Gerenciamento Baseado em Modelos da Configuração de Sistemas de Segurança em Ambientes de Redes Complexos

João Porto de Albuquerque Pereira

Tese de Doutorado

Instituto de Computação
Universidade Estadual de Campinas

Gerenciamento Baseado em Modelos da Configuração de Sistemas de Segurança em Ambientes de Redes Complexos

João Porto de Albuquerque Pereira[1]

5 de julho de 2006

Banca Examinadora:

- Prof. Dr. Paulo Lício de Geus
  IC-UNICAMP (Orientador)
- Prof. Dr. Joni da Silva Fraga
  DAS-UFSC
- Prof. Dr. Edgard Jamhour
  PPGIA-PUC-PR
- Prof. Dr. Ricardo Dahab
  IC-UNICAMP
- Prof. Dr. Jacques Wainer
  IC-UNICAMP
- Prof. Dr. Otto Carlos Muniz Bandeira Duarte (Suplente)
  COPPE-UFRJ
- Prof. Dr. Edmundo Roberto Mauro Madeira (Suplente)
  IC-UNICAMP

[1] **Suporte financeiro parcial concedido por CAPES (Coordenação para Aperfeiçoamento dos Programas de Ensino Superior) e DAAD (Serviço Alemão de Intercâmbio Acadêmico).**

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Miriam Cristina Alves – CRB8a / 859

Pereira, João Porto de Albuquerque.

P414g Gerenciamento baseado em modelos da configuração de sistemas de segurança em ambientes de rede complexos / João Porto de Albuquerque Pereira -- Campinas, [S.P. :s.n.], 2006.

Orientador : Paulo Lício de Geus

Tese (doutorado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Redes de computadores – Medidas de segurança. 2. Redes de computadores (Gerenciamento). 3. Modelagem de dados. 4. Internet (Redes de computação). I. Geus, Paulo Lício de. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Model-based configuration management of security systems in complex network environments

Palavras-chave em inglês (Keywords): 1. Computer networks – Security measures. 2. Computer networks – Management. 3. Data modeling. 4. Internet (Computer networks).

Área de concentração: Sistemas de Computação/ Redes de computadores

Titulação: Doutor em Ciência da Computação

Banca examinadora: Prof. Dr. Paulo Lício de Geus (IC-UNICAMP)
Prof. Dr. Joni da Silva Fraga (DAS-UFSC)
Prof. Dr. Edgard Jamhour (PPGIA-PUC-PR)
Prof. Dr. Ricardo Dahab (IC-UNICAMP)
Prof. Dr. Jacques Wainer (IC-UNICAMP)

Data da defesa: 24/05/2006

Programa de Pós-Graduação: Doutorado em Ciência da Computação

Gerenciamento Baseado em Modelos da Configuração de Sistemas de Segurança em Ambientes de Redes Complexos

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por João Porto de Albuquerque Pereira e aprovada pela Banca Examinadora.

Campinas, 24 de maio de 2006.

Prof. Dr. Paulo Lício de Geus
IC-UNICAMP (Orientador)

Tese apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

(Folha com a assinatura da banca)

Prefácio

Esta tese colige os resultados de pesquisas realizadas durante meu programa de doutoramento, o qual se dividiu entre o Laboratório de Administração e Segurança de Sistemas (LAS) no Instituto de Computação da Universidade Estadual de Campinas (Unicamp) e o grupo de trabalho de Redes e Sistemas Distribuídos (*Arbeitsgruppe Rechnernetze und verteilte Systeme*) no Departamento de Informática da Universidade de Dortmund (Fachbereich Informatik der Universität Dortmund) na Alemanha. Os trabalhos se realizaram sob a orientação do Prof. Dr. Paulo Lício de Geus e do Prof. Dr. Heiko Krumm, que é co-orientador *de facto* desta tese, embora não apareça formalmente nessa função por força de restrições de regulamento.

Devido a esse caráter inter-institucional e internacional, a pesquisa foi desenvolvida na língua inglesa (*lingua franca*), que é, portanto, utilizada nos capítulos centrais da tese — em grande parte, versões revistas e estendidas de artigos anteriormente publicados. Uma introdução e um resumo das conclusões (capítulos 0 e 9, respectivamente) são, no entanto, oferecidos na língua portuguesa com o intuito de dar uma visão geral das motivações e contribuições da pesquisa ao leitor não proficiente em inglês.

Na impossibilidade de mencionar todas as várias pessoas que contribuíram para esta tese, gostaria de agradecer aqui àquelas que tiveram participação direta em sua realização e com as quais tenho o prazer de dividir os créditos dos acertos nela contidos (os desacertos vão inteiramente por minha conta). Aos professores Paulo Lício de Geus e Heiko Krumm agradeço por seus comentários, pelas trocas de idéias, pelas críticas, pelas pacientes revisões e pelo apoio, combinados a uma grande liberdade, fatores que possibilitaram e impulsionaram este trabalho. A Holger Isenberg e René Jeruschkat agradeço pelo trabalho em conjunto no contexto de suas teses de mestrado (*Diplomarbeiten*), que muito contribuiu para os resultados aqui alcançados. Sou grato também aos membros da banca de avaliação (Prof. Dr. Joni da Silva Fraga, Prof. Dr. Edgard Jamhour, Prof. Dr. Ricardo Dahab e Prof. Dr. Jacques Wainer) pelas importantes críticas e comentários, e a Helen Mary Murphy Peres Teixeira pela revisão de meus artigos.

Esse trabalho não se teria realizado sem o apoio abnegado e o companheirismo de Gabriela. A gratidão que lhe devo está muito além da que posso aqui exprimir.

J. P. A. P., Campinas, junho de 2006.

A Gualberto Cavalcante Porto

Resumo

Os mecanismos de segurança empregados em ambientes de redes atuais têm complexidade crescente e o gerenciamento de suas configurações adquire um papel fundamental para proteção desses ambientes. Particularmente em redes de computadores de larga escala, os administradores de segurança se vêem confrontados com o desafio de projetar, implementar, manter e monitorar um elevado número de mecanismos, os quais possuem sintaxes de configuração heterogêneas e complicadas. Uma conseqüência dessa situação é que erros de configuração são causas freqüentes de vulnerabilidades de segurança.

O presente trabalho oferece uma sistemática para o gerenciamento da configuração de sistemas de segurança de redes que corresponde especialmente às necessidades dos ambientes complexos encontrados em organizações atuais. A abordagem, construída segundo o paradigma de *Gerenciamento Baseado em Modelos*, inclui uma técnica de modelagem que trata uniformemente diferentes tipos de mecanismos e permite que o projeto de suas configurações seja executado de forma modular, mediante um modelo orientado a objetos. Esse modelo é segmentado em *Subsistemas Abstratos*, os quais encerram um grupo de mecanismos de segurança e outras entidades relevantes do sistema — incluindo seus diferentes tipos de mecanismo e as inter-relações recíprocas entre eles.

Uma ferramenta de *software* apóia a abordagem, oferecendo um diagrama para edição de modelos que inclui técnicas de visualização de *foco e contexto*. Essas técnicas são particularmente adaptadas para cenários de larga escala, possibilitando ao usuário a especificação de certa parte do sistema sem perder de vista o contexto maior no qual essa parte se encaixa. Após a conclusão da modelagem, a ferramenta deriva automaticamente parâmetros de configuração para cada mecanismo de segurança do sistema, em um processo denominado refinamento de políticas.

Os principais resultados deste trabalho podem ser sumarizados nos seguintes pontos: (i) uma técnica de modelagem uniforme e escalável para o gerenciamento de sistemas de segurança em ambientes complexos e de larga escala; (ii) um processo para o projeto de configurações apoiado por uma ferramenta que inclui técnicas de foco e contexto para melhor visualização e manipulação de grandes modelos; (iii) uma abordagem formal para a validação do processo de refinamento de políticas.

Abstract

The security mechanisms employed in current networked environments are increasingly complex, and their configuration management has an important role for the protection of these environments. Especially in large scale networks, security administrators are faced with the challenge of designing, deploying, maintaining and monitoring a huge number of mechanisms, most of which have complicated and heterogeneous configuration syntaxes. Consequently, configuration errors are nowadays a frequent cause of security vulnerabilities.

This work offers an approach to the configuration management of network security systems specially suited to the needs of the complex environments of today's organisations. The approach relies upon the *Model-Based Management* (MBM) paradigm and includes a modelling framework that allows the design of security systems to be performed in a modular fashion, by means of an object-oriented model. This model is segmented into logical units (so-called *Abstract Subsystems*) that enclose a group of security mechanisms and other relevant system entities, offering a more abstract representation of them. In this manner, the administrator is able to design a security system—including its different mechanism types and their mutual relations—by means of an abstract and uniform modelling technique.

A software tool supports the approach, offering a diagram editor for models, which includes focus and context visualisation techniques. These techniques are particularly suitable to large-scale scenarios, enabling a designer to precisely specify a given part of the system without losing the picture of the context to which this part belongs. After the model is complete, the tool automatically derives configuration parameters for each security mechanism in the system, in a process called policy refinement.

The major results of this work can be summarised as follows: (i) definition of a uniform and scalable object-oriented modelling framework for the configuration management of large, complex network security systems; (ii) development of a configuration design process assistes by a tool that implements *focus and context* techniques to improve visualisation and manipulation of large models; (iii) a formal validation approach of the policy refinement process.

Contents

List of Figures

Chapter 0

Introdução

A ampla utilização de computadores e tecnologias de comunicação de dados, conectados a uma crescente Internet, requer a adoção de medidas de proteção para controlar o risco de ataques através da rede. Com tal intuito são empregados os *sistemas de segurança de redes*, que podem ser vistos como um conjunto de *serviços de segurança*, visando prover a operação e comunicação seguras da rede, nos moldes ditados pela política de segurança da organização. Cada um desses serviços de segurança é implementado por um ou mais *mecanismos de segurança*.

Em consonância com as necessidades de proteção dos ambientes de rede atuais, as tecnologias dos sistemas de segurança de redes têm-se complexificado significativamente. Aos tradicionais *firewalls* [119] — baseados em filtragem de pacotes — incorporam-se mecanismos como Redes Privadas Virtuais (VPNs) [103], associações criptográficas fim-a-fim — utilizando, por exemplo, IPSec [57] —, gerenciamento descentralizado de confiança (*decentralized trust management*) [12, 11] e o uso de pontos de aplicação distribuídos para uma política global (*firewalls distribuídos*) [9, 51].

Em direção oposta, há um crescente esforço pela definição de padrões e normas para o projeto de alto nível da segurança da informação de organizações. Padrões como o ISO/IEC 17799:2000 [49] — derivado da primeira parte da norma britânica BS 7799 (*Código de Prática para a Gestão da Segurança da Informação*)[1] — especificam guias de conduta e conjuntos de melhores práticas para a definição de controles de alto nível de abstração relacionados à segurança da informação.

No entanto, a transição dessas políticas e controles de segurança de alto nível para a efetiva implementação dos mecanismos é, na maioria das vezes, abrupta e sujeita a erros. O processo usual parte de uma descrição em alto nível de abstração diretamente para a implementação de um sistema complexo, composto de diversos tipos de mecanismos — os quais possuem, na maioria das vezes, sintaxes de configuração com idiossincrasias completamente discrepantes. Assim, o administrador da segurança se vê confrontado com a complexa tarefa de ajustar corretamente as configurações desses diferentes mecanismos, para que assegurem o cumprimento das políticas de sua organização. Como novas vulnerabilidades e técnicas de invasão são descobertas diariamente,

[1] Esse padrão foi traduzido para o português e incorporado como norma técnica da ABNT como NBR ISO/IEC 17799 [5].

os mecanismos necessitam, ainda, ser continuamente reconfigurados para acompanhar a evolução das ameaças.

Enquanto que um progresso significativo foi alcançado na melhoria da tecnologia de segurança de redes nos últimos anos, apenas uma modesta atenção tem sido dada à sua interface de configuração. Há, de fato, alguns produtos com uma interface gráfica um pouco mais palatável. Estes restringem-se, entretanto, a tipos particulares de mecanismos, não mitigando a árdua tarefa da configuração da segurança como um todo. Na prática, o administrador da segurança se vê obrigado a lidar com sintaxes de configuração complicadas e heterogêneas, a maioria das quais não é intuitiva e, em alguns casos, até mesmo induz ao erro. (ver por exemplo [117]). A situação é especialmente dramática em ambientes de larga escala, nos quais se torna muito difícil obter uma visão global dos numerosos mecanismos de segurança empregados, os quais têm de ser postos em harmônica cooperação. Nesse contexto, um único desajuste entre dois quaisquer mecanismos pode deixar vulnerável todo o sistema. Além disso, a distância entre a especificação das políticas em alto nível e a implementação torna difícil, também, a auditoria dos mecanismos de segurança empregados, isto é, a verificação de que estes aplicam de fato as políticas de alto nível especificadas.

Tendo em conta o cenário delineado, não deveria surpreender a conclusão do paradigmático trabalho de Anderson: "a maioria das falhas de segurança se deve a erros de implementação e gerenciamento"[3]. Além disso, a situação não parece ter evoluído satisfatoriamente, pois os mesmos resultados foram confirmados aproximadamente dez anos depois num recente estudo sobre três serviços de grande porte na Internet [78]. Este estudo conclui que os erros de configuração constituem a maior categoria dos erros de operação – estes sendo, mesmo, a causa mais freqüente de falha em dois dos três serviços analisados.

Por conseguinte, abordagens que ofereçam abstração, integração e ferramentas de suporte ao gerenciamento da configuração de mecanismos de segurança são fundamentais para tornar o processo de configuração menos sujeito a erros e mais efetivo. Dentro desse processo, quatro tarefas básicas podem ser distinguidas do ponto de vista do administrador:

(i) o projeto do sistema de segurança, incluindo a definição das tecnologias e mecanismos a serem empregados, assim como o posicionamento dos diferentes componentes dentro da rede;

(ii) a implantação da configuração projetada no sistema real;

(iii) a manutenção da configuração, possibilitando a introdução de mudanças, de forma a obter adaptabilidade em face a novos requisitos;

(iv) monitoramento do sistema durante a operação, assegurando um funcionamento compatível com o comportamento esperado.

O presente trabalho abrange as três primeiras etapas do gerenciamento da configuração (para trabalhos relativos ao monitoramento ver [118] e [30]). Apresenta-se aqui uma nova abordagem ao problema da configuração de sistemas de segurança que oferece uma solução abrangente para o gerenciamento das configurações de mecanismos nos ambientes de rede complexos e de larga escala encontrados em organizações atuais.

Para apoiar a fase de projeto, emprega-se uma técnica de modelagem que viabiliza o projeto modular do sistema de segurança a ser gerenciado, mediante um modelo orientado a objetos. Esse modelo é segmentado em unidades lógicas (denominadas *Abstract Subsystems*, ou subsistemas abstratos) que abarcam um grupo de mecanismos de segurança e outras entidades relevantes do sistema, oferecendo, também, uma representação mais abstrata deles. Dessa forma, o administrador do sistema pode projetar um sistema de segurança – incluindo seus diferentes tipos de mecanismos e suas relações mútuas – por meio de uma técnica de modelagem abstrata e uniforme.

Uma ferramenta de suporte apóia a modelagem, provendo um editor gráfico de modelos. Esse editor incorpora conceitos de *foco e contexto* – que são originários da pesquisa em visualização da informação [18] –, através das técnicas *visão olho-de-peixe* (*fisheye-view*) [34] e *zoom semântico* (*semantic zooming*) [60, 76]. Além disso, este trabalho se utiliza das abordagens de hierarquias de política [73] e gerenciamento baseado em modelos [63] para assistir às fases de implantação e manutenção supra-citadas. Assim, um modelo do sistema organizado em diferentes níveis de abstração proporciona uma modelagem assistida passo-a-passo pela ferramenta, culminando em um refinamento automático[2] das políticas até a geração de parâmetros de configuração de baixo nível para os mecanismos. Logo, enquanto que este refinamento automático contempla a fase de implantação da configuração, o suporte à fase de manutenção é provido pela possibilidade de edição dos modelos e repetição do processo de geração automática.

Para assegurar que o comportamento do sistema resultante da configuração gerada estará em conformidade com as políticas abstratas definidas pelo usuário, este trabalho inclui também uma validação formal do processo de construção de hierarquias de políticas. A validação inicia com a formalização do modelo gráfico em uma notação algébrica, a qual é então utilizada para definir condições de consistência para que uma instância do modelo seja válida. Com o intuito de analizar o efeito produzido pela configuração gerada sobre o ambiente de rede real, uma série de axiomas são definidos para capturar as assunções que estão implícitas na modelagem. Em seguida, dois teoremas provam que as condições de consistência definidas — em conjunto com os axiomas — são suficientes e necessárias para garantir a validade do refinamento, isto é, elas implicam a *corretude* do refinamento. Prova-se, também, que os algoritmos de refinamento e o esforço adicional necessário para a validação de instâncias do modelo tem um desempenho satisfatório em relação ao crescimento dos modelos (escalabilidade), sendo, portanto, apropriados para tratar os ambientes de rede complexos e de larga escala do mundo real.

0.1 Organização da Tese

Esta tese consolida e estende resultados de diversas publicações anteriores do autor [83, 85, 86, 87, 88, 89, 90, 91, 92]. Além disso, duas teses de mestrado foram co-orientadas pelo autor na

[2] O processo de refinamento de políticas é também denominado "transformação de políticas" (*policy transformation*) em alguns trabalhos da área como [115, 113].

Universidade de Dortmund (Alemanha) [53, 55] e também contribuem com resultados para o presente trabalho. Como o conteúdo desta tese está relacionado em muitos pontos com esses trabalhos anteriores, eles não serão explicitamente mencionados no texto — exceção feita em referências a tópicos discutidos mais detalhadamente em alguma publicação em particular.

O restante desta tese está organizado nos seguintes capítulos.

Capítulo 1. Duplicata da atual introdução na língua inglesa.

Capítulo 2. Conceitos básicos relativos ao domínio do problema tratado neste trabalho são apresentados brevemente (e.g. serviços de segurança, mecanismos de segurança de redes) e ferramentas e abordagens existentes para o gerenciamento de sistemas de segurança são apresentadas e discutidas. Em particular, trabalhos anteriores em *Gerenciamento Baseado em Modelos* (MBM, na sigla em inglês) — paradigma sobre o qual se assenta este trabalho — são descritos detalhadamente e comparados a outras abordagens. Verifica-se, também, os problemas que esses trabalhos demonstram no tratamento de ambientes de rede complexos.

Capítulo 3. Contextualização e definição do escopo de pesquisa deste trabalho. Um arcabouço conceitual discute as fases do projeto de sistemas de segurança dentro do contexto maior do projeto da segurança da informação. O objeto de pesquisa é localizado no nível de abstração de *projeto* e define-se objetivos específicos concernentes à obtenção de uma abordagem escalável para o gerenciamento da configuração de sistemas de segurança.

Capítulo 4. Uma técnica de modelagem é proposta através da definição de um meta-modelo em três camadas. A principal novidade neste modelo é a introdução do *Diagrama de Subsistemas Abstratos* (DAS, na sigla em inglês), um nível de abstração baseado no princípio de divisão e conquista. Um cenário simples serve para exemplificar e discutir o uso prático da técnica de modelagem desenvolvida.

Capítulo 5. O suporte oferecido pela abordagem à representação de políticas é analizado para, então, discutir-se o processo de refinamento automático de políticas. A ferramenta de suporte implementada é também descrita, com ênfase nas técnicas de *foco e contexto* incorporadas para aperfeiçoamento da navegação e visualização de grandes modelos.

Capítulo 6. Uma abordagem formal para a validação do refinamento de políticas é apresentada. Como ponto de partida, critérios para a validação são selecionados e justificados, e o meta-modelo é formalizado. Posteriormente, condições de consistência e axiomas são definidos, permitindo a prova de teoremas de generalização e validação. O desempenho e a significância da abordagem de validação proposta são também analisados.

Capítulo 7. Discussão da aplicação prática dos resultados de pesquisa anteriores. Dois exemplos simples de utilização são apresentados e um caso de estudo realista, que considera um ambiente de rede complexo, é analizado. Os resultados experimentais do estudo de caso são apresentados e discutidos.

Capítulo 8. Tece conclusões para o presente trabalho, discutindo os resultados obtidos e oferecendo possíveis direções para trabalhos futuros.

Capítulo 9. Resumo em português do capítulo anterior.

Apêndice A. Apresentação sucinta dos algoritmos de refinamento desenvolvidos.

Apêndice B. Provas dos teoremas de generalização apresentados na Seção 6.4.4.

Apêndice C. Outros modelos do estudo de caso, omitidos na Seção 7.2.

Chapter 1

Introduction

The utilisation of computers and data communication networks is growing notably, thus making them an essential resource to many kinds of organisations, as businesses, academic and governmental entities. In order to offer protection against network-based attacks, a great variety of security technologies and mechanisms is utilised in these networked environments.

As those security services and mechanisms are increasingly employed—attaining thereby dazzlingly knotty scenarios—importance and costs of security management escalate. In this scenario, the security administrator is faced with the complex task of setting up and fine-tuning these mechanisms in order to correctly enforce the security policy of the organisation. Since new vulnerabilities and intrusion techniques are discovered daily, the mechanisms' configurations also have to be continually adapted to accompany the evolution of the threats.

Whilst significant progress has been made on improving network security technology in recent years, only quite modest attention has been given to its configuration management. Though there are some products with a somewhat nicer graphical interface, these are restricted to their own particular mechanism types and thus do not mitigate the laborious task of security configuration as a whole. In practice, a security administrator must deal with a variety of complex and heterogeneous configuration syntaxes, most of which are unintuitive and in some cases even misleading (see [117] for instance). The situation is especially dramatic in large-scale environments where it is very hard to have a global view of the numerous security mechanisms that have to be put into harmonic cooperation. In these situations, a single maladjustment between two mechanisms can leave the whole system vulnerable.

Taking the outlined scenario into account, it is not surprising that Anderson's paradigmatic work concluded that "most security failures are due to implementation and management errors" [3]. Furthermore, this situation does not appear to have changed satisfactorily, since the same results were confirmed nearly ten years later by a recent study on three large-scale Internet services [78]. This work concluded that configuration errors are the largest category of operator errors (these themselves the leading cause of failure in two of the three services studied).

Therefore, along with the recent research approaches that consider improving the usability of security for the end-user (e.g. [2]), there is an urgent need to start treating security administrators and operators as “first-class users” [78], in order to make configuration management less error-prone and more effective.

From the standpoint of the system administrator, four basic tasks of the network security configuration management can be distinguished:

(i) the *design* of the security system, including the definition of technologies and mechanisms to be employed, as well as the placement of the diverse security components over the network;

(ii) the *deployment* of the designed configuration into the real system;

(iii) configuration *maintenance*, enabling the introduction of changes to achieve adaptability in face of new requirements;

(iv) *monitoring* of the system during run-time in order to assure compliance with expected behaviour.

As far as monitoring system activities is concerned, recently interesting user-centred approaches have been developed, such as NVisionIP [118] and a more general one suggested by Dourish and Redmiles [30]. This phase is not included in the scope of this work.

Instead, this work addresses the first three phases of the configuration management process. A novel approach to the configuration problem is introduced, yielding a comprehensive solution for the configuration management of security mechanisms found in the complex, large-scale network environments of today’s organisations.

To support the design phase, a modelling technique was developed to allow the design of the security system to be performed in a modular fashion, by means of an object-oriented system model. This model is segmented into logical units (so-called *Abstract Subsystems*) that enclose a group of security mechanisms and other relevant system entities, and also offer a more abstract representation of them. In the abstract view the details are hidden and dealt with in the internal specification of each subsystem, thereby improving the comprehensibility of the model. In this manner, the system administrator is able to design a security system—including its different mechanism types and their mutual relations—by means of an abstract and uniform modelling technique.

