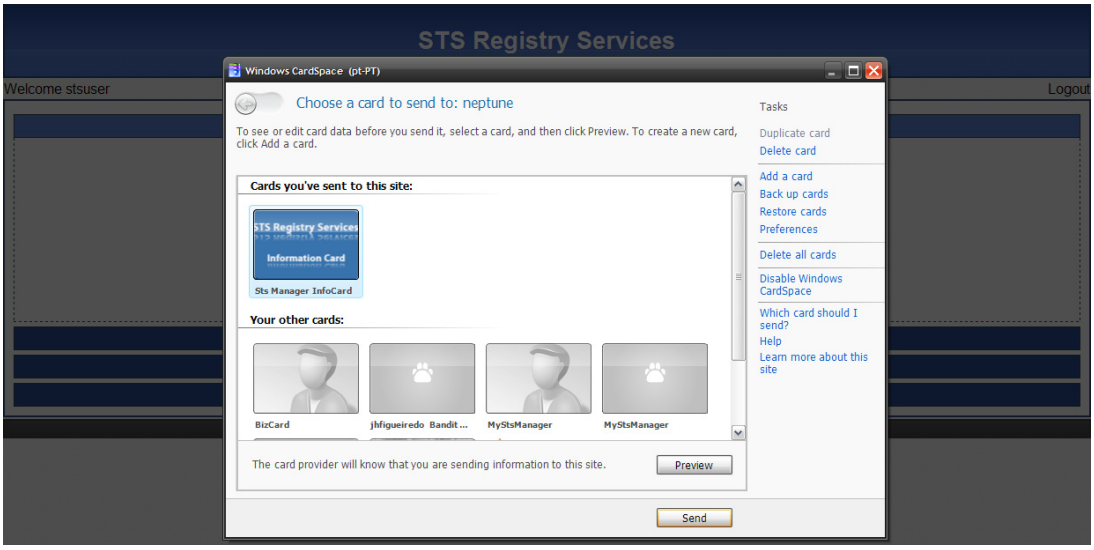

Figura 7.11 – Registo ou uso de um *managed information card*

Este processo de registo de um *InfoCard* foi adicionado ao *STSMan* para que os utilizadores possam usar *InfoCards* no *login* ao serviço. O registo está disponível na área *User Information*, através de um botão para esse efeito (Figura 7.2).

7.2.2.3 Login com *managed information cards*

O processo de *login* numa aplicação *web*, recorrendo a um *InfoCard* previamente obtido e registado, é semelhante ao processo de registo atrás descrito. A diferença reside no tratamento das *claims* contidas no *security token* recebido. A aplicação *web*, em vez de usar um par *username/password* para validar o utilizador e iniciar uma nova sessão, usa o PPID para esse efeito. No *STSMan* disponibilizou-se na página de *login* (Figura 7.1) um botão para a apresentação de um *InfoCard*, com a participação do *Identity Selector* e do STS.

7.3 Authorization Manager

O *Authorization Manager*, usualmente designado por *AzMan*, é uma arquitectura de segurança para a plataforma Windows, baseado no modelo RBAC. O *AzMan* pode ser usado em qualquer aplicação que necessite o controlo de acessos baseado neste modelo.
O *AzMan* é constituído por dois componentes: um *runtime* e uma *interface* de administração. O *runtime* (AZROLES.DLL) expõe um conjunto de interfaces COM usadas pelas aplicações que implementam o modelo RBAC. A *interface* de administração (Figura 7.13) é uma consola MMC para a criação e gestão de políticas de autorização.

7.3.1 Integração do *Authorization Manager* na *STSLib*

Para a integração do *AzMan* na *STSLib* implementou-se o módulo de extensão `IssuedClaimsModuleAzMan` (Figura 7.12), para o terceiro ponto de extensão. Este módulo, através das interfaces COM do *AzMan*, começa por inicializar um `AzAuthorizationStore` (`Microsoft.Interop.Security.AzRoles`) e um `IAzApplication` (`Microsoft.Interop.Security.AzRoles`) com o ficheiro XML contendo as políticas de autorização e nome da aplicação, ambos obtidos do ficheiro de configuração do serviço STS (Figura 5.33).
Na recepção de um pedido RST, e caso este módulo de extensão esteja incluído na configuração do STS, o conjunto de *claims* requeridas é analisado pelo método `GetOperations`. Caso exista *claims* do tipo `authorizationdecision`, as operações associadas são inseridas numa lista e devolvidas para que o módulo de extensão continue com a sua validação, caso não exista, o processamento prossegue para o próximo módulo no *pipeline* de emissão.
Após a recepção da lista de operações, o módulo valida o pedido de acesso através do método `AccessGranted`. Este método recebe a lista de operações e o URI do RP, que serve de *scope* nas políticas definidas no *AzMan*, obtém os identificadores únicos das operações do *AzMan* e valida-as em conjunto com o *scope* e a informação da autenticação do requerente

perante o STS. Caso a autorização seja concedida, é incluida na lista de *claims* a emitir um nova *claim* do tipo `authorizationdecision`, confirmando a autorização. No processo de obtenção dos identificadores únicos das operações, o *AzMan* devolve ao método `AccessGranted` uma lista de pares nome/id de todas as operações configuradas para a aplicação considerada. De seguida, o método filtra a lista obtida considerando apenas as operações recebidas nas *claims*.

```
public class IssuedClaimsModuleAzMan : IIssuedClaimsProcessor
{
  private IAzApplication  azApp;

  public IssuedClaimsModuleAzMan()
  {
    AzAuthorizationStore store = new AzAuthorizationStoreClass();
    store.Initialize(0, "msxml://" + StsConfiguration…("path"), null);
     azApp = store.OpenApplication(StsConfiguration…("appName"), null);
  }

  public void GetIssuedClaims(RST rst, Collection<Claim> claims)
  {
    List<string> claimOps = GetOperations(rst.Claims);
    if (claimOps.Count == 0)
      return;
    if (AccessGranted(claimOps, rst.AppliesTo.Uri.ToString()))
      claims.Add(new Claim(ClaimTypes.AuthorizationDecision, "ok", …));
    else
      throw new FaultException("AzMan: Access was denied!");
  }

  private bool AccessGranted(List<string> claimOps, string scope)
  {
    Dictionary<string, int> opToId =
      new Dictionary<string, int>( azApp.Operations.Count);
    object[] operations = new object[claimOps.Count];

    foreach (IAzOperation op in _azApp.Operations)
      opToId.Add(op.Name, op.OperationID);

    int i = 0, opId;
    foreach (string claimOp in claimOps)
    {
      if (opToId.TryGetValue(claimOp, out opId))
        operations[i++] = (object)opId;
      else
        return false;
    }

    IAzClientContext ctx =
      _azApp.InitializeClientContextFromStringSid(GetSID(), 0, null);

    object[] scopes = { scope };
    object[] results = (object[])ctx.AccessCheck("sts", scopes, operations, …);
    return (int)results[0] == 0;
  }

  private List<string> GetOperations(IList<ClaimTypeRequirement> claims){…}
  private string GetSID(){…}
}
```

Figura 7.12 – Módulo de extensão IssuedClaimsModuleAzMan

7.3.2 Validação da integração do *Authorization Manager* na *STSLib*

Para a validação da integração do *AzMan* na *STSLib*, primeiro criou-se uma nova política de acesso através da *interface* de administração do *AzMan* (Figura 7.13). Nesta configuração criou-se duas operações (*op_test3* e *op_test4*), dois *roles* (*Role3* e *Role4*) e dois *scopes*

(*http://www.jf_rp.com/RP/service.svc/test3* e http*://www.jf_rp.com/RP/service.svc/test4*). De seguida associou-se a primeira operação ao primeiro *role*, e este ao primeiro *scope* e a um utilizador (A) do sistema, e da mesma forma associou-se a segunda operação, *role* e *scope* a outro utilizador (B) do sistema.

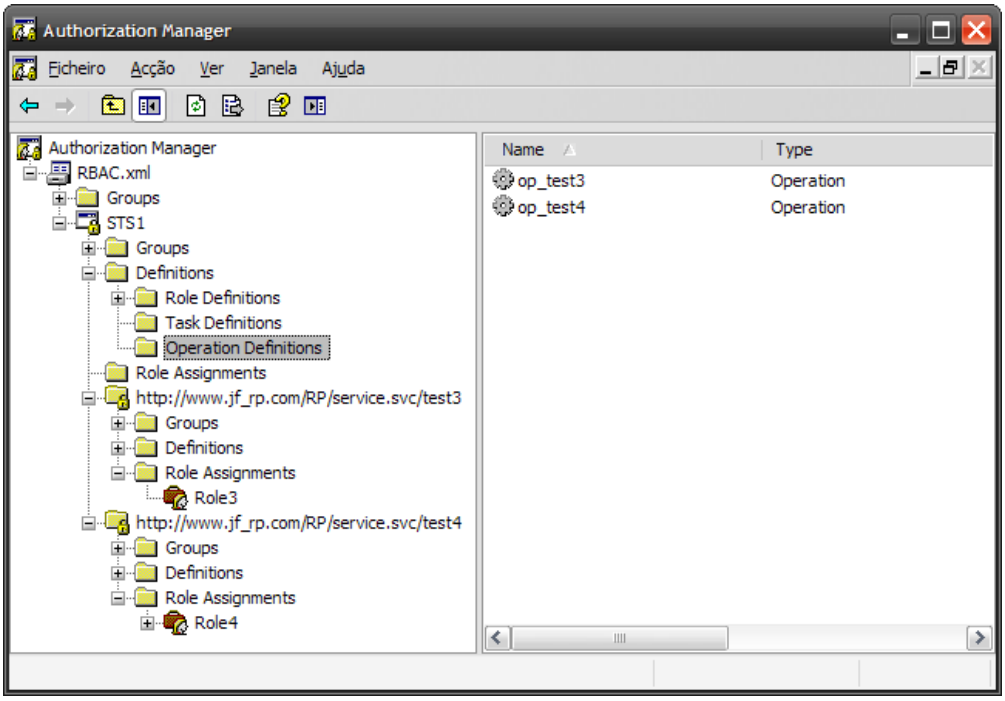

Figura 7.13 – *Interface* de administração do *Authorization Manager*

Desta forma definiu-se que apenas o utilizador A pode aceder à operação *op_test3* e apenas o utilizador B pode aceder à operação *op_test4*.