A software tool supports the modelling process, providing a graphical editor to input the models. This editor incorporates the concept of *focus & context*—that originated from research on information visualisation [18]—through the diagram navigation and visualisation techniques of fisheye-view [34] and semantic zooming [60, 76]. Combining the use of these techniques with the modelling developed enables a scalable handling of large system models: the designer can consider not only the system as a whole, visualising the interconnections and dependencies between abstract representations of its various components in a *Diagram of Abstract Subsystems* (DAS), but also examine and model each subsystem in detail, without losing the perspective of its insertion in the entire system.

This work also builds upon policy hierarchy [73, 115] and model-based management [63] approaches in order to assist the above-mentioned configuration management phases of deployment and maintenance. It employs models of the system and its security policies that are organised vertically in different abstraction layers—ranging from a business-oriented representation at the uppermost level, down to the Diagram of Abstract Subsystems. This organisation affords a stepwise, tool-assisted system modelling, along with an automated policy refinement (also known as *policy transformation* or *policy translation*) that culminates in the generation of low-level configuration parameters based on the model. While this guided derivation of low-level parameters contemplates the deployment of the security configuration, its maintenance is supported by the possibility of editing the models and repeating the automatic generation process.

Furthermore, in order to ensure that the system behaviour resulting from the generated configuration upholds the abstract policies defined by the modeller, the present work includes a formal validation approach for the policy hierarchy building process. The validation starts with the formalisation of the graphical model using an algebraic notation, which is thus utilised to define consistency conditions for a layered model to be valid. In order to reason about the effect of the generated configuration on the real environment, a series of axioms are defined to capture the assumptions that are implicit in the modelling. Subsequently, two theorems prove that the defined conditions are sufficient and necessary to guarantee the validity of the refinement; i.e. that they imply the refinement *correctness*. The refinement algorithms and the additional effort needed to validate model instances are also proved to scale satisfactorily—being thereby suited to realistic, large-scale environments.

1.1 Organisation

This work consolidates and extends research results from a number of previous publications [83, 85, 86, 87, 88, 89, 90, 91, 92]. Additionally, two diploma theses at the University of Dortmund [53, 55] were co-supervised by the author and contribute to the achievements here presented. Since the content of the present work is related in many points to these previous works, they will not be explicitly mentioned during the text—with the exception of references to topics further elaborated in some particular publication.

As this work was developed within a cooperation between the State University of Campinas (Unicamp) and the University of Dortmund in Germany, the body of this thesis is written in English (common denominator), whilst this introduction (duplicated in the previous chapter) and a summary of the conclusions are also presented in Portuguese. The reader who is proficient in English shall concentrate exclusively on the part written in English without lost of content.

The remaining of this work is organised into the following chapters.

Chapter 2. The basic concepts within the domain of problems that this work addresses are briefly presented (e.g. security services, network security mechanisms), and existing tools and approaches to the management of security systems are discussed. Particularly, previous work on Model-Based Management (MBM)—the paradigm upon which the present work relies—are explained and compared to the other approaches. The scalability problems of previous MBM work are also described. This chapter includes parts of the works [90, 92].

Chapter 3. Context and scope of this research work are defined by means of a conceptual framework that discusses the phases of the security system design within the wider context of the information security project (firstly presented in [85]). The research object is localized in the *design* abstraction level, and specific goals to be pursued are defined in order to achieve a scalable approach for the configuration management of security systems.

Chapter 4. A modelling framework is proposed that consists of a three-layered meta-model. The main novelty of this model is the introduction of the *Diagram of Abstract Subsystems* (DAS), an abstraction level based on the divide-and-conquer principle. A simple scenario is used to exemplify and discuss the usage of the modelling framework developed. This chapter encloses parts of [87, 90] and extends them.

Chapter 5. The policy support of the approach is analysed, and the policy refinement process—which automatically yields configuration files for the managed mechanisms—is discussed by means of an example. The supporting tool implemented is also described, with an emphasis on the *focus and context* techniques incorporated to improve visualisation and manipulation of large models. Parts of [83, 87] are the basis of this chapter.

Chapter 6. A formal approach to the policy refinement validation is presented. This approach was briefly described in [91] and was further elaborated in [83]. The validation criteria considered are firstly selected and justified, and the meta-model is formalised. Subsequently, consistency condition and axioms are defined, so that generalisation and validation theorems can be proved. The performance and meaningfulness of the validation approach are also analysed.

Chapter 7. The practicality of the research results achieved in the previous chapters is explained. Two simple utilisation examples are presented in detail and a realistic case study, which considers a complex network environment, is analysed. The experimental results gathered from a case study (that firstly appeared in [87, 90]) are presented and discussed.

Chapter 8. Casts conclusions, discussing the accomplishments of this work and pointing out to future work.

Chapter 9. Synopsis of the previous chapter in Portuguese.

Appendix A. Brief description of the refinement algorithms developed.

Appendix B. Proof of the generalisation theorems of Sect. 6.4.4.

Appendix C. Other models from the case study, which were omitted in Sect. 7.2.

Chapter 2

Basic Concepts and Related Work

The object of this work is the configuration management of *network security systems*. In this context, network security systems are defined as a set of *security services* that supports the secure operation and communication of networks, according to an organisation's security policy. Each one of these security services is implemented by means of one or more *security mechanisms*.

2.1 Network Security Services

Stallings [109] presents a useful classification of network security services, according to the security requirements they cover, into the following categories:

Confidentiality: services that protect transmitted data from passive attacks;

Authentication: services to assure that the communication is authentic both in terms of certifying that the entities are who they claim to be at the communication initiation; and that no third-party can impersonate one of the partners during the communication;

Integrity: services that assure that the message is received as sent;

Nonrepudiation: services that prevent either sender or receiver from denying a transmitted message;

Access Control: services that limit and control the access to host systems and applications via communication links;

Availability: services that prevent or recover from loss of availability of system elements due to attacks.

A central task in the design of a network security infrastructure consists thus of selecting and configuring mechanisms that implement the aforementioned services following the security policy of an organisation [10, Ch.26]. Nevertheless, the task of securing a network goes further, including for instance the hardening of configurations and hardware of workstations and servers, frequent backups, application of security update patches (for a comprehensive treatment of the subject see [36, 109, 21]). The concern of this work though is the design and management of proper configuration for network security services—which is itself far from being a trivial task, as argumented in Chapter 1.

There are many security mechanisms both from different commercial vendors and from the open-source community, such as end-to-end cryptographic associations (using, for instance, IPSec [57]); authentication and authorisation services (like Kerberos [70, 59]); diverse monitoring, logging and auditing systems; automated intrusion detection systems (IDS) and intrusion prevention systems (IPS)—to name just a few (see [36, 109] for a comprehensive listing).

In this work, the implementation focuses on two well-established security mechanism types: *firewalls* and *virtual private networks*; they will be succintly presented in the next sections. However, the approach is expected to be general enough to be easily adapted to cover other mechanisms.

2.1.1 Firewalls and Virtual Private Networks (VPNs)

Originally, the term *firewall* is used to name special walls that prevent fires from spreading to other parts of a building [67]. This metaphor was applied in the network security world to denominate special devices placed at the perimeter of a network, in order to protect this network against threats coming from the outside world; i.e. mostly from the Internet [119].

Further, firewalls can be placed not only between the network of an organisation and the public network, but also inside internal networks to isolate administrative domains (ADs). An AD is defined as a collection of network resources under control of a single administrative entity, with common security policy. Firewalls are today broadly defined as each device, software or equipment organisation that controls network traffic [21].

Chewisck et al. [21] identify the following firewall types:

Packet Filters: the simplest and cheapest firewall type (as they are incorporated in most routers), they discard packets depending on source and destination addresses, as well as on port numbers and flags—but without considering context information;

Application-Level Filtering: they are specific for a particular application and examine in detail the corresponding service at the application level (also known as *proxies* [84]);

Circuit-Level Gateways: they act at the transport level (TCP), acting as intermediary between client and server, and relaying packets received from one connection to the other (e.g. SOCKS);

Dynamic Packet Filters: add a stateful inspection to the simple packet filtering functionality, i.e. they keep track of the connection history, and can permit the traversing of packets that belong to open connections—e.g. replies to UDP queries;

Distributed Firewalls : the newest firewall type, it is still on the research stage [9, 51]; the main idea consists of making every host into a firewall that filters traffic to and from itself according to a policy that is centrally defined.

Spafford and Schuba [102] present a reference model for firewalls that prescribes a fundamental set of functions required by firewall systems to enforce network domain security policies. This set encloses five functions: (a) authentication function, (b) integrity function, (c) access control function, (d) audit function, and (e) access enforcement function[1].

[1] A loosely related work with a quite different scope is the firewall modelling utilising Petri nets formalism

Virtual Private Networks (VPNs) [103, 21], in turn, are used to build virtual infrastructures on existing infrastructures. They enable an expansion of a protected (private) network to encompass hosts that are outside the perimeter of the internal network. The communication from these hosts is tunnelled using cryptography through an insecure environment (generally the Internet) up to reaching the secure network, such that the external hosts behave virtually as internal hosts. There are three types of VPNs [21]:

- *Remote Branch Offices* enable the networks of two branches of the same company that are geographically separated to constitute a unique security perimeter and even to share the same address space;
- *Joint Ventures* enable the cooperation between two different organisations, offering to one of them access to certain services in the partner's network without having to open the services to everyone in the Internet;
- *Telecommuting* allows an employee to access resources of the organisation's network from home (or elsewhere) as if s/he would be physically at the office.

There are many firewall and VPN products on the market, from vendors such as Check Point [112], Cisco [19], Lucent Technologies, Symantec, and Network Associates, to name just a few (an updated list can be found at [44]). Open-source mechanisms also exist for different platforms, e.g. the Linux NetFilter [77], OpenBSD packet filter pf [82], and VPN solution isakmpd daemon [52] (the last two were used in this work).

2.2 Policy-Based Network Management

The approach of *policy-based network management* (PBNM) focuses on the automation of management tasks through management applications that perform tasks following the instructions of management policy descriptions. According to [104] "*a policy is a rule that defines a choice in the behaviour of a system*" and the separation of policies from the hard-coded implementation of management applications permits the policy to be modified flexibly. Therefore the strategy for managing the system can be changed dynamically. Policies can be enhanced and adapted to changing conditions without changing the implementation of the management system. The architecture of the policy-based management system is affected by the presence of policy enforcement units, each near to a corresponding managed resource [114]. Such policies are distributed and deployed to the units in the shape of dedicated descriptions.

A series of policy description languages already exist and standards are emerging in different fields, e.g. [15, 29, 106, 113]. The *Ponder* language reflects a more general and comprising policy approach [27]. Primarily it supports obligation and access control policies. Obligation policies state conditions and related actions. They can be used to define the steps of active management interventions. Access control policies provide elements for positive and negative authorization,

proposed in [101].

information filtering, delegation, and orders to refrain. They can be used to define the ranges of user, system and management actions. In connection with all policy elements, constraints can be defined to limit their applicability. Moreover, Ponder supports the structuring of policies in accordance to management domains and functions.

Policies to be supplied to and interpreted by resource-near enforcement units essentially express management actions in a low-level, resource-oriented and efficiently executable form, mostly following the "if *condition* then *action*" metaphor (e.g. [66]). Advantages of more abstract and system-integral policy descriptions, however, have been recognized early. In particular, the approach of policy hierarchies proposes the employment of hierarchical abstraction layers supporting the stepwise refinement of policies [73, 114]. Due to the increasing details and resource dependence of low-level policies—without additional means—the refinements cannot be computed mechanically. Further important design problems of management policies concern the global consistency of those policies, which are composed of different parts. The parts can reflect certain, but not necessarily orthogonal management aspects or are oriented at certain resources. Therefore, especially in complex and heterogeneous networked systems, global policies tend to contain incompatible or contradictory elements (see e.g. [56]).

2.3 Network Security Management Tools and Approaches

Though many of the firewall commercial products include configuration tools, they do not seem to have the integrated security management as primary concern. Nevertheless, a series of market product tools in this direction is slowly appearing, for example, the *Checkpoint* INSPECT visual policy editor [20], Cisco manager [42], and the product independent *Solsoft NP* Tool [31]. Moreover, Microsoft's Window-based group policy editor tool shows that the tool approach is not restricted to security and access control policies but can also cover user management and configuration management issues [4].

As for research work, there are already a number of different approaches that address the configuration management problem. They are classified into groups and discussed in the next sections.

2.3.1 Rule-based Approaches

Haixin et al. [39] present a policy-based framework for the automatic management of firewall rules in large networks. This framework is based on an access control policy with three "levels of abstraction": organisational, global, and local. The rule syntax used in all the three layers is a 3-tuple $\langle source, destination, action \rangle$, where source and destination are IP addresses (or ranges) and action is accept/deny. Consequently, the three levels correspond only to different topology levels, in which one is more general than the others but they do not have really different abstraction levels.

Lee et al. [64] describe a framework to specify high-level firewall rules using Ponder [26], and to implement such descriptions on reconfigurable hardware. It employs a two-level optimisation approach that allows software and hardware optimisations to proceed independently. Although the representation syntax for policies is the powerful Ponder language, which supports the generic definition of policies, the special policy types defined for this particular application are filtering rules-like.

While solving the problem of generating packet filter rules from device-independent specifications, these approaches do not really offer an abstract view of the problem. Some of the authors argue that the rules are defined in "high-level", but they in fact deal with transport- and network-level concepts like protocols and IP addresses, which are only grouped in domains. The abstraction is thus reduced to independence of particular syntaxes, but the policies are still expressed in a rule-based syntax similar to that of filtering rules in the form (event, action) or yet "*if condition then action*" (like in [75]).

Yet one step closer to the packet filter rules, there are open-source tools that support multiple flavours of packet filters, such as HLFL [43] and firewall builder [35]. Though these tools are able to support uniformly a number of mechanisms using a single configuration syntax, they do not model the topology. Further, their policy representation language has, in fact, roughly the same level of abstraction as the filter languages.

2.3.2 Logic-based Approaches

Guttman [38] presents an approach that defines a security policy for a network in a logic-based language, as a global policy about what packets can get where. It offers an additional method for solving the localisation problem; i.e. to determine the filtering decisions of individual routers. These decisions can be based only on local information. Additionally, an automated verification method for the localisation problem is provided. However, Guttman's approach does not provide a separation of the policies from the network topology, i.e. between the architectural and the security policy design tasks mentioned in Sect. 3.1. This makes policy editing and reuse much harder, as policies and system model are intertwined.

Burns et al. [16] present a tool that given a security policy specified in prolog-based language automatically checks that the configuration of a network with multiple firewalls adheres to the policy. The network topology, mechanism configuration and behaviour are also represented by logic assertions, so that a policy engine is able to verify the compliance of the a network state with the specified policy. The wide scope of this approach requires a very comprehensive modelling of network devices, capabilities, and configurations; the achievement of the ambitious project's goal depends on the success of this modelling. Though the authors are also concerned about extending the work to cover the modular validation of policies, this approach is not so mature like the preceding ones in this respect.

The logic-based approaches have the advantage of offering scalability gains and more comprehensive analysis capabilities, allowing the formal evaluation of system security properties. However, the syntax used is both far from the expert system view and from a business view of the problem, requiring extra training for the correct policy specification.

2.3.3 Trust Management Approaches

Blaze, Ioannidis and Keromytis introduce in [13] a policy management scheme for IPSec based on trust management. In the Strongman project [58], the authors present a more general solution in which multiple policy languages are used to specify security requirements for different application domains. Those high-level policy specifications are compiled into a common intermediate, assertion-based policy language *KeyNote* [11], such that different applications' policies can be composed. The focus here is to provide better support across application domains by separating high and low level policies and to achieve scaling by the local enforcement of global security policies.

A recent work of the same group introduces the concept of *Virtual Private Services*[50]. This work also splits policy specification and policy enforcement, striving to provide local autonomy within the constraints of a global security policy—which is also defined using the *KeyNote* system. Further, virtual security domains are created, each with its own security policy. The system is able to cope with scale and heterogeneity, by converting the global policy into a form in which it can be enforced locally.

While focusing on the management of highly distributed systems, these approaches offer a good treatment of issues like credential management, and policy delegation and distribution. For the purposes of the present work though—i.e. the configuration management of network security services within complex organisations—they make it hard for administrators to have an understandable view of the organisation's security policy, the network elements, and the relation between them. This is due to the fact that the policy language used is not high-level enough to provide an abstract, intelligible view of the problem.

2.3.4 Graphic and Analysis Tools

Firmato [6, 7] is a firewall management toolkit with: (1) an entity-relationship model containing both the security policy and network topology, (2) a model definition language, which is used as an interface to define an instance of the entity-relationship model, (3) a model compiler that translates a model instance into firewall-specific configuration files, and (4) a graphical firewall rules illustrator. The security policy modelling is executed in two phases. In the first phase, the modeller should define *roles* and describe which services the roles are permitted to use, regardless of its location on the network (i.e. a topology-independent phase) . In the second phase, the administrator specifies the assignments of roles to actual hosts. The commercial descendent of Firmato, the Lumeta system [116], includes extensions to initialise its topology model automatically by interrogating routers, and to analyse and transform existing rule sets discovering configuration weaknesses.

The *role* concept used in Firmato is similar to that of classical RBAC work [97], but permissions are associated to a pair of source/destination hosts, each of which has been assigned a role. This association is somewhat unintuitive, since the initiator and the target hosts are intertwined in the same abstract entity (roles). As for the rules illustrator, the picture produced is of difficult understanding, since nested host groups are separately represented and connected by edges. Thus, the actual network topology is hard to recognise and the representation of large models tends to get rapidly clumsy.

2.3.5 Hierarchical Approaches

The approaches presented in the previous section are mostly concerned with the technical view of the management problem. They proceed bottom-up, starting from the mechanisms to be configured and achieving representation languages that are too bound up with the mechanism types initially considered. However, there are already some approaches that turn the problem on its feet by starting from convenient abstract models that are able to address the management problem from a business point of view.

The Power prototype [74] aims at supporting the building of policy hierarchies by means of a tool-assisted policy refinement. In this prototype, the policy specification task is divided into the definition of policy templates by an “expert”; and the use of the wizards to create a business-level policy by a “consultant”. The tool is then able to refine the high-level policy according to the templates and finally producing the configuration information for particular devices. However, the complexity hidden from the consultant, must be faced by the expert, who is expected to define the policy and refinement rules in a prolog-like language. It is not obvious to me to which degree this process can be less error-prone than the mechanism configuration itself.

Cuppens et al. [25] present a formal approach to specify network security policies based on the the Or-BAC model, which is an extension of the Role-Based Access Control Model (RBAC). They use a XML syntax to specify an abstract level network security policy, and a derivation process that (1) distribute the abstract level network policy specification over several security components, generating a set of generic firewall rules; and (2) compile the generic rules into specific configuration files for the intended target platform. This work has the advantage of using the formal syntax of access control models to represent policies, yielding a higher-level view of the network policy based on organisations, roles, activities and views. However, the network architecture is not modelled explicitly, but this information is implicitly distributed in organisations and in the attribution of hosts to roles. This makes it hard for administrators to get a picture of the modelled environment and of what is actually allowed by the model, bringing difficulty also to the reuse of policies in different environments.

Both approaches lack analysis capabilities, so the problem of the correctness in the policy refinement is left open; i.e. the question whether the generated lower-level policies uphold the abstract policy remains not answered.

Another hierarchical approach is the Model-Based Management. Since the present work relies upon this approach, it is expounded and analysed in contrast to the other related work in the next section.

2.4 Model-Based Management (MBM)

The term Model-Based Management (MBM) is used in this work to name the management approach developed in works such as [62, 61, 63]. This approach consists of a particular type of policy-based management that supports the building of policy hierarchies by means of an interactive graphical design. It adopts concepts of object-oriented system design and relies upon a model of the system that is vertically structured into a set of layers. The objects and associations of a layer represent the system to be managed and security policies on a certain abstraction level. Based upon the model, a policy refinement can be accomplished such that configuration parameters for security mechanisms can be automatically derived. A software tool is also provided to assist the modelling of system objects and policies, as well as to support the automated policy refinement (for a more comprehensive explanation of the MBM approach see [92]).

The approach was initially used in the management of backup services [62], and later applied to the management of a series of network security services: packet filters (ipchains [100, 61] and CheckPoint Firewall-1 [108]), VPNs [111, 63], Kerberos [94], firewall logs [40], and to the integrated management of packet filters and VPNs [37]. Lück [68] offers an overview on the previous work and advances in the formalisation of the model and in the formal proof of the refinement correctness. The following section summarises the common model and main ideas of these works.

2.4.1 Previous Work on MBM of Network Security Services

The structure of the model is shown in Fig. 2.1 (reproduced from [37]), where three abstraction levels can be distinguished: *Roles & Objects* (RO), *Subjects & Resources* (SR), and *Processes & Hosts* (PH). Each level is a refinement of the superordinated level in the sense of a "policy hierarchy". The uppermost level represents the business-oriented view of the network whereas the lowest level is related to the technical view. The vertical subdivisions differentiate between the model of the actual managed system (with productive and control elements) and the policies that regulate this system. This last category encompasses requirement and permission objects, each of which refers to the model components of the same level and expresses security policies.

The uppermost level (RO) is based on concepts from Role-Based Access Control (RBAC) [97]. The classes in this level are:

Role: Roles in which people, who are working in the modelled environment, act.

Object: Objects of the modelled environment which should be subject to access control.

AccessMode: Ways of accessing objects.

The class *AccessPermission* allows the performer of a *Role* to access a particular *Object* in the way defined by *AccessMode*. In order to reduce the amount of necessary *AccessPermissions*, the classes *RoleSet*, *ObjectSet* e *AccessModeSet* were introduced.

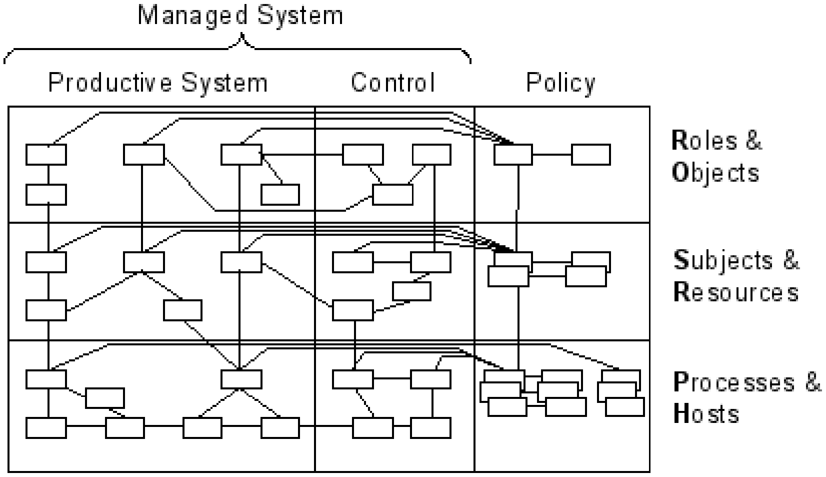


Figure 2.1: Common Meta-Model of Previous Work on MBM

An additional class *Constraint* was introduced to deal with special situations observed in business-oriented policies and their representation by *AccessPermissions*. The *Constraint* class is used to limit the number of *Objects* an *AccessPermission* applies to. A business policy example that illustrates this need is "Dial-in-users are allowed to use the Workstation in their offices". Here, each role performer (a user accessing the network via dial-in server) is allowed to use only the Workstation especially associated with him or her.

The second level (SR in Figure 2.1) consists of a more complex set of classes. Objects of these classes represent: (a) people working in the modelled environment (*User*); (b) subjects acting on the user's behalf (*Subject*); (c) sets of roles which are appropriate for the associated *Subject* (*ActiveRoleSet*); (d) services in the network that are used to access resources (*Services*)—a service has references to all resources it is able to access; (e) resources in the network, either real resources or *Services*, which in turn can be used as resources by other services (*Resource*); and (f) relations (*Relation*), that are used to express associations between *Users* and *Resources*. The *ServicePermission* class allows a subject to access a resource by using a service.

The SR level offers a transition from the business-oriented view, represented in RO level, to a more technical perspective, service-based. This fact is due to the combined use of the service-oriented management approach (see for instance [71]), achieving a relatively abstract view of the management system—which is defined from the point of view of the services that it will provide. Thus the system internal structure is not expressed in that level, but in the third level (PH) of the model (Figure 2.1).

The lowest level (PH) is responsible for modelling the mechanisms that will be used to implement the services defined in level SR. Therefore, it will have even more classes than before, representing for instance the *Hosts*, with their respective network *Interfaces*, and the *Processes*, that perform communicative actions relying on *Sockets*. *LinkPermissions* allow the transition of packets between processes. The authors define several other classes that will not be mentioned here for the sake of brevity.

Whereas the two uppermost levels are common to all previous works on MBM, the third one is specific to each application domain covered, comprising a different set of classes in each case (although the sets are not disjoint). This fact reveals a good abstraction-level segmentation since it was possible to keep the two uppermost levels homogeneous, even though dealing with pretty different security applications.

To support tool-assisted interactive configuration of the mechanisms—e.g. packet-filters and VPNs—specific graphical tools are supplied. The tools primarily consist of a graphical editor (see Figure 2.2) with the following functions:

- Interactive construction of the model by selecting and positioning of model-elements associations;
- Organisation of model-elements in folders;
- Checking of Model-dependent constraints;
- Configuration of attributes by means of property dialogues;
- Guided derivation of different permission variants through the model-hierarchy levels.

The editor allows the selection of classes from each level of the model. Associations between two objects can be modelled by drawing an edge using the mouse. By selecting the menu item 'Check Model', the tool checks whether the model is complete and consistent in the sense of given constraints. Moreover, dialogs ask for input where information for a model element is missing or faulty. Therefore, these steps support interactive derivation of *ServicePermissions* from *AccessPermissions*, *LinkPermissions* from *ServicePermissions*, and the appropriate configuration files from *LinkPermissions*.

Behind this refinement process there is a formal correctness proof [68]. For this purpose, the model entities and assumptions on the relationships between these entities are formalised using predicate logics. As such, theorems can be proved to ensure that the lower-level policies generated uphold the abstract policies modelled. Consequently, the configuration files derived will be in conformance with the high-level policies modelled.

2.4.2 Scalability Problems

A common problem of the existing applications of MBM occurs when dealing with larger systems, since the model tends to lose much of its understandability, getting obscure due to the great number of components (as attested in [37]). Indeed, it can be noted from the previous discussion that the PH level—which shall depict the entire system, with its processes, hosts, network interfaces etc.—has its complexity quickly increased as the size of the modelled system grows.

This fact is also illustrated in Fig. 2.2, which shows the model of a very simple scenario with only one *AccessPermission* at the uppermost level: the workers of a company shall be allowed to access the corporate web server. Despite the simplicity of this RO level, the model unfolds into

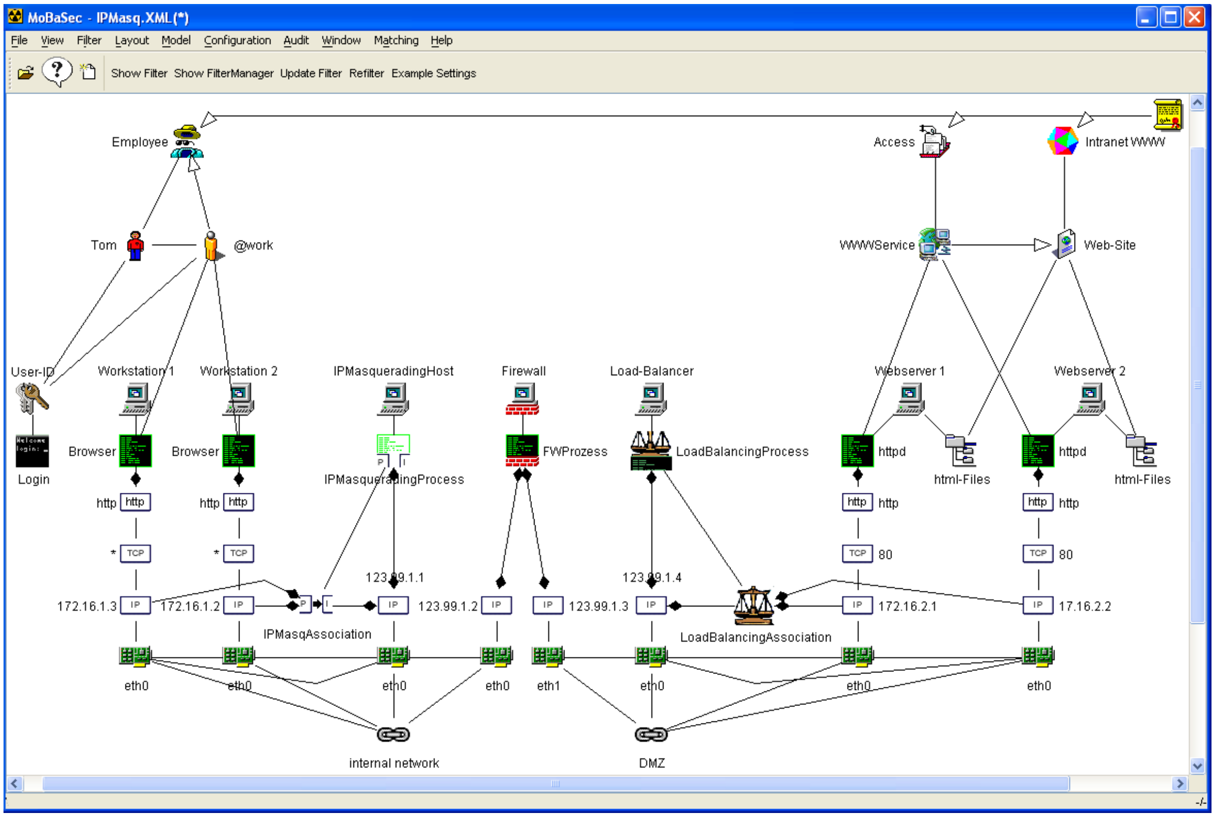


Figure 2.2: Tool Interface

a considerable number of objects at the lowest level (bottom of Fig. 2.2), in order to represent mechanisms like IP-masquerading, firewalls and load balancers.

Due to the simpleness of this example, the resulting model is still reasonable. However, it can be noted that models of larger real environments tend to become quite confusing. Furthermore, the refinement algorithms must consider this large number of model elements, for example, in costly path discovery processes. This results in a restricted support for the realistic, complex network environments of today's organisations.

2.4.3 Comparison between MBM and other works

The Model-Based Management paradigm divides the configuration process in three abstraction levels. This division has been shown to be adequate since it was possible to keep homogeneous the two uppermost levels dealing with two different applications. Therefore, the abstraction provided by MBM is advantageous in comparison to the graphical tool Firmato and to the rule-based approaches analysed in Sect. 2.3, which are restricted to the particular network security services they address. MBM, in contrast, is able to uniformly cover different services.

In MBM, policies are defined by the modeller only at the most abstract level, using the concise notation of Role-Based Access Control models. This makes the policy specification process less error-prone in comparison to the logic-based and trust management approaches described in Sect. 2.3, which rely on lower-level policy languages that have a more complicated syntax. On the other hand, MBM employs a central policy definition process, which is not beneficial in the highly distributed and autonomous environments covered by trust management work. This is not a critical issue for the purposes of the present work though, since an organisation is assumed to have control over its own security policy definition.

Furthermore, MBM uses a static system model and thus does not offer the behavioural analysis of network states striven by Burns et al. [16]. Nevertheless, MBM does support the analysis of policy hierarchies, enabling the verification of the *correctness* of the generated low-level permissions against the high-level policies modelled—which is missing in the other hierarchical approaches reviewed in Sect. 2.3. I believe that the policy hierarchy analysis is of greater importance and perhaps it is indeed more feasible in complex environments than a comprehensive network state analysis.

As for the system representation, the object-oriented modelling and the tool-assisted policy refinement supplied by MBM make the model development more comfortable and quality-assured. The administrator is provided with a means for specifying an understandable system model that is near from her/his mental picture of the network. Indeed, differently from the graphical and analysis approaches of Sect. 2.3, the system model in MBM directly corresponds to the actual network mechanisms. However, the existing applications of MBM have problems to support realistic, large-scale environments, since the representation of large models gets rapidly confused and the refinement algorithms do not scale adequately.

2.5 Chapter Summary

This chapter has introduced the network security services employed in current networks. Particularly, two security mechanism types were described, firewalls and VPNs. Further, the policy-based management approach and previous work on the configuration management of network security services were exposed and analysed. Pondering advantages and drawbacks, the Model-Based Management (MBM) offers an adequate foundation to address the purposes of the present work. Nevertheless, existing applications of the MBM paradigm have proved to have problems in coping with large models.

Chapter 3

Research Approach and Goals

A paradigm shift is underway, and a number of recent threads point towards a fusion of security with software engineering or at the very least to an influx of software engineering ideas

Ross Anderson [3]

The first task in the management of network security services is the design of the adequate network security infrastructure for an organisation, or the development of a network security system. However, this task is not performed alone, but as a part of a complex and comprehensive process: the information security project of the organisation. This chapter inverts the perspective of the previous one: the development of network security systems is observed not from the point of view of the services to be configured (bottom-up), but rather within the context of the organisation's information security project as a whole (top-down).

In pursuit of this goal, a conceptual framework for the development of a network security system is proposed [85]. The framework analyses the network security design in comparison to the various abstraction levels and phases that constitute the development life-cycle of a organisation's information security project. Thus, the framework acts here both as a contextual orientation and as an abstract requirements definition for the particular phase addressed by this work: the network security system design.

3.1 Conceptual Framework

Figure 3.1 schematically illustrates the abstraction levels of the network security system development cycle. This framework follows the classical software engineering "waterfall" life-cycle paradigm [93], in which the development proceeds in top-down fashion. Besides, the framework is also based on Sterne's security policy classification [110] and on the concept of policy hierarchies [73, 115].

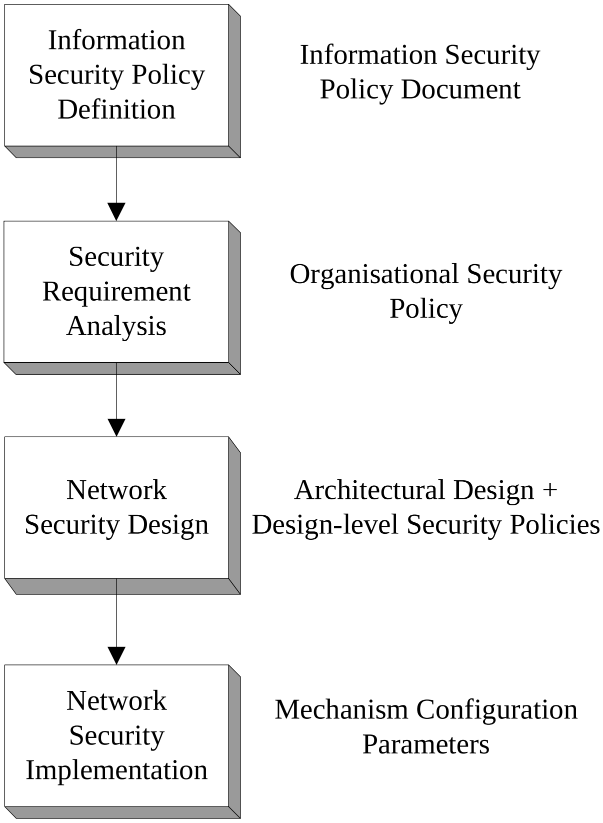

Information Security Policy Definition
Information Security Policy Document
Security Requirement Analysis
Organisational Security Policy
Network Security Design
Architectural Design + Design-level Security Policies
Network Security Implementation
Mechanism Configuration Parameters

Figure 3.1: Development Life-cycle of Network Security Systems

The first phase of the framework (Figure 3.1) is the*Information Security Requirements Definition*, which consists of the elaboration of the *Information Security Policy Document* with the goal of providing “management direction and support for information security” [49]. The network security system is just a small part (although an important one) of an organisation’s information security infrastructure and must be considered together with “several other fields, such as physical security, personnel security, operations security, communication security, and social mechanisms” (Icove, cited by Schuba and Spafford [102]). Therefore, the definition of security policies in this phase must have a broad scope, usually done by performing an enterprise business risk analysis following a guideline manual such as the ISO/IEC 17799 [49]. The final product of this step is a document in natural language describing a set of high-level information security policies and controls.

The next step, *Information Security Requirement Analysis*, consists of the (preferably formal) representation of the previous high-level security policies, achieving a set of laws, rules, and practices that regulate how an organisation manages, protects, and distributes information resources. Compared to the previous phase, the analysis aims to establish *how* the security objectives previously gathered can be accomplished. The resulting set will be thereafter called *Organisational Security Policy*—using a term proposed by Sterne [110].

This second phase is not always done with a formal approach—following Gödel's definition that a specification is formal if it is mechanically checkable [81, p. 14]—but there are benefits that make it worth, e.g. a formal model can be *analysed* to detect conflicts between policies, and its formal character eliminates ambiguities that may be present in the natural language high-level policies. Despite the several research efforts that have been done on formal specification of security requirements (since the seminal work of Bell and La Padula [8]) there still are important technical challenges in this field—as pointed out by Rushby [95]—making this analysis not always easy. Leiwo and Zheng [65] present a conceptual framework for dealing with high-level policies using a formal approach, allowing conflict detection and harmonisation in a layered fashion.

The following phase is the *Security System Design*, which comprises the process of identifying the aspects of the organisational security policy that can be automated and then selecting the appropriate enforcement instruments: the security services[1]. The main goal of the present work is to address this phase. In our context, only the technical security policies will be concerned from this phase on. More specifically, the focus shall be on *network domain security policies*, following the definition of Schuba and Spafford [102]: "a subset of a security policy, addressing requirements for authenticity and integrity of communication traffic (...), authorisation requirements for access requests (...), and auditing requirements". These four types of security policies, namely *authenticity and integrity*, *confidentiality*, *access control*, and *auditing*, are the ones that will be enforced by network security services such as packet filters, proxy agents [84], end-to-end cryptographic associations [57], and logging agents (see Sect. 2.1).

The last phase of the framework, in turn, is the *Network Security System Implementation*. It comprises the selection and configuration of the network security services already presented in Sect. 2.1.

Since this work focuses on the design phase, it will be further elaborated in the next section.

3.1.1 Network Security System Design

The *Network Security System Design* task aims at creating a formal model of the system to be actually implemented. This model must represent: (a) each security service that will be used; (b) the interaction among different services; and (c) the link between each high-level security policy and the corresponding components that will enforce it. For that purpose this phase can be conceptually subdivided into two parts: the *Security Policy Design* and the *Architectural Design*.

The *Security Policy Design* consists of achieving a set of policies characterised as being near enough to technical implementation—that is, mechanically executable—but vendor- and device-independent. A smooth transition from the policies in the previous phase (*Information*

[1] The first three levels of this framework (Figure 3.1) correspond, respectively, to the following types of policy proposed by Sterne: "Security Policy Objective", "Organisational Security Policy", and "Automated Security Policy" [110]. This author, however, does not explicitly mention hierarchical levels in his work.

Security Requirement Analysis) to the set of policies in this level should be also provided, so that the lower-level permissions can be traced back to the corresponding abstract policies. Further, the configuration parameters of mechanisms in the implementation phase of the framework (see Fig. 3.1) should be ideally automatically derived from the design-level security policies, in order to improve the reliability of the system configuration.

The *Architectural Design* in turn aims at establishing the overall structure of the network security system, by representing the several mechanisms that implement security services—such as routers, workstations, packet filters, VPN gateways—and their interactions, i.e. the communication flows among them. This *Architectural Design* approach is inspired in that adopted by most of software engineering methodologies [93, 107, 14], and is generally associated with the improvement of quality attributes such as performance, reliability, modifiability, and maintainability [107, Ch. 10].

A characteristic that can be borrowed from software engineering work on architectural design is *compositionality* or *hierarchical decomposition*, defined as a mechanism that allows architectures to describe systems at different levels of detail: complex structure and behaviour may be explicitly represented or they may be abstracted away into a single component [72]. In this manner, the model is capable of representing lower-level subsystems yet to be developed as a black-box component in a high-level model. Those subsystems could then be independently developed, in a *top-down* fashion development process also called refinement or *zooming-in*. A *bottom-up* strategy might also be used: the development can begin with lower-level subsystems and then go to the model of higher-level functionalities using the pre-defined lower-level black-boxes—using abstraction or *zooming-out*.

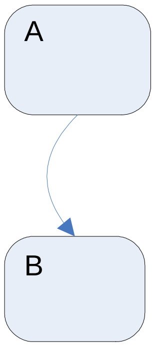


Figure 3.2: *Zooming-in*

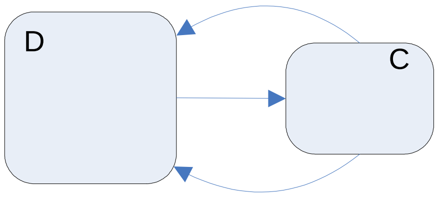


Figure 3.3: *Zooming-out*

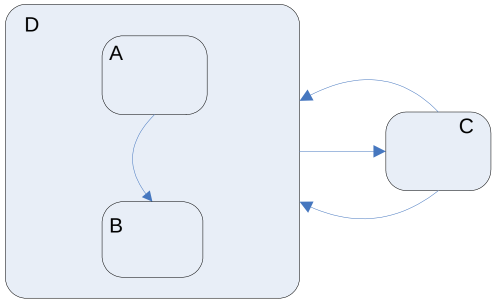


Figure 3.4: Complete System

An example of these features can be found in the classical work of Harel that introduces *Statechart* diagrams [41]. Figs. 3.2, 3.3 and 3.4 are adapted from that work and illustrate both *zooming-in* and *zooming-out*. The first is presented in Fig. 3.2 and is achieved by looking 'inside' a given component D (in Harel's work, a state), disregarding external interface for the time being. Figure 3.3 is the result of *zooming-out* the system in Fig. 3.4, i.e. of eliminating the inside of D by abstraction. Therefore, the *compositionality* feature is able to make the modelling process more flexible and to produce models both more understandable and more scalable, as a result of both *zooming-in*—considering only the elements inside a given component, and eliminating thereby the complexity of all remaining components—and *zooming-out*—hiding the complexity of the 'inside' of particular components to define the interactions among them in a more abstract fashion.

Additionally, the architectural components must also be connected to the policies, so that at the end of the design phase the overall structure of the network security system is precisely defined by the architectural model, and the functional responsibility of each system component is determined by a set of associated security policies. However, it is important that the definition of policies and system architecture is done independently, so changes in one of this elements can be propagated to the other without having to re-specifying the whole model.

3.2 Research Goals

By comparing the Model-Based Management (Sect. 2.4) paradigm and the framework previously described, one can notice that MBM operates precisely in the abstraction level of the *Network Security System Design* phase. As regards to policies, MBM offers an elegant solution for the *Security Policy Design* tasks, since it provides not only an instrument for formalising the part of the organisational security policy related to network services (by means of the RBAC model), but also an automated refinement process that creates lower-level policies from the abstract ones. As such, a guided and smooth transition from the highest-level to lower-level policies is supplied, culminating with the automated derivation of configuration parameters for the network security services.

Nevertheless, the features prescribed for the *Architectural Design* are not adequately covered by the existing applications of MBM. In those works, the actual system is modelled in a 1:1 basis, i.e. to each element in the actual system there is an object in the model. As such, the modeller cannot have an abstract and intelligible view of the system as whole. This is also the cause of the scalability problems discussed in Sect. 2.4.2, since the system representation used has its complexity increased at the same pace as the complexity increase of the real system.

A canonical way of addressing such problems is to use the "divide and conquer" approach (from the Latin *divide et impera*); i.e. to break the problem into several subproblems that are similar to the original problem but smaller in size, solve the subproblems separately, and then combine these solutions to create a solution to the original problem [23, p. 12]. This approach thus involves three steps:

(1) *divide* the problem into subproblems;

(2) *conquer* the subproblems by solving them;

(3) *combine* the solution to the subproblems into the solution for the original problem.

By applying this principle to network architectures, one can use the architectural design technique of *compositionality* described in the previous section, such that the modularisation of a system into smaller segments (*divide*) would allow us to deal with each of them in detail separately (*conquer*). Then we would be able to reason about the whole system through a more abstract view of the composition of those parts (*combine*).

Therefore, the driving idea of the present research work is to apply the *divide and conquer* principle to the management of large-scale network security systems, relying upon the model-based management paradigm on the one hand, and upon lessons learned from the architectural design of software engineering techniques on the other hand. The focus is thus to provide the security administrator with a practical approach for the configuration management of network security systems which is suited to the large-scale, complex network environments found in today's organisations.

Specifically, this work strives to achieve the following goals:

1. an abstract modelling of the system architecture—which enables one to reason about the system building blocks, about the interaction between blocks, and about the distribution of policy participants over the system—combined with a detailed modelling of particular model segments;

2. a graphical representation that encloses both modelling forms and the system policies, providing thereby convenient visualisation and manipulation of large models;

3. policy refinement algorithms that are able to cope with large systems;

4. formal validation of the policy refinement correctness, which must also scale satisfactorily;

5. a software tool that assists the editing of models and that apply the refinement algorithms to produce configuration parameters for security mechanisms.

3.3 Chapter Summary

This chapter has presented a conceptual framework to the development life-cycle of network security systems. This framework contextualises the network security system design and subdivides it into two tasks: the architectural design and the security policy design.

With basis on the conceptual framework and on the network security services and approaches discussed in Ch. 2 on the other hand, specific research goals for the present work were defined. The five selected goals concern the achievement of a scalable approach for the configuration management of large-scale network security systems.

Chapter 4

Modelling Framework and Usage

This chapter introduces the modelling technique of this work by presenting its underlying meta-model; i.e. by defining the common structure behind each model instance. The meta-model defines each class that is available for the modeller, and also the connection possibilities between classes. Said it another way, the meta-model lists for a class A all the other classes whose objects can be connected to objects that pertain to A. The meta-model could also be called an *ontology*, since it defines the entity classes and potential relationships between them in the "world" of this work.

This chapter thus concerns the objects that shall be inputted by the modeller. After the definition of the meta-model in the next section, Sect. 4.2 explains the usage of the modelling framework by means of a test scenario. The model objects that are automatically generated in the policy refinement process are not described in this chapter, but in Ch. 5.

4.1 Meta-Model Definition

Figure 4.1 depicts the three-layered structure of the meta-model. Each of the levels is a refinement of the superior one in the sense of a "policy hierarchy" [73, 115]; i.e. as we go down from one layer to another, the higher-level system's view contained in the upper level is complemented by the lower-level system representation, which is more detailed and closer to the real system.

Thus, the horizontal dashed lines of Fig. 4.1 delimit the abstraction levels of the model: *Roles & Objects* (RO), *Subjects & Resources* (SR), and *Diagram of Abstract Subsystems* (DAS). As for the vertical subdivision, it differentiates between the model of the actual managed system (on the left-hand side) and the security policies that regulate this system (on the right-hand side).