No *Relying Party* apresentado na secção **Relying Party e Sujeito (cliente)** implementou-se as duas interfaces WCF, `IRPServiceTest3` e `IRPServiceTest4`, com o objectivo de associar as suas operações às operações do *AzMan*, atrás referenciadas. Para isso adicionou-se ao ficheiro de configuração do RP (Figura 6.4) a configuração apresentada na Figura 7.14. Na primeira operação WCF, identificada por `test3`, configurou-se a necessidade da apresentação de um *security token* contendo uma *claim* de autorização para a operação `op_test3` (`claimType="http://…/claims/authorizationdecision#op_test3"`), de forma que apenas o utilizador A possa aceder ao recurso da operação `test3`. Da mesma forma configurou-se a operação `test4` associada ao `op_test4` e ao utilizador B.

```
<configuration>
 ...
 <system.serviceModel>
   <services>
     <service
       name="RelyingPartyService.RPServiceTest1"
       behaviorConfiguration="RelyingPartyService.RPServiceTest1Behavior" >
       ...
       <endpoint
         address="test3"
         binding="wsFederationHttpBinding"
```

```
        bindingConfiguration="IRPServiceTest3Binding"
        contract="RelyingPartyService.IRPServiceTest3" />
      <endpoint
        address="test4"
        binding="wsFederationHttpBinding"
        bindingConfiguration="IRPServiceTest4Binding"
        contract="RelyingPartyService.IRPServiceTest4" />
      ...
    </service>
  </services>
  <bindings>
    ...
    <wsFederationHttpBinding>
      <binding name='IRPServiceTest3Binding'>
        <security mode="Message">
          <message issuedTokenType="http://…/oasis-wss-saml-token-profile-1.1#SAMLV1.1"
              issuedKeyType="SymmetricKey">
            <claimTypeRequirements>
              <add claimType="http://…/claims/authorizationdecision#op test3" />
            </claimTypeRequirements>
            <issuerMetadata address='https://…/mex' >...</issuerMetadata>
          </message>
        </security>
      </binding>
      <binding name='IRPServiceTest4Binding'>
        <security mode="Message">
          <message issuedTokenType="http://…/oasis-wss-saml-token-profile-1.1#SAMLV1.1"
              issuedKeyType="SymmetricKey">
            <claimTypeRequirements>
              <add claimType="http://…/claims/authorizationdecision#op test4" />
            </claimTypeRequirements>
            <issuerMetadata address='https://…/mex' >...</issuerMetadata>
          </message>
        </security>
      </binding>
    </wsFederationHttpBinding>
  </bindings>
  ...
 </system.serviceModel>
</configuration>
```

Figura 7.14 – Configuração do *Relying Party* integrado com o *Authorization Manager*

Para testar as duas operações configuradas, e consequentemente validar a implementação da *STSLib*, implementou-se na aplicação cliente apresentada na secção **Relying Party e Sujeito (cliente)** e na Figura 6.5, duas acções (*Test3* e *Test4*), uma para cada operação atrás referenciada. Para que o utilizador A possa aceder ao recurso da primeira operação (*test3*) terá que primeiro inserir o nome e a palavra-chave associada e depois premir o botão *Test3*. Caso o utilizador A tente aceder ao recurso da segunda operação (*test4*) receberá um mensagem proveniente do STS informando que não tem autorização de acesso.

7.4 OpenID

O modelo *OpenID* foi apresentado na secção ***OpenID*** do capítulo **Contexto e Enquadramento** e os vários aspectos desta integração são apresentados nas secções seguintes.

7.4.1 Integração do modelo *OpenID* na *STSLib*

Para a integração do modelo *OpenID* numa das possíveis configurações que o STS poderá ter, foi necessário acrescentar novos módulos de extensões para o processamento da informação necessária, especificamente os módulos `IssuedClaimsModuleOpenID` e `IssuedTokenModuleOpenID`.

7.4.1.1 Recepção de um RST com extensões ISIP e *OpenID*

Como já foi referido anteriormente, a integração do modelo *OpenID* passa pelo uso do modelo ISIP para a definição de um *OpenID InfoCard* e a requisição de um *OpenID security token* ao STS. Na Figura 7.15 apresenta-se uma amostra de um pedido RST caracterizado pelo pedido de um *security token* do tipo *OpenID*, além da informação proveniente do modelo ISIP.

```
<wst:RequestSecurityToken Context="ProcessRequestSecurityToken"
    xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
  <wst:RequestType>http://…/ws-sx/ws-trust/200512/Issue</wst:RequestType>
  <wsid:InformationCardReference xmlns:wsid="http://…/ws/2005/05/identity">
    <wsid:CardId>http://www.jf sts.com/card/randomnnumber124</wsid:CardId>
    <wsid:CardVersion>2</wsid:CardVersion>
  </wsid:InformationCardReference>
  <wst:Claims Dialect="http://schemas.xmlsoap.org/ws/2005/05/identity">
    <wsid:ClaimType Uri="http://…/claims/identifier" xmlns:wsid="…" />
  </wst:Claims>
  <wst:KeyType>http://docs.oasis-open.org/ws-sx/ws-trust/200512/Bearer</wst:KeyType>
  <wsp2004:AppliesTo xmlns:wsp2004="…">…</wsp2004:AppliesTo>
  <wst:TokenType>http://specs.openid.net/auth/2.0</wst:TokenType>
  <wsid:RequestDisplayToken xml:lang="pt" xmlns:wsid="http://…/ws/2005/05/identity" />
</wst:RequestSecurityToken>
```

Figura 7.15 – Amostra de um RST requisitando um *OpenID security token*

7.4.1.2 Processamento das *claims* para o *OpenID security token*

Para o processamento das *claims* a incluir no *OpenID security token* foi implementado o módulo de extensão `IssuedClaimsModuleOpenID` (Figura 7.16). Este módulo tem o objectivo de gerar as *claims* a emitir, cujo conteúdo são pares chave/valor definidos no protocolo *OpenID*. O módulo começa por gerar uma chave simétrica e um identificador aleatório, persistindo-os para posterior uso na validação do *OpenID*, pelo *OpenID Provider*.
Após a geração da maioria dos pares do protocolo *OpenID* é gerado a assinatura MAC da informação contida, com a chave simétrica, e é inserido como a última *claim* do conjunto.