The two topmost levels are gathered from previous work on MBM [61, 63] and extended. The RO level is based on concepts from Role-Based Access Control (RBAC) [32, 97] and the second level (SR in Fig. 4.1) offers a system view defined on the basis of the services that will be provided. They are both briefly described in Sect. 4.1.1.

The third model layer (DAS) is the main contribution of this work. It aims at offering a

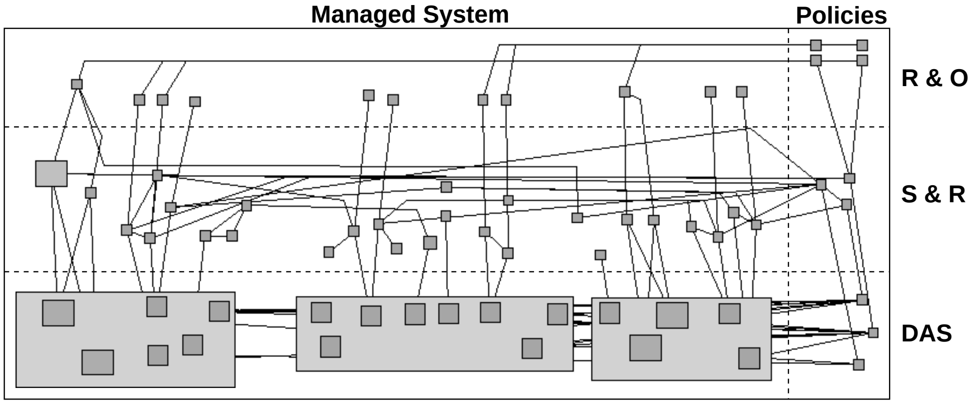


Figure 4.1: Meta-Model Overview

modular description of the system's *overall structure* thereby improving the scalability of the modelling technique.

4.1.1 Layers RO and SR

The topmost level of our model describes the system relying on the concept of roles; i.e. system permissions are established based on functional roles in the enterprise, and then appropriately assigned to users [32]. The main classes in this level are depicted on the top of Fig. 7.4 and represent: *Roles* in which people who are working in the modelled environment act; *Objects* that should be subject to access control; and *AccessModes*; i.e. the ways of accessing objects. The class *AccessPermission* expresses an authorisation policy, allowing the performer of a *Role* to access a particular *Object* in the way defined by *AccessMode*. Figure 4.2 shows the graphical representation used for each of these classes.

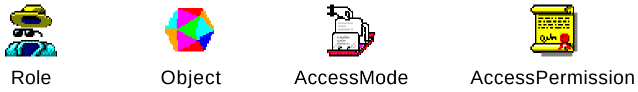


Figure 4.2: Classes of the RO level

These classes correspond to and convey the same semantics of the equally named entities in the RBAC terminology (see [97]). It should be noted that we represent RBAC *Permissions* by a pair of an *Object* and an *AccessMode* (similarly to the decomposition of *privileges* into *operations* and *objects* adopted by [33]). The many-to-many association between roles and permissions is thus established by means of an *AccessPermission* and its associations with the corresponding objects (*Role*, *Object* and *AccessMode*).

The second level (SR in fig. 4.1) offers a system view defined on the basis of the services that will be provided; its set of classes is shown at the bottom of Fig. 7.4. Objects of these classes represent: (a) people working in the modelled environment (*User*); (b) possible subjects acting on the user's behalf (*SubjectTypes*); (c) services in the network that are used to access resources

(*Services*); (d) the dependency of a service on other services (*ServiceDependency*); and also (e) *Resources* in the network. The iconic representation used for each of these classes is shown in Fig. 4.3.

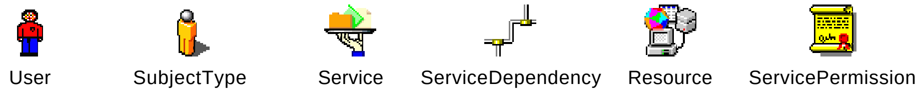


Figure 4.3: Classes of the SR level

Therefore, as regards the modelling of users, for each *Role* in the RO level related objects of the classes *User* and *SubjectType* are defined in the SR level. Both terms *users* and *subjects* refer to the equally named concepts of the RBAC terminology, and thus serve to complete the RO level model in the sense of the reference model $RBAC_0$ [97]. Note that for the sake of simplicity this work does not include more advanced RBAC features such as constraints or role hierarchies. These could be easily incorporated though, as shown in other work on MBM such as [68].

The modelling of subjects (also named sessions in the RBAC literature [97]) adopted in this work is somewhat particular. Traditionally, a subject is an automated entity in the system that act on a user's behalf, thus representing a particular session in which a set of roles is simultaneously activated [97]. The MBM approach does not cover the role activation model of RBAC, but rather only its prescriptive, static part. Nevertheless, the class *SubjectType* is used to define the *basic types* of subjects that might take place in run-time, and for this reason the suffix "type" is appended to the class name (for a further elaboration on this topic see [68]). A *SubjectType* object is thus used to represent typical settings in which the user can act in the system, e.g. "@office" or "@home", representing the subjects that are activated by the user at the office or resp. at home.

4.1.2 The DAS level

Concept of Abstract Subsystem (AS)

An *Abstract Subsystem* (AS) is an abstract view of a system segment; i.e. a simplified representation of a given group of system components. As such, an AS omits much of the detail that is contained inside of it, presenting instead only the relevant aspects for a global view of the system structure. These aspects are chosen based on a policy-oriented view of the system, which is depicted in Fig. 4.4.

In this scheme, three types of elements can be distinguished: *actors*, *mediators* and *targets*. The first type (*actors*) stands for groups of individuals in a system which have an *active* behaviour; i.e. they initiate communication and execute mandatory operations according to *obligation policies*.

The second element type encloses *Mediators*, which intermediate communications, receiving requests, inspecting traffic, filtering and/or transforming the data flow according to the *authorisation policies*. They can also perform mandatory operations based on *obligation policies*, such

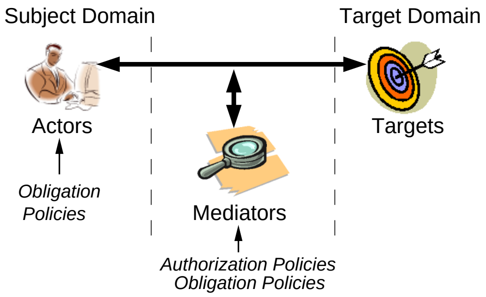


Figure 4.4: Components of Abstract Subsystems

as registering information about data flows. The *Targets*, in turn, are *passive* elements; they contain relevant information, which is accessed by *actors*.

Using this scheme as a foundation, we can now redefine an *Abstract Subsystem* as a non-directed graph whose edges represent potential (bidirectional) communication between its nodes. These nodes can be either of one of the three types mentioned above (*actors*, *mediators* and *targets*) or *connectors*. This last type of component has been added to represent the interfaces of one AS with another; i.e. to allow information to flow from, and to, an AS.

Diagram of Abstract Subsystems (DAS)

Relying upon the concepts presented in the preceding section, we now introduce a new abstraction level into the modelling of security systems: the *Diagram of Abstract Subsystems* (DAS). This layer is located immediately below the service-oriented view of the system (SR level in Fig. 2.1) and above the PH layer, which depicts the actual network mechanisms. Its main objective is to describe the *overall structure* of the system to be managed in a modular fashion; i.e. to cast the system into its constituent blocks (ASs) and to indicate the connections between them. Since these blocks consist of a policy-oriented, abstract representation of the actual subsystem components (see Sect. 4.1.2), the DAS provides a concise and intelligible view of the system architecture—in a similar sense as the one proposed in [85].

The DAS is formally a graph, comprised of ASs as nodes and edges which represent the possibility of bidirectional communication between two ASs, as shown at the bottom of Fig. 4.8. Thus, from a top-down perspective (i.e. from the SR level downwards) this diagram adds four kinds of information to the precedent model:

1. the segmentation of the users, services and resources into subsystems;

2. the structural connections amongst elements and subsystems;

3. the structural connections amongst the different participants of a policy (actors, mediators and targets);

4. mediators that are not directly related to SR level services but take part in the communication and filter or transform its data (e.g. firewalls).

On the other hand, from a bottom-up point of view, DAS substitutes the actual network of mechanisms, giving an abstract and clear view of the system architecture, which is more scalable and policy-oriented. Indeed, this diagram supports the reasoning about the structure of the system *vis-à-vis* the executable security policies, thus making explicit the distribution of the different participants of these policies over the system.

4.1.3 Security Goals, Requirements and Assumptions

In order to provide the model with more fine-grained information about accesses that the system must allow, the modelling encloses an additional type of security policies: *abstract security goals*[1]. The goals are defined at the RO level by *SecurityGoal* objects (Fig. 7.4). They extend the RBAC model by abstractly representing the security properties that are required to access an object or to perform a given access mode. The modeller thus defines a *SecurityGoal* by attaching it a label (e.g. "Top Confidential" or "Mission Critic, 4x7 availability needed"), and connecting *Objects* and *AccessModes* to the goal. In this manner, security goals complement the authorisation policies expressed by *AccessPermissions* – which represent *what* must be allowed – with qualitative information about *how* accesses must be performed (following thus the definition of policy goals in [113]).

At the SR level, each *SecurityGoal* is assigned to a *SecurityRequirement.* This work follows the standardised definition according to which a requirement is a description of a system service or constraint needed to achieve a goal [48]. Thus, a *SecurityRequirement* details the security properties that must be fulfilled to achieve the corresponding *SecurityGoal.* These properties are expressed by a vector of *security levels* (natural numbers ranging from 1 to 4) with respect to four categories: *confidentiality*, *integrity*, *availability*, and *accountability.* These categories comprise a representative subset of the most common functional security requirements as stated in standards like [22, 54] and are not intended to be exhaustive. Each 4-tuple of security level values constitutes a *security class* (for instance, $(1, 1, 1, 1)$ is the lowest possible class). Hence, a *SecurityRequirement* specifies how a *SecurityGoal* should be accomplished by the system in terms of the lowest security class that must be enforced for the related objects and access modes.

On the other hand, a model includes an additional type of objects in order to express the security properties that an entity is assumed to *assure*: the *SecurityAssumptions* (see Fig. 7.4). Similarly to the *SecurityRequirements*, *SecurityAssumptions* are also represented by a security class. At the SR level, *SecurityAssumptions* are associated to each *Service* in order to represent the security class that the service provides; i.e. the security levels one can assume in a communication that involves that service.

As for the DAS level, *SecurityAssumptions* are assigned to each AS in order to express the security class that the elements inside an AS are assumed to share. This makes possible, for instance, to model that one should expect higher confidentiality level of an internal network than of a public network. Additionally, a modeller may assign a *SecurityAssumption* to individual

[1] The concepts of security goals, requirements and assumptions elaborated in this section have evolved from the *security vectors* proposed in [94].

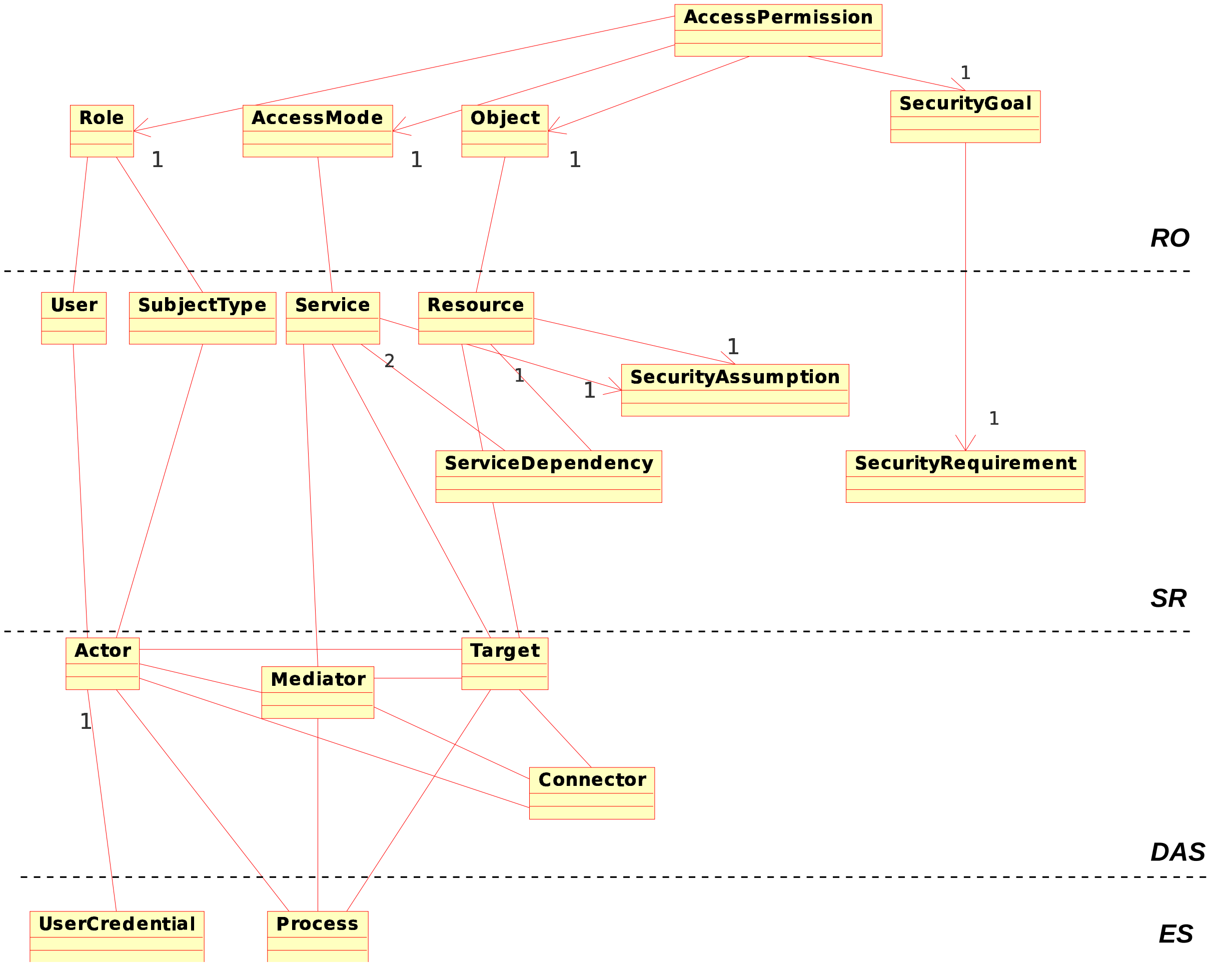


Figure 4.5: Class Diagram of Levels RO, SR, DAS, and ES

Actors, *Targets*, or *Mediators*. The particular object is then assumed to have different security properties than that of the other elements in the subsystem. Thus, as long as a given DAS object does not have a directly associated *SecurityAssumption*, it is the assumed to warrant the security class of the corresponding AS.

A class diagram encompassing the levels RO, SR, DAS, and ES is shown in Fig. 4.5. It presents the main classes mentioned above and their reciprocal relations (for clearness, all the multiplicities of the type "to many" are hidden in the associations of the figure).

4.1.4 Expanded Subsystems

Additionally, each AS in a DAS is also associated with a detailed view of the system's actual mechanisms. This expanded view is called *Expanded Subsystem* (ES) and encompasses classes of objects that represent:

- computers of the system (or *hosts*);

- *credentials* of the users, like login names or certificates;
- *processes* that take part in the communication corresponding to the policies;
- *system objects* that are manipulated by processes, e.g. data files;
- network connection entities, such as protocols, interfaces, and network segments.

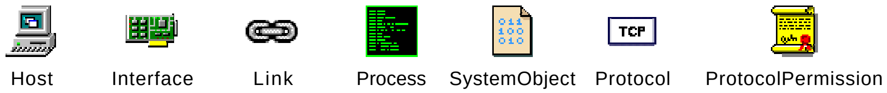


Figure 4.6: Classes of the ES level

The pictorial representations of some of these basic classes are shown in Figure 4.6. These classes are used to define both the components of the system itself and the security mechanisms employed to control the activity of the former. For the latter, special process classes are also defined to handle each specific mechanism type supported.

Notice that the ES representation has a static character; i.e. it is a picture of the system components and connections, but it does not include a behavioural description of their interactions. Therefore, the modelling of processes is somewhat particular. A process object in the ES level actually stands for a *prototype* of processes that might occur in the real environment. One should thus see a process object not as a picture of a particular process executing in the real world, but rather as an abstraction that holds the relevant common properties of all similar processes that can be launched on a certain host.

4.2 Usage of the Modelling Framework

In order to make clear the modelling technique previously presented, this section presents the modelling usage in a concrete network environment. Each step of the process is exemplified by means of a test scenario, following described.

4.2.1 Scenario Description

The considered scenario is that of an enterprise network that is connected to the Internet, as illustrated in Fig. 4.7. Our main goal is to design the configuration for the security mechanisms that are required to enable and control web-surfing and e-mail facilities for the company's office employees. The employees are allowed to use the computers in the office in order to surf on the Internet and to access their internal e-mail accounts, as well as to send e-mails to internal and external addresses. As for the ongoing communication from the Internet, the security system must enable any external user to access the corporate's web site and to send e-mails to the internal e-mail accounts.

Therefore, the highest-level security policies for this environment can be stated as follows:

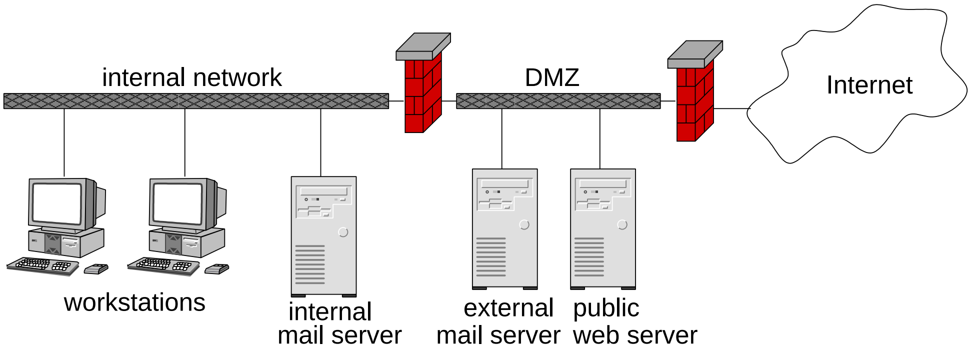


Figure 4.7: Sample Network Scenario

P1: The employees may surf on the Internet from the computers in the office;

P2: The employees may read their internal e-mails from the office;

P3: The employees may send e-mail both to external and internal addresses;

P4: Users on the Internet may access the corporate web server;

P5: Users on the Internet may send e-mail to internal company's mail addresses.

The three-layered model for this environment is shown in Fig. 4.8. We describe the main elements of the model layers SR and RO and their mutual relations in the next section. Thereafter, the modelling of the DAS level is covered in detail in Sect. 4.2.4.

4.2.2 Modelling the RO Level

Since the highest level in our model is based on RBAC concepts, the designer starts the development process by mapping the abstract policies, expressed in natural language, to the more formal syntax of RBAC. Hence, each of the five policy statements of the previous section must be expressed by objects of the types *Roles*, *Objects*, *AccessModes* and *AccessPermissions*, and their relationships.

The top of Figure 4.8 shows the resulting model at the RO level for our considered scenario. This level contains the following basic objects: the *Roles* "Employee" and "Anonymous Internet User", and the *Objects* "Internal e-mail", "Website", "Internet e-mail" and "Internet WWW". These objects are associated with *AccesModes* by means of five *AccessPermissions* (at the top, on the right of Fig. 4.8), each of the latter corresponding to one of the abstract policy statements. Thus, for instance, the *AccessPermission* "allow Internet surfing" models the policy statement **P1**, associating the role "Employee" to "surfing" and "Internet WWW". The other policy statements are analogously modelled by the remaining *AccessPermissions*.

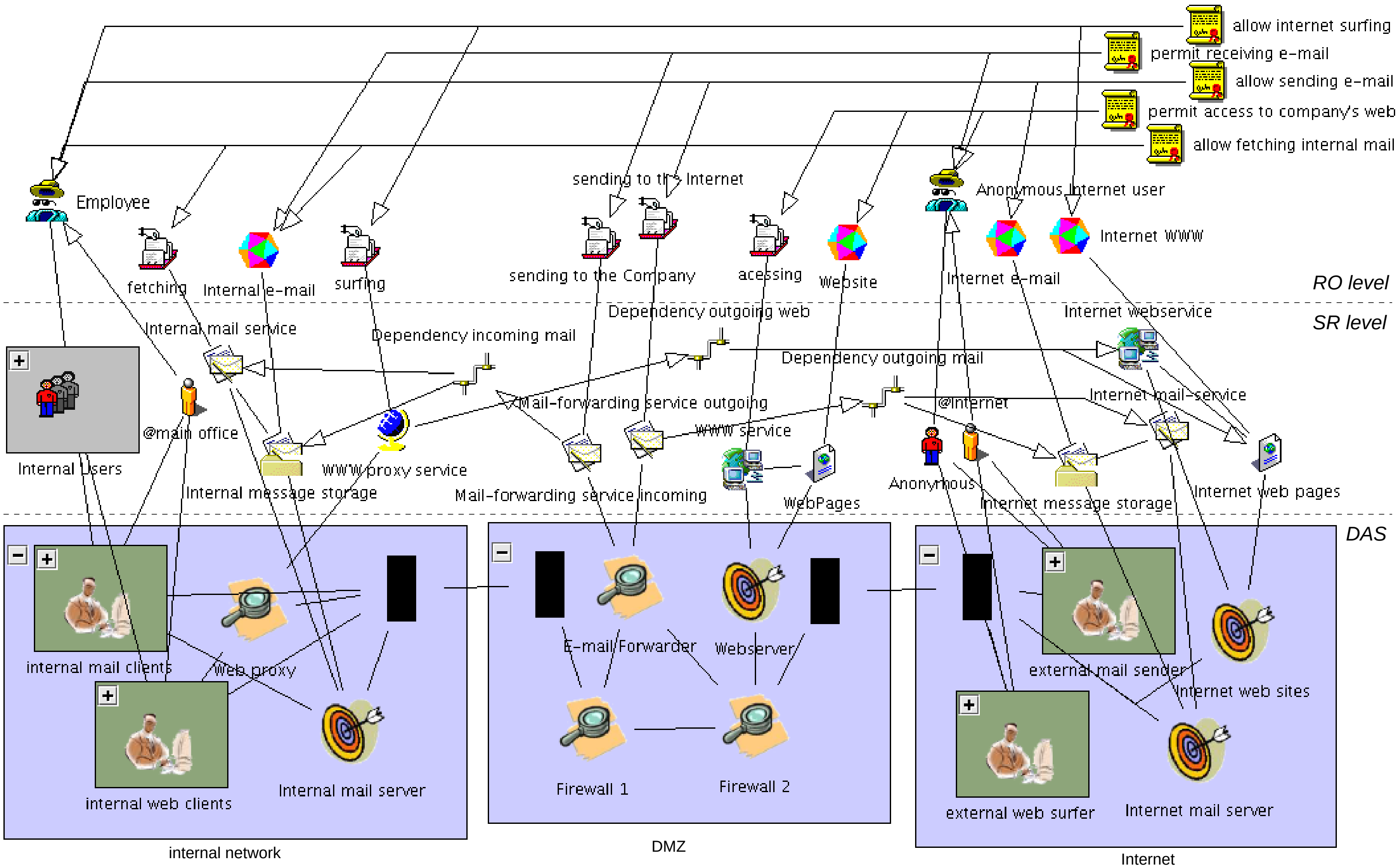


Figure 4.8: Model Example

4.2.3 Modelling of Users, Services and Resources

The second step in the design process consists of the definition of the services that the system must provide, the resources they need, and the users who may take advantage of them. As regards the modelling of users, for each *Role* in the RO level related objects of the classes *User* and *SubjectType* must be defined in the SR level. These two classes add information about each user that is authorized to act in a certain role and the possible subjects that may take place in the system.