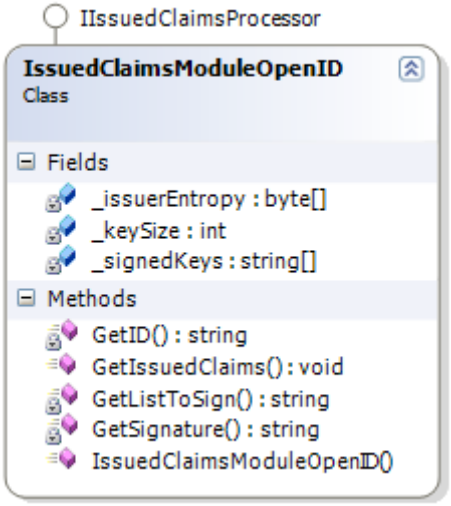

Figura 7.16 – Módulo de extensão IssuedClaimsModuleOpenID

7.4.2 Geração do *security token OpenID*

Para a geração do *OpenID security token* foi implementado o módulo de extensão `IssuedTokenModuleOpenID`. Este módulo recebe as *claims* geradas anteriormente e instancia uma classe do tipo `GenericXmlSecurityToken` (`System.IdentityModel.Tokens`) com as *claims* e outros dados necessários. Esta instância é de seguida inserida no objecto RSTR para finalmente gerar a mensagem contendo uma resposta RSTR. Na Figura 7.17 apresenta-se uma amostra de uma resposta RSTR, onde se evidência o *OpenID security token* (`OpenIDToken`) contendo um conjunto de *claims* (pares chave/valor).

```
<wst:RequestSecurityToken Context="ProcessRequestSecurityToken"
    xmlns:wst="http://docs.oasis-open.org/ws-sx/ws-trust/200512">
  <wst:TokenType>http://specs.openid.net/auth/2.0</wst:TokenType>
  <wst:RequestedSecurityToken>
    <openid:OpenIDToken xmlns:openid="http://specs.openid.net/auth/2.0">
     openid.ns:http://specs.openid.net/auth/2.0
     openid.mode:id_res
     openid.op_endpoint:http://www.jf_sts.com/OpenIdProvider/
     openid.claimed_id:http://www.jf_sts.com/19070/
     openid.identity:http://www.jf_sts.com/19070/
     openid.return_to:https://www.jf_sts.com/StsManager/StsManager.aspx
     openid.response_nonce:2008-07-15T02:01:44Z
     openid.assoc_handle:SAKJH3IOIQEIUOIIoijjsafoiU9ujcikjaasdlDSkj9=
     openid.signed:op_endpoint,claimed_id,identity,return_to,response_nonce,assoc_handle
     openid.sig:PZNucb3/5KnEHsOXEMFkg1FJAnGD+UbGR1LqsscVvEc=
    </openid:OpenIDToken>
  </wst:RequestedSecurityToken>
  <wst:RequestedAttachedReference>…</wst:RequestedAttachedReference>
  <wst:RequestedUnattachedReference>…</wst:RequestedUnattachedReference>
  <wsp:AppliesTo xmlns:wsp="http://…/ws/2004/09/policy">…</wsp:AppliesTo>
</wst:RequestSecurityTokenResponse>
```

Figura 7.17 – Amostra de um RSTR com um *security token OpenID*

7.4.3 Validação do modelo *OpenID* na *STSLib*

Para a validação da integração do modelo *OpenID* na *STSLib* adicionou-se ao *STSMan* a possibilidade de obtenção de um *OpenID InfoCard* e de requisitar ao STS um *OpenID security token*, tal como apresentado na Figura 7.5. A configuração do conteúdo e formato do *OpenID InfoCard* a gerar é definido da mesma forma que a apresentada na secção **Geração de um *managed information card***, havendo uma área reservada para a configuração de *OpenID InfoCards* no ficheiro de configuração do *STSMan*.

Para validar a requisição de um *OpenID security token* através do *OpenID InfoCard* recebido, inseriu-se no *STSMan* um objecto semelhante ao apresentado na Figura 7.10, especifico para o pedido de *OpenID security tokens*. Após o premir do segundo botão apresentado (Figura 7.5), e a escolha de o *OpenID InfoCard* anteriormente armazenado no *Identity Selector*, é espoletado um pedido RST ao STS. Caso o pedido do *OpenID security token* ao STS se realize com sucesso, o seu conteúdo é apresentado ao utilizador.

8 Conclusões e comentários finais

Apresentaram-se neste documento aspectos de desenho e implementação duma biblioteca para a realização de *Security Token Services* (*STSLib*).

O contexto e o enquadramento deste trabalho foram apresentados no capítulo 2, onde se introduzem e descrevem os conceitos, modelos e normas associados à gestão de identidade e ao controlo de acessos em *web services*. Destes destacam-se a norma *WS-Trust*, fundamental para a caracterização do conceito de *Security Token Service*, e os sistemas e modelos com os quais se queria integrar a biblioteca, como por exemplo o *Authorization Manager* e os modelos ISIP e *OpenID*. Além da análise da norma *WS-Trust*, foram consideradas outras normas também necessárias no desenho de STS, como por exemplo as normas *WS-Security* e *WS-Federation*. Por fim, descreveram-se trabalhos relacionados com o tema em estudo, analisando as opções tomadas e quais as características a considerar na nossa implementação.

A motivação para a realização desta biblioteca resultou da análise deste conjunto de modelos e normas para a identidade e o controlo de acessos em aplicações distribuídas. Nesta análise identificou-se o conceito de STS como elemento fundamental para a interoperabilidade e compatibilidade dos actuais modelos de segurança, e identificou-se os diferentes papéis que um STS desempenha, nomeadamente o de *Identity Provider*, de *Policy Decision Point* e o de elo de ligação numa cadeia de confiança.

No capítulo 4 apresentou-se uma arquitectura flexível e extensível. A arquitectura flexível visa permitir várias concretizações do STS, desempenhando diferentes papéis em diferentes contextos e aplicações. A arquitectura extensível visa a adopção de novas implementações e novas versões das normas. Esta flexibilidade e extensibilidade resulta da arquitectura ser constituído por um núcleo, que implementa um *pipeline* de emissão de *security tokens*, e um conjunto de módulos de extensão, que implementa os procedimentos necessários para a emissão.