For the test scenario, the *User* "Anonymous" and the *SubjectType* "@Internet" are defined in association with the role "Anonymous Internet User". For the role "Employee", several *User* objects are grouped in the *TypedFolder*[2] "Internal Users", and the corresponding *SubjectType* is "@main office".

As for the modelling of services and resources, a *Service* object will be basically defined for each *AccessMode* in the RO level, whereas an RO *Object* will be mapped to a *Resource*. In case where more than one service is needed to provide access to a resource, this fact is expressed by a *ServiceDependency*. This is what happens in our example model, for instance, with the *AccessMode* "sending to the company" that is related to the *Service* "Mail-forwarding service incoming e-mail". This service must rely on the "Internal mail service" in order to provide access to the resource "Internal message store" (which is associated with the RO *Object* "Internal e-mail"). This fact is then represented by the connection of the two services to "Service dependency association incoming e-mail". On the other hand, a quite simple situation occurs with the *AccessMode* "accessing" and the *Object* "Website" (RO level); in these cases the service "WWW Service" and the resource "Web pages", which are correspondingly associated to those objects, are directly connected to each other.

4.2.4 DAS Modelling

The mapping from the SR level to the *Diagram of Abstract Subsystems* (DAS) involves three steps: (i) the modularisation of the system in conformance with the respective network scenario—i.e. the segmentation into *Abstract Subsystems* (ASs); (ii) the mapping of the elements of the SR level (users, services, resources) to components inside each AS; and (iii) the establishment of structural connections in the DAS, reflecting the associations between elements inside an AS and those between ASs, which are performed by means of *Connector* objects (Sect. 4.1.2). Subsequently, each AS can be expanded independently in order to achieve at the PH level a detailed representation of its mechanisms, and thereby enabling the generation of the corresponding configuration files. These steps will be described in turn in the following sections.

[2] Typed folders are, as the name suggests, entities that simply group a number of elements of the same type, in order to offer a more compact representation. Further details can be found in [37].

Segmentation into ASs

The subdivision of the system into ASs shall be guided by the structural blocks of the analysed environment. The abstract components of a DAS are thus aggregated according to the groups of mechanisms that already exist in the real system, such as departments, workplaces and functions.

An important criterion to be considered is the semantic unity of an AS; i.e. the group of mechanisms enclosed in an AS must have a common property that is clearly distinguishable. As such, this property assures the *cohesion* of the AS, so that it thereby represents a logical grouping identifiable in the real environment (like the ones previously mentioned), instead of only consisting of a mere agglomeration of heterogeneous elements.

On the other hand, the modularisation criterion of *coupling* (also a classical measure in the modularisation techniques of software engineering) should be taken into account in this context as well. The more the elements enclosed in the detailed view of an AS are related exclusively to elements of the same AS (and hence more independent from elements of other ASs), the more concise will be the abstract representation offered by the DAS, since those internal connections will be hidden. In this manner, a lower coupling between ASs improve the scalability of the DAS.

Analysing the scenario illustrated in Fig. 4.7, the existence of a structural subdivision in three segments is clear, namely: the internal network, the demilitarised zone (DMZ) and the external network (the Internet). Therefore, the DAS for this example has an AS to each one of these segments.

Mapping of actors

An *actor* will be basically created for each group of hosts/processes (inside a given AS) which originates communication in order to act in conformance with a policy—which at the SR level is called service permission. In this manner, the actor corresponds to the *subject domain* of this permission, or, more precisely, to the part of the domain that is located in a given AS.

Nevertheless, actors can be shared by a number of different service permissions, as long as they have the subject domain comprehended by the actor in common. This contributes to a more compact representation, thus improving the scalability of the model.

In the SR level, the subject domain of service permissions is represented by *User* and *SubjectType* objects; thus each *Actor* will be connected to one or more pairs of these types of objects. In the framework of model-based management, however, service permissions are not directly modelled by the system designer but rather are automatically generated from *AccessPermission* objects located in the uppermost (RO) level. For this reason, the determination of the system's actors must start from an *AccessPermission*; thus taking the *Role* that is associated with it and identifying its corresponding *User* and *SubjectType* objects (in the SR level). Subsequently, one can create an *Actor* object that will contemplate the relevant *AccessPermission* and, consequently, is also related to the service permission that will be generated from it.

Considering our test scenario, an *Actor* object "internal web clients" is created in the AS "internal network" for the first policy—which is modelled by the first *AccessPermission* in

Fig. 4.8, namely “allow Internet surfing” (AP1). Similarly, for the *AccessPermission* AP3 (“allow sending e-mail”) the *Actor* “internal mail clients” is created in the same AS, whereas, in the AS “Internet”, the actors “external mail sender” and “external web surfer” correspond respectively to the objects “permit receiving e-mail” (AP2) and “permit access to own web” (AP4). The *AccessPermission* “allow fetching e-mail” (AP5) can be covered by the previously created *Actor* “internal mail clients”, since its subject domain is the same as that of AP3.

Mapping of mediators

As regards to *Mediators*, two types can be distinguished. *Mediators* of the first type are a refinement from services which perform the “middleman functions” described in Sect. 4.1.2, for instance, proxies and mail forwarders. Therefore, they are achieved by means of a straightforward mapping from those services of the SR level, positioning them in the appropriate AS. Indeed, a service can be covered by a number of *Mediators*, each one residing in a different AS. On the other hand, a given *Mediator* object can map more than one service.

In our sample scenario, the “E-mail-Forwarder” *Mediator*, in the “DMZ” AS, stands for both services of handling incoming and outgoing e-mails. As for the “internal network”, the “Web proxy” *Mediator* maps the “WWWProxyService”.

The second type of *Mediators* consists of technical mechanisms that are not modelled in the SR level but are required in order to control the communication; i.e. they transform and/or filter it according to authorisation policies (like packet filters, IP-masquerading), or inspect the data according to obligation policies (e.g. IDS, event monitors). In this manner, the system designer shall create this type of mediators whenever these functionalities are required; i.e. a *Mediator* object will then appear wherever a security mechanism like the previously mentioned ones is to be placed in the respective actual network environment.

Examples of the second type of *Mediators* are illustrated in Fig. 4.8 by the objects “Firewall 1” e “Firewall 2”. They have been introduced into the AS “DMZ”, precisely in the place where the firewalls are found in the scenario of Fig. 4.7.

Mapping of targets

Targets are obtained by a quite direct mapping from the pairs of *Service* and *Resource* objects (in the SR level) which encompass a target domain of a service permission, or a part of this domain that is placed in a given AS. In this way, each *Target* object must be connected to at least one pair of *Service* and *Resource* in the SR level, but it can also be shared by different service permissions; in the latter case, relations with other such pairs would be also present—this sharing also contributes to the conciseness of the model. Conversely, each pair of *Service* and *Resource* can be mapped to a number of *Targets*, each one located in different ASs—similar to the case of *Mediators*.

Similar to the actors, the target identification must start by considering the *AccessPermissions* in the RO level. Here, nonetheless, it is the *Object* related to a certain *AccessPermission* that is considered at first in order to establish then the corresponding *Resource* (SR level) and the *Service* which provides access to it. Finally, we create a *Target* to map this pair of objects.

When applying this method to our test scenario, then for the policy P1 (Sect. 4.2) the *Target* object "Internet web sites" is created to refine the pair of *Service* and *Resource* of "Internet webservice" and "Internet web pages", since the latter is related to the *Object* "Internet WWW" (at the level RO) of AP1. It is worthwhile to note that, in this case, the *Service* refined from the *AccessMode* "surfing" of AP1 , namely "WWW proxy service", is different from the one previously used as target; this only happens when there is a *ServiceDependency* between these two *Services*. Indeed, the "WWW proxy service" cannot provide access to the *Resource* "Internet web pages" by itself, but relies on the "Internet web service" to do it. This dependency is modelled by the object "Dependency outgoing web" in the SR level (at the centre of Fig. 4.8).

Proceeding in the same manner, the targets "internal mail server" ("internal network"), "Internet mail server" ("Internet") and "Web server" ("DMZ") map respectively the policies modelled by AP2, AP3 and AP4. As for AP5, the *Target* "internal mail server" can be shared with AP2; as such, there is no need to create another object.

Establishment of Structural Connections

In order to complete the model that has been formed thus far by the application of the procedures of the latter sections, one needs to introduce the associations between objects of the DAS; i.e. formally speaking, to add the edges of the graph. Such associations have a different meaning compared to that of the associations between an element in the SR level and an object of the DAS. While the latter represent abstraction refinements—in the sense of relating levels of a policy hierarchy—the former represent structural connections; i.e. the possibility of communication in the actual system. Despite this, only the connections that are relevant to the abstract view of the system shall be depicted here; these are namely the associations that interconnect the different participants of executable policies: actors, mediators and targets.

Once again, the establishment of these connections starts at an *AccessPermission*. Each of the objects (in the DAS) that correspond to this permission is identified and then associations are created in order to construct paths between the respective actors and targets, traversing the necessary mediators. Whenever one of these paths enters or leaves an AS, *Connector* objects are inserted at this point, representing the communication interfaces of the AS. Thus, the number of *Connectors* in an AS corresponds to the number of available physical interfaces of the actual system.

Proceeding in this manner with our test scenario, the *Connector* objects (rectangles) and connection edges (lines) shown in Fig. 4.8 are obtained.

4.2.5 Expanded Subsystems

Starting from the model that has been produced thus far (Fig. 4.8), the next stage in the model development is to expand each of the ASs separately. This means that, for each AS, the mechanisms inside it shall be modelled according to the usual procedure described in [37], resulting in a detailed representation of these mechanisms; i.e. the PH level. Afterwards, the associations between the PH level components with the objects in the AS have to be drawn, thus establishing a relation of abstraction refinement.

Each *Actor* object of an AS must be then related to its corresponding *Process*- and *UserID*-typed components in the PH level, such that the *Actor* performs the association of these components with *SubjectType* and *User* objects in the SR level. As noted in Sect. 4.2.4, an *Actor* may be used for more than one pair of *SubjectType* and *User*, thereby corresponding to several service permissions. Hence, to avoid the burden of depicting all of the single associations amongst the objects (of type *User*, *SubjectType*, *UserID* and *Process*) connected to each *Actor*, a table of 4-tuples containing these associations shall be used to store them.

The *Mediator* objects of an AS, in turn, are simply related to one or more *Process*-typed components which implement the corresponding functionalities. With regards to *Targets*, each of them is related to one or more pairs of *Processes* and *Objects* that provide the corresponding services. Therefore, a *Process* in the PH level is related via a *Target* to a *Service* in the SR level, whereas an *Object* is related in a similar fashion to a *Resource*.

Figure 4.9 shows the ESs corresponding to the ASs "internal network" and "dmz" (right). The relation from actors, mediators and targets in the abstract representation AS (at the top) to objects in the detailed view ES (bottom) is graphically indicated by edges. For instance, the *Actors* "internal mail clients" and "internal web clients" in the "internal network" (on the left-hand side) are related to objects to represent the corresponding processes that run in two different workstations, as well as to the user credentials and login names of the users that may take advantage of these processes.

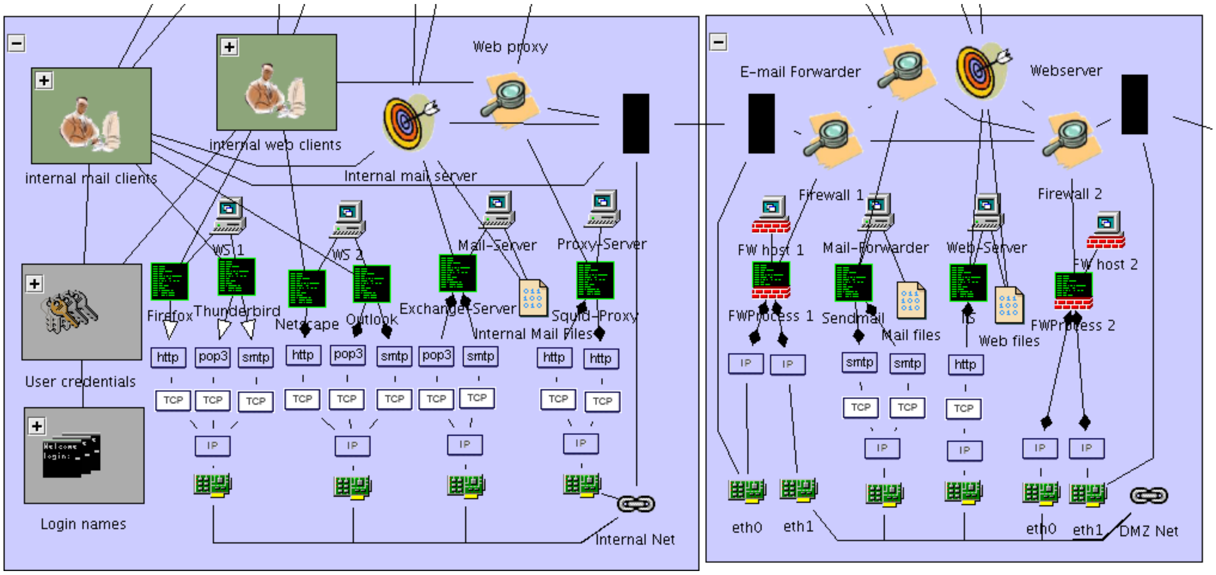


Figure 4.9: Expanded Subsystems *internal network* and *dmz* from Fig. 4.8

In the AS "DMZ" (on the right-hand side), the *Mediator* "E-mail Forwarder" and the *Target*

"Webserver" are correspondingly mapped to processes and resources that implement them. These objects in the ES that are directly assigned to AS entities are, in turn, related to a series of other objects in order to provide the model with detailed information about the communication – such as protocol stacks and the network interfaces (see bottom of Fig. 4.9). In this manner, the correspondence between the abstract view of the system (AS) and its actual mechanisms is established.

4.3 Chapter Summary

This chapter has presented a modelling technique for the management of security systems (Sect. 4.1). The modelling achieves scalability by the segmentation of the system in *Abstract Subsystems*, which enables the processes of model development and analysis to be performed in a modular fashion.

In order to clarify the semantics and the practical use of the meta-model, a top-down approach to the modelling usage is described in Sect. 4.2. The systematic mapping from a service-oriented system view to a *Diagram of Abstract Subsystems* was covered in detail, encompassing the choice of elements to be represented in the abstract view, as well as the correspondence of these elements to the actual mechanisms of the system. Each step in this processes is exemplified by means of a test scenario.

Chapter 5

Automated Refinement and Tool Support

After the modeller has achieved a valid model instance by following the principles explained in the previous chapter, the supporting tool automatically builds a policy hierarchy by deriving lower-level policy sets from the policies specified at the most abstract layer. This process is termed in the literature *policy refinement* [73, 1] or *policy transformation* [115, 113]. This work adopts the first of them. The next section elaborates on the policy support offered by the modelling framework developed and presents the automated process of policy hierarchy building.

Thereafter, Sect. 5.2 briefly describes the common supporting tool of the MBM approach, which was already existing in the beginning of the present work and was further developed both by other research groups on MBM (e.g. the SIRENA project [47, 46]) and in the context of the present work. The tool firstly assists the modelling by means of a diagram editor, and it then executes the policy refinement process. At the end of this chapter, Section 5.3 presents the *focus and context* techniques incorporated to the tool as a main contribution of the present work.

5.1 Automated Policy Hierarchy Building

The fully automated derivation of low-level, executable policies from a set of abstract specifications is, in the general case, not practical [105, 115]. Nevertheless, as our modelling is structured in different abstraction levels, the analysis of the system's objects, relationships and policies at a certain abstraction level enables the generation of lower level policies, based also on the system's model at the lower level and on the relations between entities of the two layers. As such, the model entities of a certain level and their relationships supply the contextual information needed to automatically interpret and refine the policies of the same level. At the end of this refinement process, configuration parameters are yielded for each system mechanism to be configured.

An overview on the hierarchical structure of the policies supported in our modelling is presented in Fig. 5.1. This figure emphasises the policies, so several other element types are omitted. While boxes with rounded corners represent the model entities defined by the modeller (described

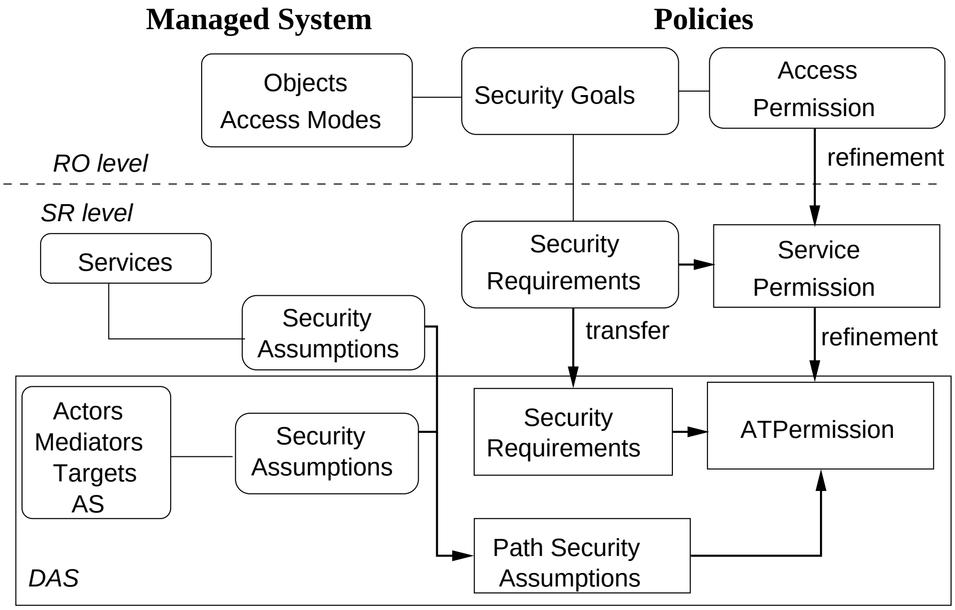


Figure 5.1: Policies and Security Requirements in the System

in Ch. 4), the objects generated during the automated refinement process are depicted as normal rectangular boxes—they will be following explained in this chapter. As for the connecting lines, the thicker, arrowed ones represent associations established automatically during the refinement process, while the thinner lines with no arrows stand for the assignments given by the modeller.

Before presenting the refinement process developed in this work, the next section analyses the policy support provided by the modelling and its semantics in comparison with a classification framework [105].

5.1.1 Policy Support and Semantics

According to Sloman and Lupu [105], policies can be sorted into two basic types: *authorisation* and *obligation policies*. Authorisation policies are used to define access rights for a subject (management agent, user, or role) and can be either *positive* (defining the actions subjects are *permitted* to perform on target objects) or *negative* (specifying the actions subjects are *forbidden* to perform on target objects). As such, authorisation policies are used to define access control rules implemented by several types of mechanisms in a network security system, such as packet filters, Kerberos, and VPNs.

Obligation policies are, in turn, event-triggered *condition-action* rules that can be used to define the activities subjects (human or automated manager components) must perform on objects in the target domain; i.e. the *duties* of these subjects. In the network security context, obligation policies can be used to specify the behaviour of mechanisms such as logging agents, intrusion detection systems (IDS) and watchdogs.

In the right hand columns of Fig. 5.1, one can notice that the policy refinement is subdivided in two parallel tracks, corresponding to the two policy types supported by the modelling of this work: *security goals and requirements*, and *authorisation policies*. In the uppermost model level (RO), authorisation policies are represented by means of *AccessPermission* objects (Sect. 4.1.1). The set of *AccessPermissions* is given by the modeller and acquires in this context a particular meaning. On the one hand it defines the explicit permission for *Roles* to access *Objects* (in the way defined by an *AccessMode*) – corresponding to positive authorisation policies. On the other hand, MBM adopts closed policies [98]; i.e. the default decision of the reference monitor is denial. This implies that all triples of *Role*, *Object* and *AccessMode* not belonging to the set of *AccessPermissions* are forbidden. They thus implicitly define the negative authorisation policies which the security mechanisms must as well enforce.

Moreover, as a particularity of the MBM approach that differentiates it from traditional access control models, the high-level policies and system model additionally represent features that the system to be managed *must* implement. As such, the positive and negative *authorisation policies* for users—i.e. the *user privileges*, or things that users *may* do or not—are also to be interpreted as *obligation policies* for the system—i.e. the *system duties* or things that the system *must* support (allow) or not (forbid). All of these different connotations of policies at the highest level must be propagated to the inferior levels by the policy refinement process.

As for explicitly defined obligation policies, these are not represented in MBM since the modelling used in MBM builds upon RBAC (Sect. 2.4), which in its basic form does not enclose obligation policies (referred to as duties in [97]). However, besides the above policy elements, our modelling framework also includes an extension to the RBAC model by means of the classes *SecurityGoals*, *SecurityRequirements* and *SecurityAssumptions* (Sect. 4.1.3). During the process of policy refinement described in the sequence, the security levels warranted by mechanisms (*SecurityAssumptions*) are thus checked against the security requirements prescribed in the *SecurityRequirements*. As such, a *SecurityRequirement* with a high level of confidentiality could determine, for instance, the decision between using an encrypted tunnel instead of plain text. Another practical example occurs when a security mechanism that supports logging has this functionality activated in order to fulfil the high traceability level required by the prescriptive *SecurityRequirement* associated to the *SecurityGoal* of its corresponding *AccessPermission*. The examples show that, though *SecurityRequirements* do not directly represent obligation policies, the analysis of the security requirements they express can yield configuration parameters for mechanisms that do correspond to the “need to do” aspects in the system, and hence to obligation policies.

The following sections describe the step-wise refinement process for each model level in turn.

5.1.2 Refinement RO/SR

The automated refinement of authorisation policies starts from the analysis of the *AccessPermissions* in the RO level and their related objects in order to generate a set of corresponding permissions in the SR level. Thus, each triple of *Role*, *AccessMode* and *Object* (r, am, o) related to an *AccessPermission* produces a set of 4-tuples (u, st, sv, r), each of which expresses an authorisation for a *SubjectType* on behalf of a *User* to use a *Service* in order to access a *Resource*. These tuples are represented by *ServicePermission* objects (see Fig. 5.1).

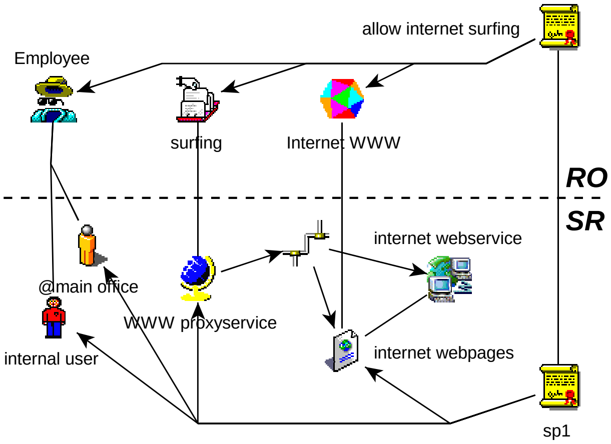


Figure 5.2: Example of Refinement $RO \rightarrow SR$

A refinement example is shown in Fig. 5.2. The *AccessPermission* “allow internet surfing” is refined into the *ServicePermission* “sp1” that associates the objects “internal user”, “@main office”, “WWW proxyservice”, and “internet webpages”. Each of these elements refine the RO-level objects that are related to the permission “allow internet surfing”, namely: “Employee”, “surfing”, and “Internet WWW”, respectively.