No capítulo 5 descreveram-se aspectos de implementação da biblioteca. A implementação teve que considerar as características da plataforma WCF para a concretização da arquitectura proposta, em conjunto com as especificações das normas consideradas. Na apresentação da implementação descreveram-se os aspectos essenciais que ligam a plataforma WCF à arquitectura da biblioteca, e quais as opções tomadas para a obtenção de uma biblioteca flexível e extensível. Para a concretização destes objectivos apresentou-se a arquitectura implementada para inclusão de módulos de extensão e alguns dos módulos implementados, necessários para a realização de *Security Token Services* como especificado na norma *WS-Trust*.

Além da implementação de uma biblioteca para a realização de *Security Token Services* como especificado na norma *WS-Trust*, implementou-se um conjunto de extensões à norma, nomeadamente as extensões para os modelos ISIP e *OpenID* e a integração com o

Authorization Manager. Estas extensões foram apresentadas no capítulo 6 em conjunto com os aspectos da sua implementação para integração com a *framework* base.

Verificou-se a correcta implementação das normas relevantes através da criação de um conjunto de cenários de teste. Estes cenários, além de validar a correcta implementação, visaram também servir de demonstradores dos vários papeis propostos para os STS. Estes cenários foram realizados através da implementação de duas plataformas de testes. A primeira plataforma valida e demonstra os diferentes papéis que o STS desempenha no acesso a recursos restritos de um *web service*. A segunda plataforma valida e demonstra a participação do STS no controlo de acesso a uma aplicação *web*, através de um *browser* e um *Identity Selector* (*Cardspace*). Os cenários de teste implementados abrangem todos os papéis considerados para o STS e validam todos os módulos integrados na biblioteca.

8.1 Trabalho futuro

Para o trabalho futuro considerou-se o estudo de novas abordagens, especificamente:

- O uso de uma arquitectura semelhante à usada com o IHttpModule [42], onde é definida uma pilha de módulos, para definição e integração de módulos de extensão.
- A utilização de linguagens lógicas, como o Prolog, para a definição de políticas de emissão.
- A definição de um processo alternativo aos ficheiros de configuração para a configuração e inclusão dos módulos de extensão.

Apêndice A - *Windows Communications Foundations*

A plataforma *Windows Communications Foundations* (WCF) foi desenvolvida pela Microsoft para a implementação de aplicações distribuídas orientadas aos serviços. Esta plataforma é a nova escala na evolução que se iniciou com COM, COM+ e MSMQ e continuou com .Net *Remoting*, ASMX, *System Messaging* e .Net *Enterprise Services* ([43],[44],[45],[46],[47],[48]). A plataforma WCF unifica estas tecnologias num modelo de programação para o desenvolvimento de serviço com qualquer linguagem .Net.

Com esta plataforma é possível comunicar através de *Web Services*, possibilitando assim a interoperabilidade com outras plataformas que utilizam SOAP, como Java EE. A plataforma também suporta muitas das especificações definidas nas normas WS-* para a obtenção de comunicações seguras com qualquer plataforma que também as suportem.

A.1 Características

Cada serviço WCF possui três componentes principais (Figura A.1):

- Uma classe que implementa um serviço disponibilizando um ou mais operações.
- Um processo de alojamento onde o serviço é executado.
- Um ou mais *endpoints* que dão aos clientes acesso ao serviço. Todas as comunicações com o serviço WCF ocorrem através destes *endpoints*.

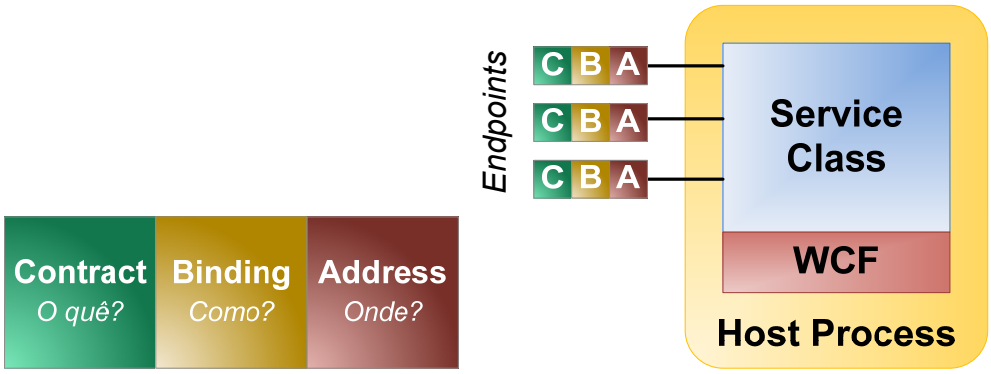

Figura A.1 – Serviço WCF[11]

A.1.1 *Endpoints*

Os serviços implementados através de WCF expõem *endpoints* que clientes e serviços usam para a troca de mensagens. Cada *endpoint* consiste no seguinte:

- Um endereço, definido por um URL, que indica onde o serviço está localizado.

[11] Adaptado de: WCF 1 Hour Overview.ppt, Payam Shodjai

- Um *binding* que define como o cliente deve comunicar com o serviço, definindo aspectos como o protocolo de transporte, a segurança e o tipo de codificação.
- Uma referência para o contrato do serviço WCF exposto no *endpoint*.

A.1.2 *Bindings*

Um *binding* especifica como o cliente se liga ao *endpoint* do serviço WCF, para isso cada *endpoint* terá que ter um *binding* associado. A plataforma WCF já tem incorporado um conjunto de *bindings* predefinidos que cobre a maior parte dos cenários, no entanto, também é possível criar novos *bindings* customizados.
Um *binding* contém as seguintes categorias de informação:

- Informação sobre o protocolo. Como por exemplo os mecanismos de segurança a usar e as parametrizações para as transacções.
- Informação sobre o protocolo de transporte, como TCP, HTTP e *Named Pipes*.
- Informação sobre a codificação de mensagens, como texto ou XML, binário ou *Message Transmission Optimization Mechanism* (MTOM).

Os bindings contêm uma colecção de *binding elements*, como por exemplo o *SecurityBindingElement* e o *TcpTransportElement*, que definem uma pilha de comunicação. A operação da *stack* depende da ordem com que os *binding elements* são inseridos na colecção.
A plataforma WCF disponibiliza um conjunto de combinações predefinidas de *binding elements*, apropriadas para diversos cenários comuns. Por exemplo, o *BasicHttpBinding* visa comunicações *Web Service* básicas, e o *WSHttpBinding* adiciona ao anterior suporte para as normas *WS-**, o *NetMsmqBinding* usa filas de mensagens na comunicação entre aplicações WCF e o *MsmqIntegrationBinding* integra aplicações WCF e MSMQ.
Todos os tipos de *bindings elements*, predefinidos e customizados, podem ser especificadas em ficheiros de configuração ou no código.

A.1.3 Contratos WCF

Os contratos WCF definem as interfaces dos serviços WCF. São implementados programaticamente no serviço, e são expostos aos clientes através de *metadata*. A plataforma WCF disponibiliza cinco tipos de contratos que apresentamos de seguida.

A.1.3.1 Contrato de serviço

Um contrato de serviço define quais as interfaces que o serviço suporta, e são mapeadas em elementos *portType* na linguagem WSDL. Os Contratos de serviço são implementadas como interfaces .Net e são descritas com o atributo `ServiceContract`.

```
[ServiceContract]
public interface IMyContract { ... }
```

Figura A.2 – Contrato de serviço

A.1.3.2 Contrato de operação

Os contratos de operações definem quais as operações que o serviço suporta, e são mapeadas para operações na linguagem WSDL. As operações são definidas adicionando métodos a uma interface *Service Contract* e são descritas com o atributo `OperationContract`.

```
[OperationContract]
void SomeOperation();
```

Figura A.3 – Contrato de operação

A.1.3.3 Contrato de dados

Um contrato de dados define de que forma tipos complexos são serializados quando são usados nas operações dos serviços WCF. São definidos aplicando os atributos `DataContract` e `DataMember` às classes.

```
[DataContract]
public class SomeType{ [DataMember] public int ID;}
```

Figura A.4 – Contrato de dados

A.1.3.4 Contrato de mensagem

Um contrato de mensagem visa controlar explicitamente o cabeçalho e corpo da mensagem SOAP, descrevendo o seu formato. Este contrato disponibiliza uma forma para a adição de cabeçalhos SOAP customizados a mensagens.

```
[MessageContract]
public class MyRequest {
    [MessageHeader] public string field1;
    [MessageBody] public string field2;
}
```

Figura A.5 – Contrato de mensagem

A.1.3.5 Contratos de falha

Estes contratos mapeiam as excepções geradas no código WCF para falhas SOAP. O tipo especificado no *FaultContract* não terá que ser obrigatoriamente uma excepção embora este caso seja o mais comum.

```
[OperationContract]
[FaultContract(typeof(DivideByZeroException))]
void SomeMethod();
```

Figura A.6 – Contrato de falha

A.2 *Service Model* e *Channel Stack Layer*

A arquitectura da plataforma WCF é constituída por duas camadas: *Service Model Layer* e *Channel Stack Layer*. Na camada *Service Model Layer* define-se os endereços, *bindings* e os contratos do serviço WCF, e cujos comportamentos são modificáveis através de *behaviours*. A camada *Channel Stack Layer* define os protocolos e as codificações usadas para o transporte da mensagem.

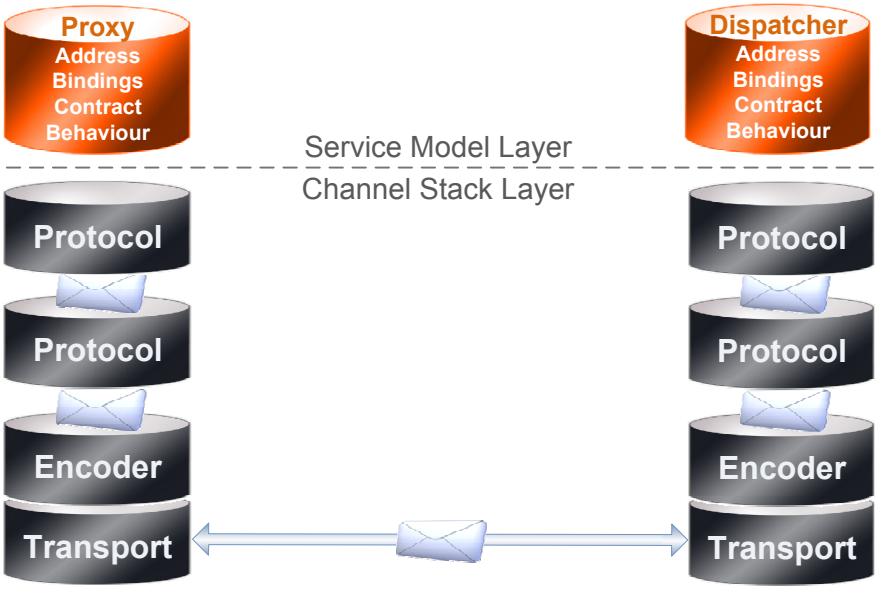

Figura A.7 – *Service Model* e *Channel Stack Layer*[12]

A.3 Segurança

A plataforma WCF fornece um conjunto extenso de funcionalidade de segurança em três áreas:

- **Transferência**. A transferência suporta dois mecanismos para a protecção de mensagens, o modo transporte e o modo mensagem. O modo transporte usa os mecanismos do protocolo de transporte subjacente, como *Secure Socket Layer* (SSL), para a protecção do canal. O modo mensagem usa *WS-Security* e outras especificações para cifrar e assinar as mensagens. O modo transporte é mais eficiente em comunicações ponto a ponto, enquanto o modo mensagem apresenta como vantagem a protecção de mensagens através de nós intermediários. A plataforma WCF também suporta um modo misto, onde a integridade e a confidencialidade é garantida pelo protocolo de transporte, enquanto a autenticação é suportado através de credenciais inseridas na mensagem.