In addition, a security requirement is also calculated for each generated *ServicePermission*. This is accomplished by comparing the security classes assigned to the *SecurityGoals* of *Object* and *AccessMode* that participate in the *AccessPermission* being refined (as Fig. 5.1 illustrates). The *SecurityRequirement* results from choosing the maximum level for each vector dimension between the values of the two security classes. For instance, if an *Object* has a security goal assigned to the class $(3, 3, 2, 1)$, and the *AccessMode* has a security goal assigned to $(3, 1, 4, 2)$, the resulting *SecurityRequirement* has the class $(3, 3, 4, 2)$.

5.1.3 Refinement SR/DAS

Subsequently, the *ServicePermissions* are refined into *ATPermission* objects (actor-target permissions, ATP for short), which represent authorisation policies in a DAS (Sect. 4.1.2). Since ATPs are paths in the DAS graph, this refinement phase consists of, for each *ServicePermission* (u, st, sv, r), finding the shortest path between each *Actor* that is connected to the pair (u, st), and each compatible *Target* that is connected to a pair like (sv_t, res). If the service sv has direct access to the resource r, then sv_t—i.e. the service related to the *Target*—and sv will be the same. Otherwise, sv must rely upon a dependency chain that leads to sv_t, in order to access r. In this case, to each service that takes part in the dependency chain, a corresponding mediator must be found along the path between *Actor* and *Target*.

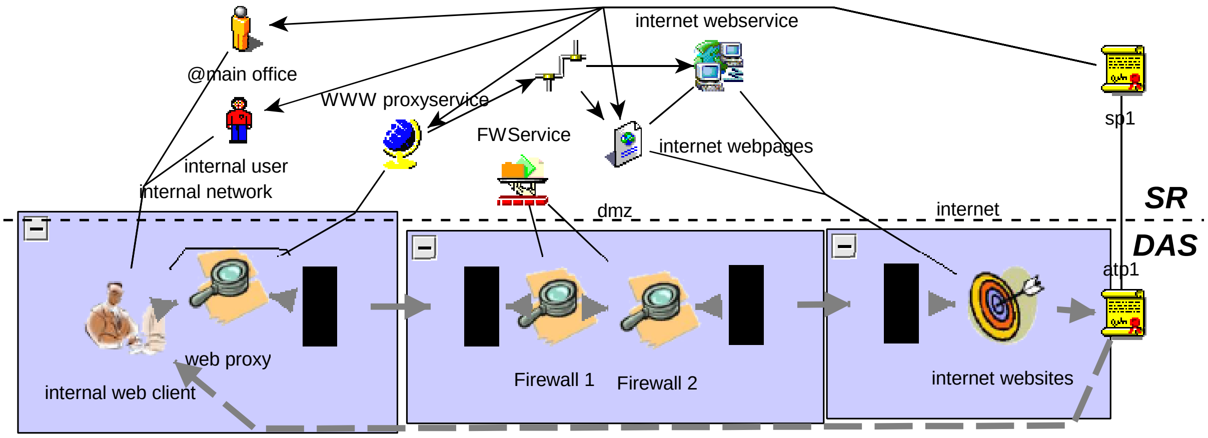


Figure 5.3: Example of Refinement $SR \rightarrow DAS$

A visualisable example for this refinement step is given in Fig. 5.3 (it is in fact a continuation of the refinement in the previous section). For the *ServicePermission* "sp1' ' (top, at the right hand side) the refinement algorithm determines the ATP "atp1" that starts in the *Actor* "internal web clients" (this is indicated by the gray, dotted line from the ATP to the *Actor*) that is connect to the "internal user" and "@main office". The path traverses the *Mediators* "web proxy", "Firewall 1" and "Firewall 2" (as well as some connectors along the way), and ends in the *Target* "internet web sites" (indicated by the dotted line from the latter to the ATP) that is connected to the resource "internet webpages". Notice that the mediator "web proxy" that refines the "WWW proxy service" must be in the path, since there is a dependency chain for this service to reach the resource "internet web pages"; otherwise, "atp1" could use the shortest path that goes from "internal web clients" through the connectors directly to "Firewall 1".

During the path discovery, the refinement algorithm also checks the security assumption of the path against its security requirement, which in turn comes from the requirement that was determined for the *ServicePermission* in the preceding step (as illustrated in Fig. 5.1). The *path security assumption* reflects security levels that can be provided by all the elements along the path, and it is calculated with basis on the assumptions of the individual objects and the services that take part in the communication path, as illustrated in Fig. 5.1 (this topic is further elaborated later on in Sect. 6.3.1). Only paths that comply with the requirements become ATPs; thus, if the tool cannot find such a valid path to refine a *ServicePermission*, an error message is presented to the user, so she/he can modify the system model to make it congruous to the policies.

5.1.4 Refinement DAS/ES

The following refinement phase comprises the automated generation of policies that consider the equipments and security mechanisms defined in the ESs. For this purpose, the tool generates for each ATP a corresponding *Allowed Expanded Path* (AEP) that represents an authorised path in the expanded subsystem views. Each AEP connects a process (that refines an *Actor*) to other processes (that refine the *Mediators* and the *Target*) through their related protocol stack, interface, and network objects. Since the path discovery was already accomplished in the previous refinement step, the refinement algorithm DAS/ES is quite simple. It just expands the ATPs according to the related objects in the detailed view of the ESs.

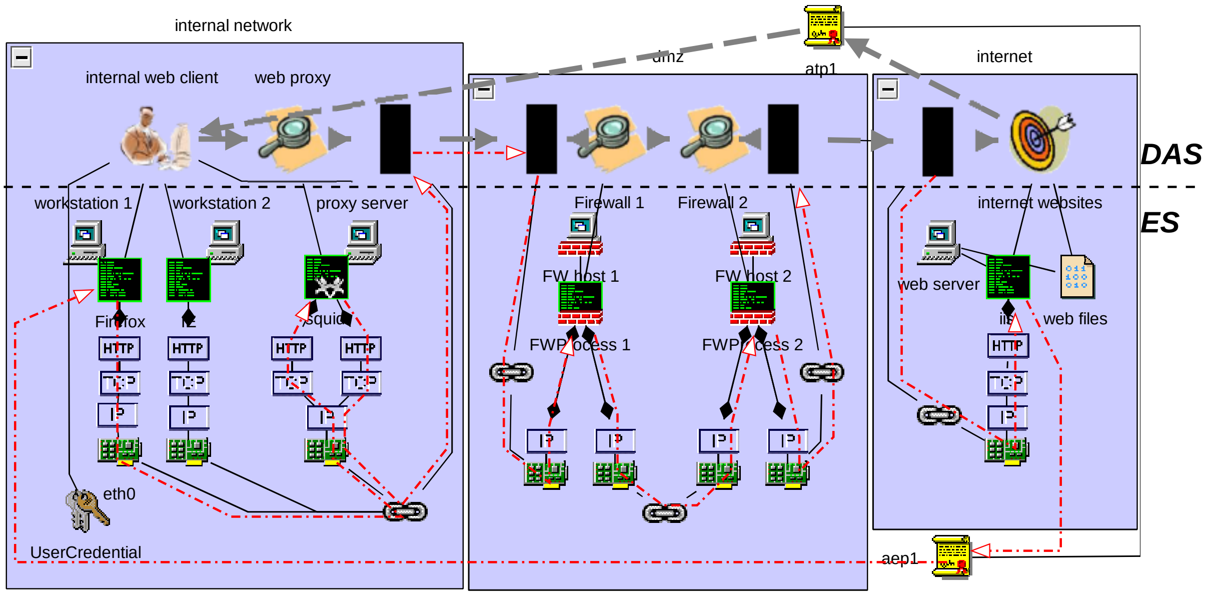


Figure 5.4: Example of Refinement $DAS \rightarrow ES$

The ATP of the example in the previous section and one of its corresponding AEPs are shown in Fig. 5.4. The path allowed by “aep1” is marked in the figure by thinner, dotted and dashed arrows. It starts in the subsystem “internal network” with the process connected to “workstation1”—which is related to the actor “internal web client”—and goes through protocol stacks and network links up to the process in the “proxy server”—which is related to the mediator “web proxy”. From this point, the path continues through protocols and links, flows to the subsystem “dmz” traversing connectors and it then passes through two firewalls, reaching the subsystem “internet” and its end point is thus the process related to the “web server”—which is in turn associated with the target “internet websites”. For the same ATP, an additional AEP similar to “aep1” is also generated, beginning in the process related to the “workstation2”; it is omitted in Fig. 5.4.

It is worthwhile noticing that the compliance with security requirements must not be checked in this phase, since it was already verified for the abstract path (ATP) . Indeed, an AEP contains only processes that are related to the abstract entities in the corresponding ATP, whose security properties were already checked. The premise is that only processes are active elements that must be controlled, and passive objects, like protocols and links, must not be considered—a classical assumption in access control models [96] (off course, problems like covert channels are not addressed here).

5.1.5 Configuration Parameter Generation

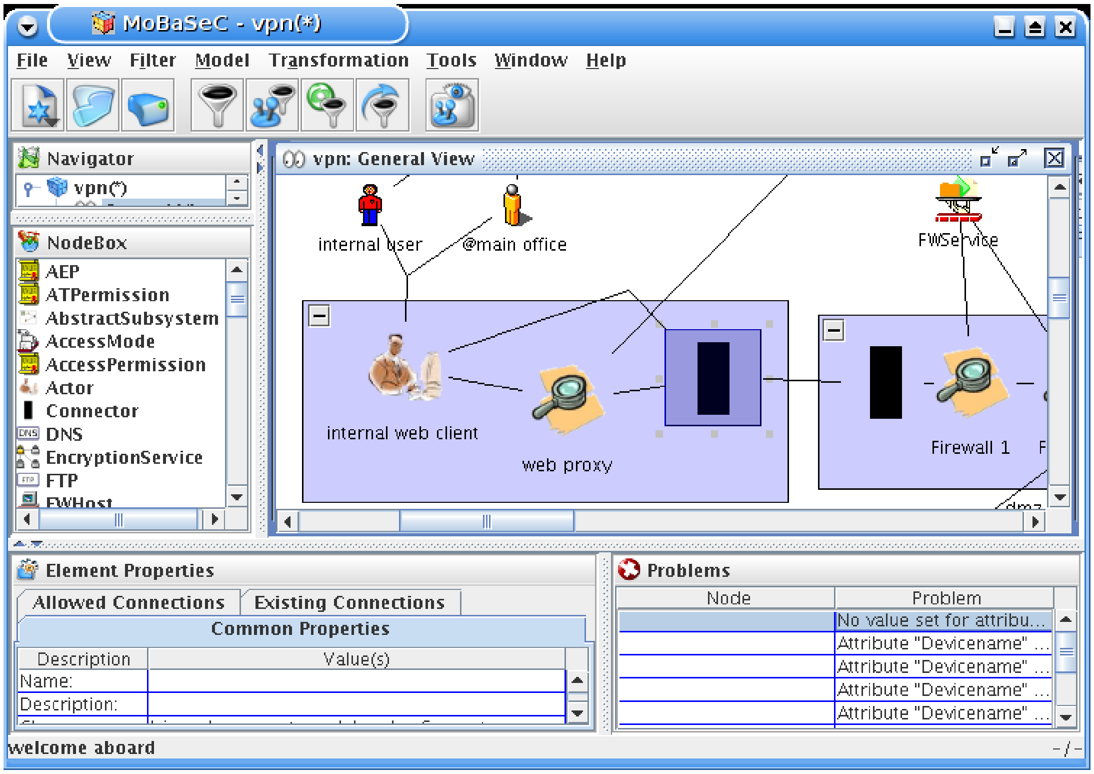
In the last phase of the refinement process, a special back-end function for each mechanism type supported is executed. It analyses the characteristics of each AEP that passes through the mechanisms of a type (such as communication protocols, addresses, and ports) to produce corresponding low-level, device-dependent configuration parameters. Clearly, the configuration for a given mechanism must allow only the accesses corresponding to the AEPs that traverses the mechanism.

For instance, considering "aep1" of the example in the previous section (Fig. 5.4), a back-end function for a specific web proxy software would analyse the path and converts it into configuration parameters for the proxy mechanism in the host "proxy server" in order to allow the access. Another specific back-end function would then analyse "aep1" and generate configuration files for the two firewalls in the "dmz" subsystem; i.e. it would produce packet-filter rules that correspond to the characteristics of the path (addresses involved, protocol, ports, connection orientation etc.). In this work, back-ends to the OpenBSD pf packet filter, and the VPN daemon isakpmd were implemented. They are presented later on in Sect. 7.1.

5.2 Supporting Tool

The Model-Based Management is supported by a generic tool called MoBaSeC (Model-Based Service Configuration). The architecture of this tool is defined in such a way, so that different applications of MBM are covered by the same common tool. The details of each application are defined by means of a meta-model, i.e. a set of classes that specify the node classes of each layer, the allowed connections between classes, and the consistency rules to which each model instance must comply. These rules can define the properties whose values must be set (i.e. mandatory properties), the allowed range for values, the minimum and/or maximum number of connections to objects of a given class, or any other consistency restriction concerning the model entities. In addition to that, the meta-model also encloses the implementation of refinement algorithms that generate lower-level policies for a model instance (as the ones described in the next section). As such, the same basic tool (MoBaSeC) can be used to support the management of an arbitrary application context, as long as there is a meta-model that defines the structure for models in that particular context.

MoBaSec is implemented in Java and basically consists of an object-oriented graph editor whose interface is shown in Fig. 5.5 (for a comprehensive explanation of the tool see [68]). To define a model instance, the user simply selects one of the classes available in the meta-model that are shown in the panel "Nodebox" (at the centre in the left hand side), and creates a new object of this class in a window that contains a view of the model instance (in the right hand side at the centre). Connections between two objects are thus established by dragging-and-dropping edges to connect the objects. Furthermore, objects have properties that can be set in a special



Description	Value(s)
Name:	
Description:	

Node	Problem
	No value set for attribu...
	Attribute "Devicename" ...
	Attribute "Devicename" ...
	Attribute "Devicename" ...
	Attribute "Devicename" ...

welcome aboard

Figure 5.5: Graphical interface of the supporting tool

panel (at the bottom in the left hand side). The property values provide both general information about the objects (e.g. names and descriptions) and, for some classes, special information that will be used in the configuration files generation, such as IP-addresses and port numbers.

Furthermore, the tool checks the consistency rules defined in the meta-model in order to ensure that the model instance is valid. The verification of basic rules is performed on-the-fly as the user inputs the model objects and properties—e.g. whether a connection between two objects is allowed. Once the model is complete, a menu item can be used to trigger a more comprehensive model checking, in which a number of general consistency rules are applied, and the problems found are listed in a separated panel (in the right hand side at the bottom of Fig. 5.5). In this manner, the user receives immediate feedback and can correct the problems before the policy refinement takes place.

MoBaSeC also relies upon the *Model-View-Controller* architectural pattern [17], according to which the model should be separated from its graphical representation. Consequently, the user might define different model views that, for instance, can filter some of the objects, so he/she is able to better visualise given parts of the model.

5.2.1 Extensions of this work

To support the modelling technique previously described, a specific meta-model has been developed for this work. This meta-model consists of a set of classes and methods that define (1) the possible model elements and their potential connection to other element types, (2) consistency checks to be applies to model instances (as described in Ch. 6), and (3) the refinement algorithms that implement the procedure described in Sect. 5.1.1. Implementation details can be found in [53] and especially in [55].

Furthermore, in order to enhance the handling of large models, the tool also incorporates the *focus and context* techniques *semantic zooming* and *fisheye views.* They will be explained in the next section.

5.3 Focus & Context

The term *focus & context* refers to techniques that allow a user to centre his view on a part of the model that is displayed in full detail (*focus*), while at the same time perceiving the wider model surroundings in a less detailed manner (*context*). According to Card *et. al* [18], they rely on three premises: a) the user needs both the overall picture and a detailed view; b) there may be different requirements for the information in the detailed view than in the overall picture; c) both displays may be dynamically combined into a single view. The major advantage of using these principles is the improved space-time efficiency for the user; i.e. the information displayed per unit screen area is more useful and, consequently, the time required to find an item of interest is reduced as it is more likely to be already displayed [76].

We employ the *focus & context* concept within two different methods. First, it is applied to the structure of compounded elements of the model, i.e. to the model entities that aggregate a group of simple objects. This application explores the exhibition of these entities in different levels of detail and is called *semantic zooming.* Second, we employ a visualisation technique that relies on a specific graphical projection of the model into the two-dimensional Euclidean space, which is called *fisheye view.* The next sections describe each of these techniques in turn.

5.3.1 Semantic Zooming

The concept of *semantic zooming* [76, 60] is based on the ability to display model objects in different abstraction levels, depending on their distance from the focus. Thus, objects inside the focused region of a diagram are exhibited in their full detailed form, whereas objects located at the borders are shown in the most simplified way. The regions between these two extremes are displayed with intermediary levels of detail. In this manner, the presented information is selectively reduced by adjusting the level of detail in each region to the user's interest in this region, a basic principle for *focus & context* techniques. However, in the particular case of semantic zooming, the different levels of detail employed are not related to graphical properties of the objects, but rather to the kind of information that they represent; i.e. to their semantics.

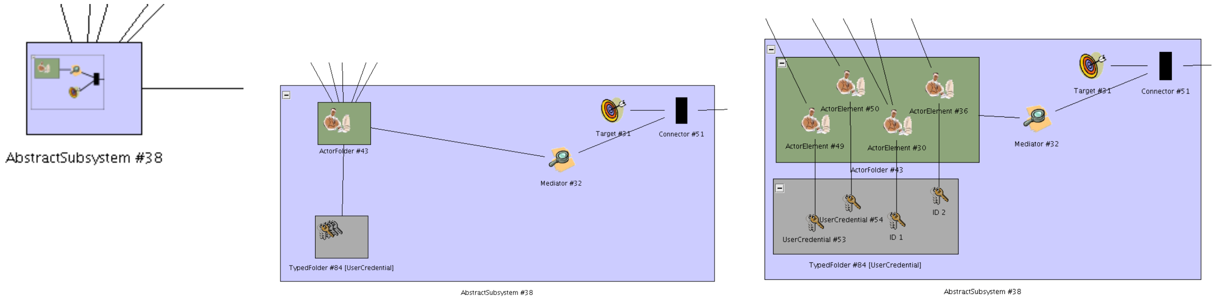


Figure 5.6: *Semantic Zooming* applied to an AS

In our context, there are two classes of compounded objects to which semantic zooming is applied: *typed folders* and *Abstract Subsystems*. A *typed folder* is an object that aggregates a group of objects of same class (or type), for the sake of the conciseness of the representation . On the other hand, *Abstract Subsystems* contain objects of various classes (see Sect. 4.1.2) and may also enclose *typed folders*. In both cases, the level of detail shown can be changed by the selective display of internal objects.

In the simplest situation, two different representations of typed folders are available: a closed (all internal objects are hidden) and an open folder view. As for the ASs, three different levels of abstraction are used: i) a full detailed view that includes all the internal objects; i.e. both the abstract and the expanded view; ii) an abstract representation encompassing only the objects pertaining to the abstract view; and iii) a "closed" view in which all internal objects are not displayed. These three different representations are shown in Figure 5.6, with decreasing abstraction levels that range from closed folder view (left) to a complete detailed view (right).

5.3.2 Fisheye View

The term *fisheye view* is used for the type of projection created by a fisheye lens used in photography. This type of lenses achieves a 180° field of view and is uncorrected. It results in an optical enlargement of objects near the centre in relation to those at the borders. This feature emulates the human visual perception, which by the effect of the eye movements has a clear focused area and a gradual loss of visual resolution in the direction of the peripheral regions. A fisheye view combines thereby a complete image overview with a gradual degradation of detail that increases with the distance from the focus—and it is thus well suited to implement the concept of *focus & context*. In contrast to semantic zooming, the fisheye view manipulates the size of the objects in order to change the amount of information displayed.

An early formalisation of the fisheye view for data visualisation is presented by Furnas [34] and a practical application for graph visualisation is offered by Sarkar and Brown [99]. The latter offers the improvement of providing a self-adaptable view with a variable radius (r_{max}) for the focused area to be enlarged. In contrast to a view with fixed radius, the variable radius is defined dynamically as the distance between the focal point and the image borders. Figure 5.7 shows the advantage of using the technique of variable radius (left): it provides a bigger enlarged

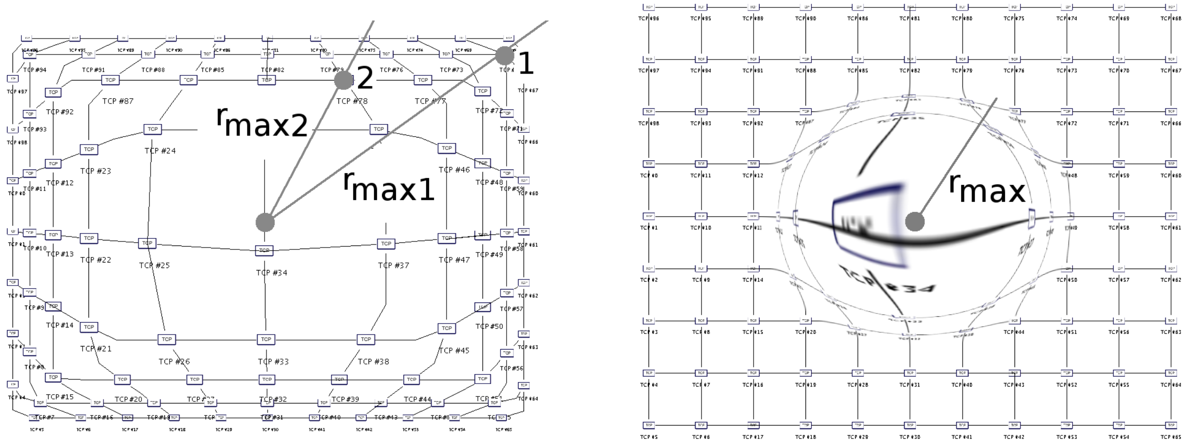


Figure 5.7: Fisheye view with variable and fixed radius

focus area than the fixed radius view (right).

For this reason, we employ a variable radius approach in which the focus area can also be freely moved by the user throughout the model. In this way, objects within the focused area are displayed in an enlarged scale whereas the others become gradually smaller as they are approach the model borders. The modeller is thereby still able to perceive the context in which he is working whilst at the same time useless details are suppressed by optical miniaturisation.

The transformation function we use (gathered from [99]) is defined with basis on the distance r between the centre of the view and the coordinate that should be projected:

$$f(r) = r_{max} \frac{(v+1)\frac{r}{r_{max}}}{\frac{r}{r_{max}} + 1}$$

In this expression, the distortion parameter $v \geq 0$ controls the relative enlargement of the focused area in relation to the surroundings. The variable radius r_{max} is the length of the radius r extended up to the border of the visible view area.

As a trade-off between efficiency and quality of the graphical projection, the edges of the models are drawn as lines, as illustrated in Figure 5.7 (left) and are not continuously transformed into curves, as illustrated in Figure 5.7 (right). While the use of curved edges in the fisheye view would improve its graphical quality, the algorithmic complexity for accomplishing this is undesirably high. For curved edges, the coordinates of each pixel on the edge need to be calculated by the transformation function each time the focus is moved by the user. In the context of modelling networks with nodes and edges, it suffices for edges to be drawn as straight lines. No photo-realistic fisheye view is needed as they aim only to represent a connection between two nodes.