[12] Adaptado da apresentação: INT09 Arquitectura da plataforma WCF, António Cruz, Pedro Félix

- **Autenticação** e **autorização** de clientes. A autenticação e a autorização são baseadas em credenciais, como certificados X.509, nome e palavra-chave, Kerberos e *tokens SAML*.
- **Auditação**. Por omissão, os eventos de segurança do WCF são escritos para o registo de eventos do Windows, para que os administradores possam ver quem acedeu a um determinado serviço.

A.4 Alojamento

Os serviços WCF podem ser alojados nos seguintes tipos de aplicações:

- **IIS**. Caso os serviços WCF usem HTTP como meio de transporte, poderá alojá-los no *Internet Information Service* (IIS), tirando partido das suas funcionalidades, como a gestão do tempo de vida, reiniciação automática após reconfigurações e a ausência de código de alojamento na aplicação. Para isso, basta criar um ficheiro .svc contendo uma referência para o serviço, um ficheiro de configuração com a configuração do serviço, e guarda-los num directório virtual do IIS.
- **WAS**. O *Windows Activation Service* (WAS) é o novo mecanismo de activação de processos do também novo IIS 7.0. Tal como no IIS 6.0, o IIS 7.0 usa, por omissão, o WAS sobre HTTP, no entanto, também poderá usar o WAS para o envio e recepção de mensagens sobre outros protocolos, como TCP e *Named Pipes*.
- **Alojamento na aplicação**. Os serviços WCF podem ser alojados dentro de quaisquer aplicações *managed*, como aplicações consola ou *Windows Forms*.
- ***Managed Windows Service***. Um serviço WCF pode ser registado como um *Windows Service*, ficando assim sobre o controlo do *Windows Service Control Manager* (SCM). Isto é conveniente para serviços WCF de longa duração alojados fora do IIS. O serviço tira proveito das funcionalidades dos *Windows Services*, como o auto arranque e controlo do SCM.

A.5 Configuração de serviços

Os serviços WCF são configurados programaticamente ou através de ficheiros de configuração. Caso o serviço esteja alojado no IIS, usa-se um ficheiro *web.config*, e para os restantes casos usa-se um ficheiro de configuração da aplicação.

Para definir um *endpoint* em código, o objecto `ServiceHost` disponibiliza o método `AddEndpoint`, através do qual é definido o endereço, o *binding*, e o contrato de serviço.

```
Uri echoUri = new Uri("http://localhost:8000/");
WSHttpBinding binding = new WSHttpBinding();
serviceHost.AddEndpoint(typeof(IMyService), binding, echoUri);
```

Figura A.8 – Configuração programática de um serviço WCF

Para definir um *endpoint* num ficheiro de configuração, define-se um elemento `<endpoint>` dentro dos elementos `<sytem.ServiceModel><services><service>` do ficheiro de configuração.

```
<endpoint address="http://localhost/MySrv" binding="wsHttpBinding" contract="IMySrv"/>
```

Figura A.9 – Configuração de um serviço WCF através de um ficheiro de configuração

A.6 Behaviors

A tecnologia WCF definiu o tipo *behavior*, que visa modificar o comportamento dos serviços. Os *behaviors* são aplicáveis aos canais, serviços, *endpoints*, contratos ou operações, e podem ser implementados directamente no código ou através de ficheiros de configuração.

Glossário

ADFS	Active Directory Federation Services
AzMan	Authorization Manager
COM	Component Object Modeling
FS	Federation Service
FS-P	Federation Service Proxy
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol over Secure Socket Layer
IIS	Internet Information Services
IP	Identity Provider
ISIP	Identity Selector Interop Profile
LDAP	Lightweight Directory Access Protocol
MMC	Microsoft Management Console
OASIS	Organization for the Advancement of Structured Information Standards
PDP	Policy Decision Point
PPID	Private Personal Identifier
RBAC	Role Base Access Control
RP	Relying Party
RST	Request Security Token
RSTR	Request Security Token Response
SAML	Security Assertion Markup Language
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer
STS	Security Token Service
STSLib	Security Token Service software library
STSMan	STS Web Manager
WCF	Windows Communication Foundation

Referências

[1] WS-* specifications: http://msdn.microsoft.com/en-us/library/ms951274.aspx

[2] Web Services Trust Language (WS-Trust), February 2005, http://specs.xmlsoap.org/ws/2005/02/trust/WS-Trust.pdf

[3] OASIS Web Services Trust, 1.3, March 2007, http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.pdf

[4] Security in a Web Services World: A Proposed Architecture and Roadmap, IBM Corporation and Microsoft Corporation, April 2002, http://msdn.microsoft.com/en-us/library/ms977312.aspx

[5] W3C Simple Object Access Protocol (SOAP) 1.2, April 2007, http://www.w3.org/TR/soap12-part1

[6] OASIS Web Services Security: SOAP Message Security 1.1, February 2006, http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[7] OASIS Web Services Security Kerberos Token Profile 1.1, February 2006, http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-KerberosTokenProfile.pdf

[8] OASIS Web Services Security SAML Token Profile 1.1, February 2006, http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-SAMLTokenProfile.pdf

[9] OASIS Web Services Security X.509 Certificate Token Profile 1.1, February 2006, http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-os-x509TokenProfile.pdf

[10] C. Neuman, USC-ISI, T. Yu, S. Hartman, K. Raeburn, MIT, RFC4120 - The Kerberos Network Authentication Service (V5), July 2005, http://www.ietf.org/rfc/rfc4120.txt

[11] Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1, 2 September 2003, http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf

[12] Bindings and Profiles for the OASIS Security Assertion Markup Language (SAML) V1.1, 2 September 2003, http://www.oasis-open.org/committees/download.php/3405/oasis-sstc-saml-bindings-1.1.pdf

[13] Conformance Program Specification for the OASIS Security Assertion Markup Language (SAML) V1.1, 2 September 2003, http://www.oasis-open.org/committees/download.php/3402/oasis-sstc-saml-conform-1.1.pdf

[14] The Directory: Public-key and attribute certificate frameworks, ITU-T Recommendation X.509, 31 March 2000

[15] The Directory: Overview of concepts, models and services, ITU-T Recommendation X.500, 9 August 1997

[16] W3C Web Services Policy 1.5, September 2007, http://www.w3.org/TR/ws-policy

[17] OASIS Web Services Security Policy, 1.2, July 2007, http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html

[18] OASIS Web Services Federation, 1.2, September 2007, www.oasis-open.org/committees/wsfed

[19] Passport: http://www.microsoft.com/presspass/features/2000/05-17passport.mspx

[20] Kim Cameron's Identity Weblog: http://www.identityblog.com

[21] Kim Kameron, The Laws of Identity, December 2005, http://www.identityblog.com/stories/2005/05/13/TheLawsOfIdentity.pdf

[22] Microsoft's Vision for an Identity Metasystem, Microsoft, May 2005, http://msdn.microsoft.com/en-us/library/ms996422.aspx

[23] Web Services Metadata Exchange, 1.1, August 2006, http://specs.xmlsoap.org/ws/2004/09/mex/WS-MetadataExchange.pdf

[24] Identity Selector Interoperability Profile 1.0, Microsoft, April 2007, http://download.microsoft.com/download/1/1/a/11ac6505-e4c0-4e05-987c-6f1d31855cd2/Identity-Selector-Interop-Profile-v1.pdf

[25] An Implementer's Guide to the Identity Selector Interoperability Profile 1.0, Microsoft, April 2007, http://download.microsoft.com/download/1/1/a/11ac6505-e4c0-4e05-987c-6f1d31855cd2/Identity-Selector-Interop-Profile-v1-Guide.pdf

[26] A Guide to Using the Identity Selector Interoperability Profile V1.0 within Web Applications and Browsers, Microsoft, April 2007, http://download.microsoft.com/download/1/1/a/11ac6505-e4c0-4e05-987c-6f1d31855cd2/Identity-Selector-Interop-Profile-v1-Web-Guide.pdf

[27] OpenID Authentication 2.0, December 2007, http://openid.net/specs/openid-authentication-2_0.html

[28] Marco Slot , Beginner's guide to OpenID phishing http://marcoslot.net/apps/openid

[29] D. Hardt, J. Bufu, Sxip Identity, OpenID Information Cards 1.0, August 2007, https://openidcards.sxip.com/spec/openid-infocards.html

[30] Dave McPherson, Microsoft Corporation, Role-Based Access Control for Multi-tier Applications Using Authorization Manager, Microsoft, July 2004, http://technet.microsoft.com/en-us/library/cc780256.aspx

[31] RSA Cryptography Standard, PKCS #1 v2.1, RSA Laboratories, June 2002, ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf

[32] WCF Federation Sample:
http://msdn2.microsoft.com/en-us/library/aa355045(VS.90).aspx

[33] Higgins Project:
http://www.eclipse.org/higgins

[34] Sandhu, Ferraiolo, Kuhn, Gavrila, Proposed NIST Standard for Role-Based Access Control, 2001,
http://xml.coverpages.org/NIST-RBAC-ACM2001.pdf

[35] J2EE Roles:
http://java.sun.com/blueprints/guidelines/designing_enterprise_applications/platform_technologies/platform_roles/index.html

[36] W3C Web Services Addressing, May 2006,
http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/

[37] SharpSTS:
https://sharpsts.com

[38] Ataque do tipo *Man in the Middle*:
http://en.wikipedia.org/wiki/Man-in-the-middle_attack

[39] Vittorio Bertocci's Blog – On Proof Tokens:
http://blogs.msdn.com/vbertocci/archive/2008/01/02/on-prooftokens.aspx

[40] Configuração para o *Proof Token* Assimétrico:
http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=2562145&SiteID=1

[41] A classe RsaSecurityTokenParameters:
http://msdn.microsoft.com/en-us/library/system.servicemodel.security.tokens.rsasecuritytokenparameters.aspx

[42] IHttpModule:
http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule.aspx

[43] Component Object Model (COM):
http://msdn.microsoft.com/en-us/library/ms809980.aspx

[44] Component Services (COM+):
http://msdn.microsoft.com/en-us/library/ms685978(VS.85).aspx

[45] Message Queuing (MSMQ):
http://msdn.microsoft.com/en-us/library/ms711472.aspx

[46] .Net Remoting:
http://msdn.microsoft.com/en-us/library/kwdt6w2k(VS.71).aspx

[47] System Messaging:
http://msdn.microsoft.com/en-us/library/system.messaging.aspx

[48] .Net Enterprise Services:
http://msdn.microsoft.com/en-us/library/ms973484.aspx