Nevertheless, a slight approximation towards curved edges is still needed in order to prevent the crossing of edges, which would be prejudicial to the comprehensibility of the model. Such crossing may happen during a fisheye view due to the change in the topology of nodes and

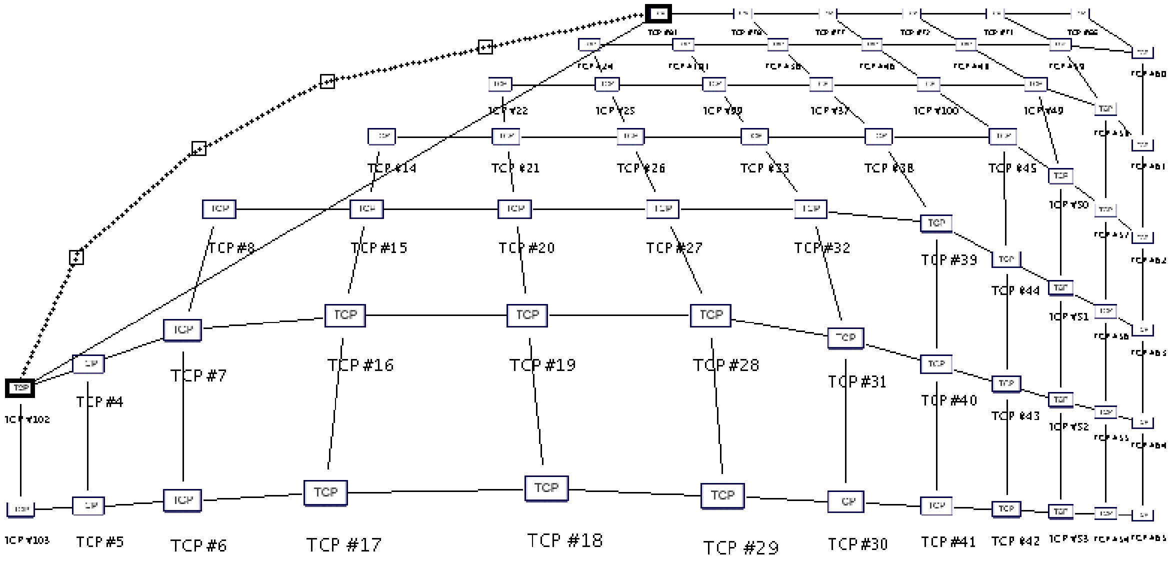


Figure 5.8: Crossing of edges caused by distortion of fisheye view and its prevention

edges that occurs as a side effect of the different distortion suffered by the central areas and the surroundings. This, in combination with the representation of edges as straight lines, can lead to new crossings of edges, such as the one illustrated in Figure 5.8 between the two marked nodes. Thus, our drawing function prevents the crossings by inserting temporary intermediate points in the edges, as exemplified by the dotted marked edge in the same figure.

A similar problem takes place in the graphical display of the AS model elements that are drawn as simple rectangular boxes by the graphical library we employ. For a photo-realistic fisheye view these boxes should also be transformed. However, taking into account the computational costs involved, in our context the much simpler approach of not transforming them is sufficient. Figure 5.9 shows the resulting visual effect of the fisheye view for a DAS. The focus in this figure is located on the leftmost AS, which is displayed in a larger scale in comparison to the scales of the other ASs (gradually reduced as one goes to the right).

5.3.3 Loosely Related Work

Though there are several applications of *focus & context* techniques for improving the usability of generic graph editors (including the recent applications to UML in [76] and the more generic approach in [60]), as far as I could investigate, they have not yet been used in the context of model visualisation and navigation for network security system design.

In a wider context, Damianou *et al.* [28] present a set of tools for the specification, deployment and management of policies specified in the *PONDER* language [26]. This language supports the definition of management and security policies and is based on domains, which are hierarchical structures used to group managed objects. The tool prototype includes a domain

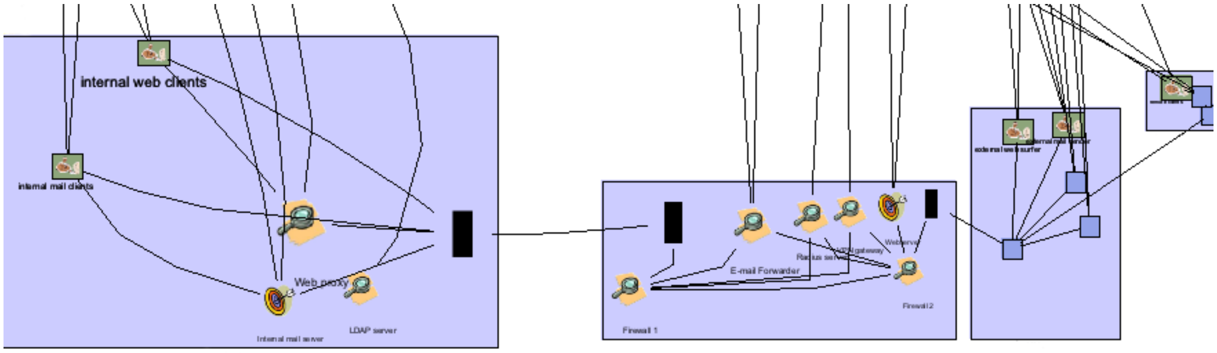


Figure 5.9: Visualisation of a DAS in the fisheye view

browser that uses the fisheye view technique to handle large structures. While centring the approach in the policies, this work does not provide a representation of the architecture of the system to be managed, making it hard for the system designer to associate the policies with his/her mental model of the system. A further work by these authors offers an approach to the implementation and validation of PONDER policies for Differentiated Services using the DMTF CIM [29] to model network elements [69]. CIM concentrates on the modelling of management information (e.g. device's capabilities and state) while our model represents the whole relevant structure of the managed system together with the management components, producing a graphical representation understandable to the security administrator.

5.4 Chapter Summary

This chapter elaborates on the policy refinement that is automatically performed by the tool, once the modelling process is complete. Firstly, the supported policy types are analysed and their relation with the other abstraction level entities is established. Then, the refinement process trough the different model levels is explained. This process culminates with the configuration parameter generation for each security mechanism modelled.

Section 5.2 presents the tool support offered by the MBM: the MoBaSec tool. The extensions implemented in the context of this work were described. In particular, Sect. 5.3 describes the *focus and context* techniques incorporated to enhance the visualisation and manipulation of large models.

Chapter 6

Formal Refinement Validation and Analysis

The consistency among abstraction levels of a policy hierarchy is a crucial issue. Only if the policy sets at the different levels are in perfect harmony, one can trust the system behaviour resulting from the application of the lowest-level policies to be in conformance with the specified abstract goals. Specifically, an automated policy refinement process as the one described in the previous section is only of practical use if we can be certain that the generated lower-level policies adhere to the abstract policies defined by the modeller.

Following the observations of Abrams and Bailey [1] and Sloman and Lupu [105], it is important to ensure the following properties:

Completeness: the desired behaviour specified in an abstract manner (i.e. the abstract positive policies) is completely implemented at the lower levels;

Consistency: all the actions enabled at the lower-level do not contradict the high-level undesired behaviour specification; i.e. the possible system behaviour is constrained by the abstract negative policies.

Note that *completeness* concerns *positive* policies, whilst *consistency* deals with *negative* policies. As such, these two properties are complementary and provide thereby the criteria necessary and sufficient to ensure the propagation of the meanings conveyed by policies and system model in the MBM approach (see Sect. 5.1.1).Thus, this work assumes that the fulfilment of these two criteria attests the *correctness* or *validity* of the policy refinement.

Therefore, the main goal of the present chapter is to validate the automated refinement described in Section 5.1.1; i.e. to prove that the application of the policy refinement to a model instance always complies with the aforementioned validation criteria. Since the refinement correctness between the levels RO and SR was already extensively studied by Lück [68], the topmost abstraction level (RO) is let out of the scope of the analysis here. The validation presented as follows is thus based on an analysis of the policy refinement that starts from the SR level. On the one hand, the analysis considers a complete model after the policy refinement, which is composed of both a group of system objects and a policy set for each of the levels SR, DAS, and ES. On the other hand, another important issue to be analysed is the effect on the real world that the implementation of the lowest-level policy set has.

The validation approach of this work consists of establishing a series of consistency conditions that the model must fulfil in order to be valid. These consistency conditions concern both policies and system objects, and are expressed in terms of relations among the various model objects and classes. Subsequently, two theorems are presented to prove that the defined conditions are sufficient and necessary to guarantee the validity of the refinement process. These theorems establish a connection between the *input* for the refinement process and its *output*. In this respect, the *input* consists of the system view and the policies at the most abstract level considered (SR), whilst the ultimate *output* is the possible system behaviour that results from using the configuration parameters produced.

Firstly, the next section presents a formalism of the relevant model entities and relationships. This formalism will serve as a basis for all subsequent sections. Thereafter, Section 6.2 gives a detailed overview over the validation approach as whole, in order to guide the reader throughout the remaining sections of this chapter.

6.1 Formalism of the Model

In the formalism used in this work, each class in the meta-model is represented by a set, whilst each object of that class is then an element of the corresponding set. For instance, the *User* Class is formalised by the set U, and an user object "Tom" is represented by an element, say u, such that $u \in U$. As appears in this example, a notation convention adopted here is that each set is denoted in capital letters, and the elements in lower case letters. Moreover, the default name for an arbitrarily chosen element of a given set will be the lower case version of the set's name.

The possible connections between two or more classes in the meta-model is then represented by a relation on those sets that formalise the classes. Each particular instance in such relation thus represents one or more edges in the model. As a convention, whenever an element x of a set in a given level is associated to another element y of a set in the immediate superior layer, we say that x is a refinement from y. Consider, for instance, the relation of refinement that exists between a *User* and a *SubjectType*, in the SR level, and an *Actor* in DAS. The sets that formalise these classes are correspondingly U, St and A. Each actor refines a pair user-subject type (see Sect. 4.2.4), so that a relation $RA \subseteq A \times U \times St$ is defined, in order to formalise this fact. For example, suppose that an actor $a \in A$ is associated to the user $u \in U$ and to the subject type $st \in St$ in the model (as in the left hand of Fig. 6.4). Thus, this association is formalised by the tuple $(a, u, st) \in RA$, and we say that a is a refinement from u, and also that a is a refinement from st, or yet that a is a refinement from (u, st). Notice also that, as a convention, the refinement relations are named beginning with the letter R followed by the refining class; i.e. the class (in the lower abstraction level) that refines the other class or classes.

In addition to these formal entities that are directly derived from the model, some auxiliary sets and functions are also defined in order to ease the notation of the expressions in the following sections.

Following these principles, the following definitions formalise the meta-model entities that are relevant for the policy refinement validation presented later on.

Definition 6.1.0.1. The SR level has the following components:

- U, St, Sv, R, disjoint sets respectively encompassing objects of the classes *User*, *SubjectType*, *Services*, and *Resources*;
- $\mathit{SvDep} \subseteq Sv \times Sv \times R$, a partially ordered relationship to express that a service depends on another to access a resource;
- $SP \subseteq U \times St \times Sv \times R$, a set of *ServicePermission* objects (see Sect. 5.1.1).

Along with the set of *positive* authorisation policies SP (Sect. 5.1.3), we also define a set of *negative* authorisation policies $\overline{SP}$, which is complementary to the former one. This second set is due to the semantic of *closed policies* that *ServicePermissions* have in our modelling (see Sect. 5.1.2).

Definition 6.1.0.2. The set of negative authorisation policies for the SR level is defined as follows.

$$\overline{SP} = \{x \in U \times St \times Sv \times R \mid x \notin SP\}$$

Definition 6.1.0.3. The DAS level comprises the following elements:

- A, M, T, C, Su, sets enclosing respectively *Actors*, *Mediators*, *Targets*, *Connectors*, and Subsystems.
- $DAS = (V, E)$, where $V = (A \cup\ M \cup\ T \cup\ C)$, and E is a set of *undirected* edges that connect the nodes in V (definition for the DAS graph itself);
- $\mathit{sub} : V \rightarrow Su$, a function that gives the subsystem to which a certain element of V is assigned in the model.

Definition 6.1.0.4. A *local DAS path* is a path in the *DAS* graph that is completely contained into a single subsystem; i.e. it *spans* one subsystem. The set of local DAS paths is defined as follows.

$$\begin{aligned} LP = \big\{ \langle v_1, \ldots, v_n \rangle \mid v_1, \ldots, v_n \in (A \cup\ M \cup\ T)\ \wedge\ \mathit{sub}[v_1] = \cdots = \mathit{sub}[v_n] \\ \wedge\ (v_j, v_{j+1}) \in E \text{ for } j = 1, 2, \ldots, n-1 \big\} \end{aligned}$$

The set P encloses all *DAS paths*; i.e. each element of P is either a local DAS path (in LP) or it spans $x > 1$ subsystems and has the recursive form $\langle p_1, c_1, c_2, p_2 \rangle$, where:

(i) $p_1 \in LP$ is local DAS path, such that $p_1 = \langle v_1, \ldots, v_k \rangle$;

(ii) $p_2 \in P$ is a DAS path that *spans* $x - 1$ subsystems, such that $p_2 = \langle v_{k+1}, \ldots, v_m \rangle$;

(iii) c_1 and c_2 are connectors such that (v_k, c_1), (c_1, c_2) and (c_2, v_{k+1}) are edges of DAS (i.e. they are elements of E).

The set ATP contains the authorisation policies at the DAS level. Each element of ATP is a DAS path between an *Actor* a and a *Target* t; i.e. it has the form $\langle v_1, \ldots, v_n \rangle \in P$, where $v_1 = a$ and $v_n = t$.

Definition 6.1.0.5. The security requirements and assumptions (see Sect. 4.1.3) are defined by the following elements:

- $SL := \{1, 2, 3, 4\}$, the set of security levels;
- $SC \subseteq SL^4$, the set of security classes (4-tuples of security levels);
- $sr : SP \to SC$, a function that gives the security class required by a *ServicePermission*;
- $sa : Sv \cup Su \cup V \to SC$, a function that returns the security assumptions of services, subsystems, and elements in DAS.

Definition 6.1.0.6. Suppose sc_1 and sc_2 are security classes in SC, such that $sc_1 = (l_1, l_2, l_3, l_4)$ and $sc_2 = (m_1, m_2, m_3, m_4)$. Thus the following operations are defined:

- $sc_1 \leq sc_2$ is the partial order in SC that comes from the product order of the ordinary integer ordering; i.e. $l_i \leq m_i$ for $i = 1, 2, 3, 4$;
- $sc_1 \sqcup sc_2 = (n_1, n_2, n_3, n_4)$ means that if $l_i \geq m_i$ then $n_i = l_i$, otherwise $n_i = m_i$ for $i = 1, 2, 3, 4$;
- $sc_1 \sqcap sc_2 = (n_1, n_2, n_3, n_4)$, means that if $l_i \leq m_i$ then $n_i = l_i$, otherwise $n_i = m_i$ for $i = 1, 2, 3, 4$.

Definition 6.1.0.7. The associations between elements of SR and DAS are defined as:

- $RA \subseteq A \times U \times St$, representing abstraction refinements from a pair of *User* and *SubjectType* objects to an *Actor*;
- $RM \subseteq M \times Sv$, refinements from *Services* to *Mediators*;
- $RT \subseteq T \times Sv \times R$, refinements from *Service* and *Resource* pairs to *Targets*;
- $RATP \subseteq ATP \times SP$, refinements from service permissions to ATPs.

Definition 6.1.0.8. The ES level has the following elements:

- Uc, Pc, H, So, Nc, sets correspondingly enclosing objects of the types: *UserCredentials*, *Processes*, *Hosts*, *SystemObjects*, and *NetworkConnection* (this encloses protocols, network interfaces, network segments, etc.);
- $ES := (W, F)$, where $W = (Uc \ \cup \ Pc \ \cup \ H \ \cup \ So \ \cup \ Nc)$, and F is a set of the *directed* edges that connect nodes in W.

Definition 6.1.0.9. A *local ES path* is a path in the ES graph that is completely contained into a single subsystem; i.e. it *spans* one subsystem. The set of local ES paths is defined as follows.

$$LEP = \{\langle v_1, \ldots, v_m \rangle \mid v_1, \ldots, v_m \in W \ \wedge \ (v_i, v_{i+1}) \in F \text{ for } 1 \leq i < m\}$$

The set EP contains all expanded paths in a model; i.e. paths formed by the concatenation of local ES paths through pairs of connectors. Each element of EP is thus either a local ES path (i.e. an element of LEP) or a path that spans $x > 1$ subsystems and has the recursive form $\langle ep_1, c_1, c_2, ep_2 \rangle$, where:

(i) $ep_1 \in LEP$ is a local ES path, such that $ep_1 = \langle v_1, \ldots, v_k \rangle$;

(ii) $ep_2 \in EP$ is an expanded path that *spans* $x-1$ subsystems, such that $ep_2 = \langle v_{k+1}, \ldots, v_m \rangle$;

(iii) c_1 and c_2 are connectors such that $(c_1, c_2) \in E$, $(v_k, c_1) \in NcC$, and $(v_{k+1}, c_2) \in NcC$.

The set AEP of *Allowed Expanded Paths* (Sect. 5.1.5) is the subset of EP whose elements represent policies in the ES level; i.e. the elements of AEP are the expanded paths that the system must allow.

Definition 6.1.0.10. The associations between elements of the ESs and DAS are defined by the following relations:

- $RUc \subseteq Uc \times A \times U$, refinements from *Actors* and *Users* to credentials;
- $RPc \subseteq Pc \times A \cup \ M \cup \ T$, refinements from *Actors*, *Mediators* or *Targets* to processes;
- $RSo \subseteq So \times T$, refinements from *Targets* to system objects;
- $NcC \subseteq Nc \times C$, connections between network connection objects and *Connectors*;
- $RES \subseteq W \times V$, general refinements from components of DAS to ES nodes;
- $RAEP \subseteq AEP \times ATP$, refinements from ATPs to AEPs;
- $sub : W \rightarrow Su$, overriding of function *sub* to map the association of ES nodes to subsystems.

6.2 Validation Approach Overview

Figure 6.1 depicts the overview of the condition sets we define in the following sections. Each of these sets is represented by a bold line with arrows pointing to the related model entities; the numbers beside indicate the section in which the set is defined. Thus, we first establish the consistency conditions for a valid refinement from the SR level to the DAS in Section 6.3. These conditions are subdivided into two groups: refinement consistency conditions and structural consistency conditions. The first subgroup is presented in Section 6.3.1 and contains conditions that validate the DAS policy set (ATP) in comparison to the two types of policies at the SR level: service permissions (SP) and security requirements (SR). The structural conditions (Section 6.3.2), in turn, establish the compatibility of the DAS structure (i.e. the managed system representation) with the ATP policy set.

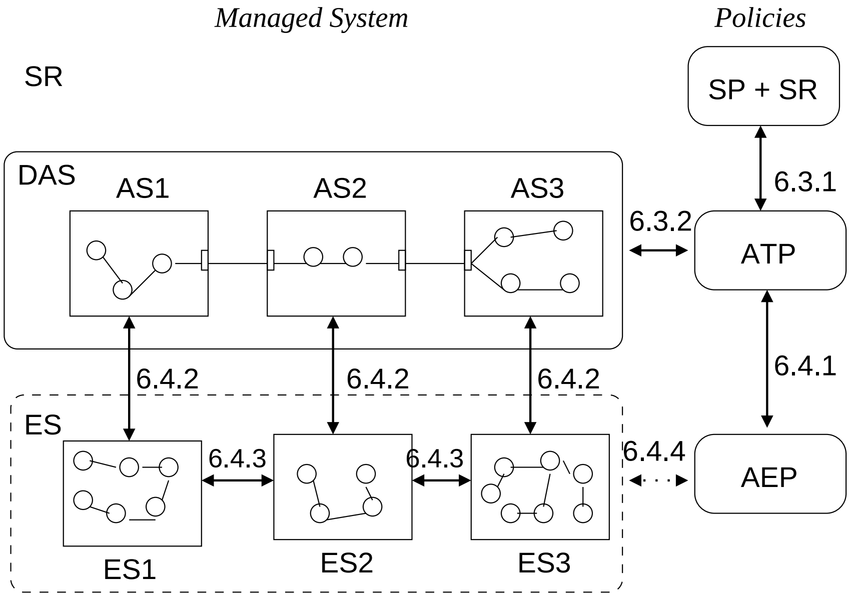


Figure 6.1: Overview of Validation Condition Sets

Subsequently, Section 6.4 analyses the relation between the DAS and the Expanded Subsystems level (ES). Three subgroups of conditions are defined here: refinement consistency conditions, local structural consistency conditions, and composition consistency conditions. The conditions of the first group are presented in Section 6.4.1 and validate the ES policy set (AEP) in comparison to ATP. As at the previous level, the second subgroup (Section 6.4.2) also comprises structural conditions, but this has an important difference: due to the segmented structure of the ES level, these conditions have a *local* scope. They aim at checking whether the abstract view of each subsystem (AS) corresponds to the subsystem elements in the expanded view (ES), and are thus restricted to consider the elements within the boundaries of each subsystem in turn. Section 6.4.3 presents the composition consistency conditions, which validate the interconnection between ESs. Furthermore, we prove in general (by means of the theorems in Section 6.4.4) that the local conditions can be generalised relying on the assertion of the composition conditions. Since these theorems do not have to be checked for each model instance as the other condition sets do, they are represented in Fig. 6.1 by a dotted line that points both to the AEP set and to the whole ES level.

In order to prove that the conditions are able to validate a model, we examine thereafter the application to a real environment of the configuration generated using the policy refinement process. Section 6.5 presents assumptions about the model capability of representing the real world environment, so called *model representativeness axioms*. Thus, with basis on these axioms and on the consistency conditions, we prove two theorems in Section 6.6:

VT1: *For each policy in the SR level, the system enables all accesses in the real world that correspond to the policy*;

VT2: *To each possible access in the real world there is a corresponding policy at the SR level.*

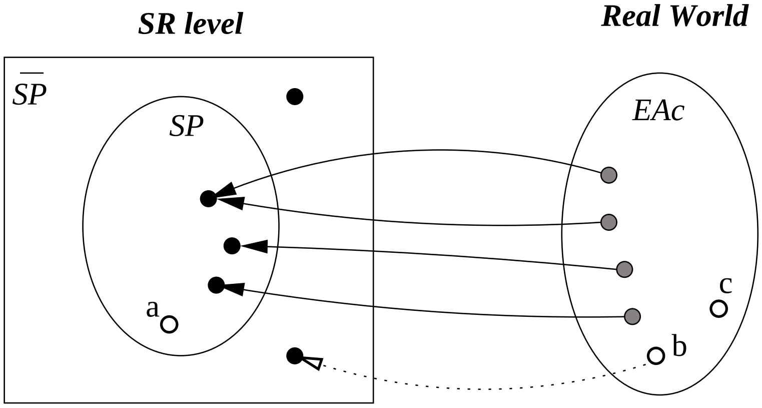


Figure 6.2: Relation between the SR level and the Real-world

These two theorems establish a relation between the *input* for the refinement process (the policies of the SR level) and its *output* (the possible accesses in the real world). This relation can be represented by the Venn diagram of Figure 6.2. The input is depicted on the left-hand side by the SP set of service permissions (i.e. the positive authorisation policies), and its complementary set $\overline{SP}$ with the negative authorisation policies (see Sect. 6.1). On the right-hand side, the output is represented by the set of enabled accesses (EAc); i.e. all potential accesses in the real world that are enabled by the whole security system. The arrows connect each of these accesses with its corresponding abstract representation in the sets SP and $\overline{SP}$. In fact, security requirements are also part of the input, but they can be seen in this scheme as a restriction of the relation of abstract representation, such that an access in the real world is only represented by an element of the SR level if the security assumptions of the former comply with the security requirements of the latter.

In Fig. 6.2, valid elements of each set are represented by black or gray circles (for the input and output, respectively), while white circles stand for invalid elements; i.e. elements that will not be present if the refinement is valid. Indeed, VT1 assures that each element of SP will have all of its corresponding accesses in the real world enabled (i.e. they will pertain to EAc). Therefore, a SP element such as a, which does not have a related element in EAc, will not exist if VT1 holds (considering all SP elements have at least one corresponding access in the real world, see Axiom 2 in Sect. 6.5). Conversely, the existence of elements such as b and c in EAc would violate VT2. The element b represents a possible real-world access that has an abstract representation in the SR level which do not pertain to the policy set SP – thus contradicting VT2. As for c, it stands for an access that does not have an abstract representation at all, and that is just as well forbidden by VT2.

To conclude the validation, Sect. 6.7 makes considerations about the validation soundness and Sect. 6.8 then analyses the verification complexity of the conditions defined.

6.3 SR/DAS Congruence

6.3.1 Refinement Consistency Conditions

The refinement validation from SR level to DAS must assert the consistency between the two levels in respect to both authorisation policies and security requirements. For this purpose, this first group of conditions concerns the refinement relation $RATP$ (Def. 6.1.0.10) between the set of authorisation policies at SR (SP) and the homologous set in DAS (ATP). To improve legibility, I will interchangeably use henceforth the terms *service permission*, *SR permission*, and the symbol *sp* to refer to elements in SP; i.e. to policies in the SR level. Analogously, the terms *DAS permission* and *ATPermission*, and the symbol *atp* shall all indicate an element of the DAS policy set ATP.

Before presenting the first condition, the predicate *atcomp* is following introduced in order to capture the compatibility between actors and targets of the model. A compatible pair actor-target will have corresponding processes associated to protocol stacks that contain the same protocol types and differ only in the connection direction (outgoing for actors and incoming for targets). Thus, they will effectively be a potential communication pair.

Definition 6.3.1.1. The predicate $atcomp : A \times T \rightarrow \{0, 1\}$ denotes whether a pair actor-target is compatible; i.e. whether the actor and the target are effectively able to communicate.

In order to facilitate the notation of service dependencies, the condition relies also on the following auxiliary predicate.

Definition 6.3.1.2. The predicate $dep : Sv \times Sv \times R \rightarrow \{0, 1\}$, indicates if a service depends on another to access a resource. It is recursively defined as follows.

$$dep[sv_1, sv_2, r] = \begin{cases} 1 & \text{if } sv_1 = sv_2 \\ & \quad \vee\ (sv_1, sv_2, r) \in SvDep \\ & \quad \vee\ \big(\exists\ sv_3 \in Sv : (sv_1, sv_3, r) \in SvDep \wedge\ dep[sv_3, sv_2, r]\big) \\ 0 & \text{otherwise} \end{cases}$$

The first refinement consistency condition aims at establishing that for each SR permission the model contains a *corresponding* DAS permission, so that all the authorisation policies in the SR level are *structurally feasible* in the DAS level. Consequently, all accesses in the DAS which are related to service permissions will be completely enabled by corresponding elements of the ATP policy set.

Condition 6.3.1.1. Let $sp = (u, st, sv_1, r)$ be a service permission in SP. For each actor $a \in A$ that refines the pair *user-subject type* (u, st), and each target $t \in T$ that refines a pair *service-resource* like (sv_1, r), given that a and t are compatible, the set ATP must contain a DAS permission atp that connects a to t and is related to sp through $RATP$. Formally stated:

$$\begin{aligned} &\forall\ sp \in SP, a \in A, t \in T, sv_2 \in Sv : \\ &\quad sp = (u, st, sv_1, r) \wedge (a, u, st) \in RA \wedge (t, sv_2, r) \in RT \wedge atcomp[a, t] \wedge dep[sv_1, sv_2, r] \Rightarrow \\ &\qquad\qquad \exists\ atp \in ATP, atp = \langle a, \ldots, t\rangle : (sp, atp) \in RATP \end{aligned}$$

Notice that sv_1 and sv_2 may be different in case of a service dependency, i.e. if sv_1 must rely upon other services to get access to the resource r (see Sect. 5.1.3). If the service sv_1 has direct access to r, then sv_1 and sv_2 will be the same (see Def. 6.3.1.2).

The second condition conversely asserts that to each DAS permission in the set ATP, a corresponding SR permission must exist in the model. This is important so that the enabled actions in DAS can be traced back to the abstract policies that authorise them. Additionally, the condition also prevents a given DAS permission to authorise actions forbidden by a negative policy in the SR level; for each atp is required to be mapped to an authorising service permission.

Condition 6.3.1.2. Let $atp = \langle a, \ldots, t\rangle$ be an *ATPermission* in ATP. Suppose there is a user $u \in U$, a subject type $st \in St$, two services $sv_1, sv_2 \in Sv$, and a resource $r \in R$, such that a refines (u, st), and t refines (sv_1, r). Thus, the set SP must contain a service permission $sp = (u, st, sv_2, r)$ that is related to atp through $RATP$ and the service sv_1 must have access to r by a dependency chain that passes through sv_2 (sv_1 and sv_2 may also be the same, see Def. 6.3.1.2). Formally stated:

$$\begin{aligned} &\forall\ atp \in ATP, u \in U, st \in St, sv_1, sv_2 \in Sv, r \in R : \\ &\qquad atp = \langle a, \ldots, t\rangle \wedge (a, u, st) \in RA \wedge (t, sv_2, r) \in RT \wedge dep[sv_1, sv_2, r] \Rightarrow \\ &\qquad\qquad \exists\ sp \in SP, sp = (u, st, sv_2, r) : (sp, atp) \in RATP \end{aligned}$$

Subsequently, the third condition shall validate the *correspondence* of pairs (atp, sp) contained in the relation $RATP$. This correspondence is established by checking not only the structural connections between the system objects related to atp and sp, but also by ensuring the satisfaction of service dependencies and the fulfilment of security requirements.

The fulfilment of security requirements demands closer examination. In this respect, the first point to be considered is that security assumptions of services that are related to the DAS elements along a given path are assumed to be active throughout the whole path. As such, the security assumptions associated to services are used to model situations in which a certain service, with particularly desirable security properties, is employed to improve the security level of the whole communication path. For instance, a packet filter with activated logging may improve the traceability of all communication flows that pass through it[1].

The security class that results from the combination of the security assumptions of all the services related to elements along an *ATPermission* is called *overall security assumption*. It is denoted by the function *osa*.

Definition 6.3.1.3. Let $atp = \langle v_1, \ldots, v_n \rangle$ be an element of ATP and $Sva \subset Sv$ the set of services associated to the elements of atp, such that $Sva = \{sv \mid (v_j, sv) \in RM \vee (v_j, sv, r) \in RT$ for some $1 \leq j \leq n\}$. The function $osa : ATP \rightarrow SC$ is thus defined as:

$$osa[atp] = \bigsqcup_{sv \in Sva} sa[sv]$$

The combination of the security classes is performed in this definition by a generalisation of the binary operator $\sqcup$ declared in Def. 6.1.0.6. It selects the greater security level independently for each dimension of the security classes; i.e. for each category of security requirement considered (see also Sect. 4.1.3). Thus, the resulting security class reflects the joint work of all the services involved.

We turn our attention now to the security class assured by a given individual DAS element in the context of a *ATPermission*. There are three security assumptions that must be considered: a) the overall security assumption of the *ATPermission*; b) the security assumption associated to the subsystem to which the element pertains (i.e. the security properties expected from the environment wherein the individual is located); and c) the security class assigned to the node itself (represented by its corresponding assumption). These three sources of information about security properties correspond to different protection layers that build upon each other. Although separately modelled, they are all active at the same time in a given component of the real system. Therefore, the security class effectively active in a certain DAS object is the result of the combination of the the security classes from (a), (b), and (c) – just like a chord is the sonorous effect of various tones simultaneously produced.

The security class that a DAS node can assure in the context of an *ATPermission* is thus henceforth called *effective security assumption*, and it is denoted by the function *esa*.

[1] In fact, encryption services that are related to Virtual Private Network mechanisms are not active in the whole path, but rather in the subpath between two VPN gateways. However, since the formalism here proposed can be simply adapted to reflect this fact, it is assumed, for the sake of conciseness, that all service assumptions act throughout the whole path

Definition 6.3.1.4. Let v be a DAS node, such that $atp \in ATP$ contains v. The function $esa : V \times ATP \rightarrow SC$ is thus defined as:

$$esa[v, atp] = sa[v] \sqcup sa\big[sub[v]\big] \sqcup osa[atp]$$

To finish the consideration of security assumptions in this track, let us analyse now the security properties that a path as a whole can be expected to assure. At this plane, we follow the principle according to which a security chain is only as strong as its weakest link. However, since security assumptions enclose 4 categories of requirements, the assumption of a whole path cannot be picked up from a single object. Rather, the weakest security level for each category must be determined independently among the effective security assumptions of each element along the path. As such, the security assumption of the path reflects common properties shared by all of its participating elements.

The security class that can be assured in the context of an *ATPermission* is thus denoted by the function *psa*.

Definition 6.3.1.5. The *path security assumption* of an *ATPermission* represents the security class that all elements along the path can provide. It is given by the function $psa : ATP \rightarrow SC$. Let $atp = \langle v_1, \ldots, v_n \rangle$ be an element of ATP, thus:

$$psa[atp] = \bigsqcap_{1 \leq j \leq n} esa[v_j, atp]$$

This definition relies upon a generalisation of the binary operator $\sqcap$ declared in Def. 6.1.0.6. It selects the lowest values independently for each dimension of the security classes – thus yielding the desired result for the whole path.

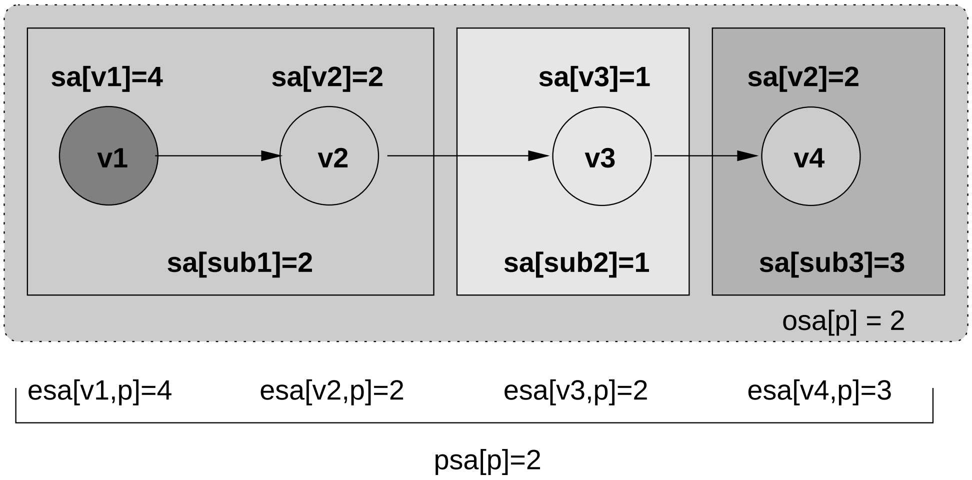


Figure 6.3: Example of active security assumptions in a path

Figure 6.3 illustrates the previous concepts with a path example and its associated security assumptions. The path p comprises four nodes (v1–v4) which are distributed over three different subsystems (sub1, sub2 and sub3). The gray tones reflect the security assumption of each element, such that the darker is the background colour of a given object the higher (stronger) is the security level that it can provide (notice that the security classes consist of a vector of four levels, but in this example only one level is considered for the sake of clearness). For instance, the node v1 is assumed to ensure level 4 (sa[v1]=4) even though the subsystem to which it belongs has an assumption of level 2 only (sa[sub1]=2). The effective security level that v1 is assumed to provide in the path is then a combination of the previous two levels with the *overall path assumption* of p – which is of level 2; i.e. osa[p]=2 – resulting in an *effective security assumption* of level 4 (esa[v1,p]=4). On the other hand, the nodes v2 and v3 are the weakest in the path since their effective security assumption amount only to level 2. Consequently, the *path security assumption* of p is also of level 2; i.e. psa[p]=2.

Finally, relying upon the considerations above we can define the third refinement consistency condition as follows.

Condition 6.3.1.3. Let $sp = (u, st, sv, r)$ be a service permission in SP and $atp \in ATP$ an *ATPermission*, such that $sp = (u, st, sv, r)$, $atp = \langle v_1, \ldots, v_n, \rangle$, and the two are related by $RATP$; i.e. $(atp, sp) \in RATP$. Hence the following propositions must hold:

(i) the actor $a \in A$ must refine the pair *user-subject type* (u, st);

(ii) the target $t \in T$ must refine a pair *service-resource* like (sv_t, r);

(iii) actor a and target t must be compatible (Def. 6.3.1.1);

(iv) the *path security assumption* of *atp* must fulfil the security requirements of *sp*;

(v) for each service sv_d on which sv depends to provide the resource r, either sv_d is the service sv_t related to the target, or a mediator that refines sv_d must be found along *atp*.

Formally stated:

$$\begin{aligned}\forall\ atp \in ATP, sp \in SP : atp = \langle v_1, \ldots, v_n, \rangle \wedge sp = (u, st, sv, r) \wedge (sp, atp) \in RATP \Rightarrow \\ (v_1, u, st) \in RA \wedge (v_n, sv_t, r) \in RT \wedge atcomp[a, t] \wedge sr[sp] \leq psa[atp] \\ \wedge \forall\ sv_d : dep[sv, sv_d, r] \Rightarrow sv_d = sv_t \vee \exists\ v_j : (v_j, sv_d) \in RM \text{ for } 1 < j < n\end{aligned}$$

A generic example of a service permission *sp* and its corresponding *atp* is shown in Figure 6.4. Each of the required items in Cond. 6.3.1.3 can be seen to be satisfied by *atp* – except the security class compliance which is not depicted by objects, but is represented in their properties.

6.3.2 Structural Consistency Conditions

The following conditions impose structural restrictions for the connections among DAS elements and SR elements, and also in relation to the ATP policy set of the DAS level.

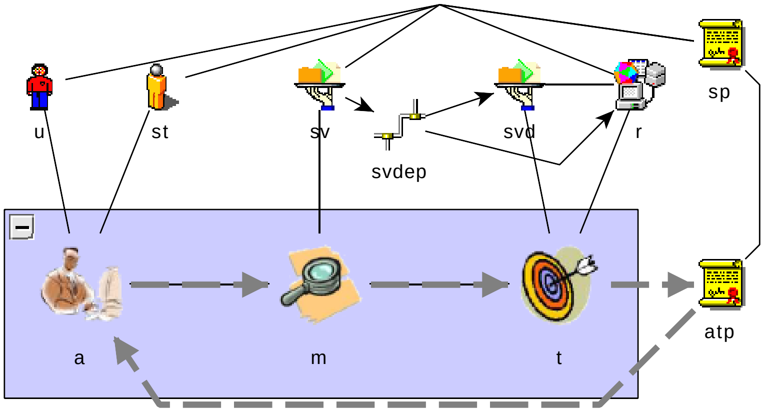


Figure 6.4: Example of correspondence between a service permission and an ATPermission

Condition 6.3.2.1. Let $a \in A$ be an actor in the model. Thus, there must exist a user $u \in U$ and a subject type $st \in St$, such that a is associated to the pair (u, st). Formally stated:

$$\forall\ a \in A \Rightarrow \exists\ u \in U, st \in St : (a, u, st) \in RA$$

Condition 6.3.2.2. Let $t \in T$ be a target in the model. Thus, there must exist a service $sv \in Sv$ and a resource $r \in R$, such that t is associated to the pair (sv, r). Formally stated:

$$\forall\ t \in T \Rightarrow \exists\ sv \in Sv, r \in R : (t, sv, r) \in RT$$

Condition 6.3.2.3 (DAS Edge Minimality)**.** Let $v_1, v_2 \in V$ be two nodes in DAS, such that there is an edge connecting them. Thus, there must be some permission $atp \in ATP$ that contain the edge (v_1, v_2).

$$\forall\ v_1, v_2 \in V : (v_1, v_2) \in E \Rightarrow\ \exists\ atp \in ATP : \langle v_1, v_2 \rangle \subseteq atp$$

Condition 6.3.2.4. Let v_1, v_2 be two vertices in DAS which are not connectors, and $c_1, c_2 \in C$ be two connectors. Suppose there is a DAS path that connects v_1 to v_2 passing through c_1 and c_2. Thus, there must exist a permission $atp \in ATP$ that includes the path $\langle v_1, c_1, c_2, v_2 \rangle$. Formally stated:

$$\forall\ v_1, v_2 \in (V - C),\ c_1, c_2 \in C : \langle v_1, c_1, c_2, v_2 \rangle \in P \Rightarrow \exists\ atp \in ATP : \langle v_1, c_1, c_2, v_2 \rangle \subseteq atp$$

6.4 DAS/ES Congruence

6.4.1 Refinement Consistency Conditions

Definition 6.4.1.1. Consider a DAS path $ap \in P$ in an expanded path $ep \in EP$ (see Defs. 6.1.0.4 and 6.1.0.9). Let $ap = \langle v_1, \ldots, v_n \rangle$, and let $e_1, \ldots, e_m$ be the ordered sequence of processes and connectors of *ep*; i.e. the sequence of the elements in *ep* in the order they occur, such that $e_i \in Pc \cup C$, for $1 < i < m$. The predicate $expand : P \times EP \rightarrow \{0, 1\}$ is true if an expanded path is a valid expansion of a DAS path, and is defined as follows.

$$expand[ap, ep] \Leftrightarrow \; m = n \; \wedge \big((e_i, v_i) \in RPc \; \vee \; e_i = v_i\big) \quad \text{for } 1 \leq i \leq m$$

Condition 6.4.1.1. Let $atp = \langle a, \ldots, t\rangle$ be an element of ATP. Suppose there are two processes $pc_a, pc_t \in Pc$, such that pc_a refines the actor a and pc_t refines the target t. Thus, an allowed expanded path $aep = \langle pc_a, \ldots, pc_t\rangle$ must exist in AEP, such that *aep* is a refinement and a valid expansion of *atp*. Formally stated:

$$\begin{aligned} &\forall\; atp \in ATP,\; pc_1, pc_2 \in Pc : atp = \langle a, \ldots, t\rangle \wedge (pc_a, a), (pc_t, t) \in RPc \Rightarrow \\ &\qquad\qquad \exists\; aep \in AEP,\; aep = \langle pc_a, \ldots, pc_t\rangle : (aep, atp) \in RAEP \wedge expand[aep, atp] \end{aligned}$$

Condition 6.4.1.2. Let *aep* be an allowed expanded path in AEP. Thus, an *atp* must exist in ATP such that *aep* refines and is a valid expansion of *atp*. Formally stated:

$$\forall\; aep \in AEP \Rightarrow \exists!\; atp \in ATP : (aep, atp) \in RAEP \wedge expand[aep, atp]$$

6.4.2 Local Structural Consistency Conditions

The following auxiliary predicates are defined to improve legibility of the conditions presented later in this section.

Definition 6.4.2.1. An expanded path in the model (Def. 6.1.0.9) is said *compatible* if it represents a valid path; i.e. if it connects processes through valid protocol stacks, and connectors. The predicate $compatible : EP \rightarrow \{0, 1\}$ indicates whether an expanded path is compatible.

This predicate is not formally defined, since it depends on model details that are not formalised, such as which protocol stacks are compatible, and which process types may be traversed (i.e. which can act as a gateway). All those details are irrelevant for the purposes of this validation and are thus abstractly treated by means of the above predicate.

Definition 6.4.2.2. A local ES path (Def. 6.1.0.9) is said *locally connected* if it connects two processes and is compatible. This fact is denoted by the predicate $lconn : LEP \rightarrow \{0, 1\}$, formally defined as follows.

$$lconn[p] \Leftrightarrow \; pc_1, pc_n \in Pc \wedge v_2, \ldots, v_{n-1} \in (W - Pc) \wedge compatible[p]$$

Now, a first set of conditions deal with the refinement relations DAS/ES.

Condition 6.4.2.1. Let v be an actor or target in DAS. Thus, a process $pc \in Pc$ must exist that refines v. Formally stated:

$$\forall\; v \in A \cup T \Rightarrow \; \exists\; pc \in Pc : (pc, v) \in RPc$$

Condition 6.4.2.2. Let m be a mediator in M. Thus, a *unique* process $pc \in Pc$ must exist that refines v. Formally stated:

$$\forall\ m \in M \Rightarrow\ \exists!\ pc \in Pc : (pc, m) \in RPc$$

Condition 6.4.2.3. Let pc be a process in Pc. Thus, an element in DAS must exist that is refined by pc. Formally stated:

$$\forall\ pc \in Pc \Rightarrow\ \exists!\ v \in A \cup\ M \cup\ T : (pc, v) \in RPc$$

Condition 6.4.2.4. Let uc be an user credential in Uc, a be an actor in A and u be a user in U, such that uc refines (a, u). Thus, a subject type $st \in St$ must exist, such that a refines (u, st). Formally stated:

$$\forall\ uc \in Uc, a \in A, u \in U : (uc, a, u) \in RUc \Rightarrow \exists\ st \in St : (a, u, st) \in RA$$

The following conditions are then related to the correspondence between DAS edges and local ES paths.

Condition 6.4.2.5. Let $v_1, v_2 \in V$ be elements of DAS, and let $pc_1, pc2$ be processes in the corresponding ES, such that there is a DAS edge $(v_1, v_2) \in E$, pc_1 refines v_1 and pc_2 refines v_2. Thus, a local ES path must exist that connects pc_1 to pc_2 and is locally connected (Def. 6.4.2.2). Formally stated:

$$\forall\ v_1, v_2 \in V,\ pc_1, pc_2 \in Pc : (v_1, v_2) \in E \land (pc_1, v_1), (pc_2, v_2) \in RPc \Rightarrow \\ \exists\ p = \langle pc_1, \ldots, pc_2 \rangle : lconn[p]$$

Condition 6.4.2.6. Let $pc_1, pc_2 \in Pc$ be two processes that are connected by a locally-connected ES path p (Def. 6.4.2.2). Thus, two DAS nodes $v_1, v_2 \in V$ must exist such that pc_1 refines v_1, pc_2 refines v_2, and there is a DAS edge $(v_1, v_2) \in E$. Formally stated:

$$\forall\ p = \langle pc_1, \ldots, pc_2 \rangle : lconn[p] \Rightarrow \exists\ v_1, v_2 \in V : (pc_1, v_1), (pc_2, v_2) \in RPc \land (v_1, v_2) \in E$$

6.4.3 Composition Consistency Conditions

The following auxiliary predicate is used in the conditions presented later in this section.

Definition 6.4.3.1. An expanded path (Def. 6.1.0.9) is considered *connector connected* if it is compatible and connects a process of a subsystem to another process in an adjacent subsystem (through a pair of connectors) without traversing other processes along the way. Formally stated:

$$cconn[p] \Leftrightarrow\ p = \langle w_1, \ldots, w_k, c_1, c_2, w_{k+1}, \ldots, w_n \rangle : \\ w_1 \in Pc \land w_n \in Pc \land w_2, \ldots, w_{n-1} \in (W - Pc - C) \land compatible[p] \land\ c_1, c_2 \in C \\ \land \langle w_1, \ldots, w_k \rangle, \langle w_{k+1}, \ldots, w_n \rangle \in LEP \land \langle v_k, c_1, c_2, v_{k+1} \rangle \in EP$$

Conditions analogous to that of the previous section are thus defined to deal with the correspondence between DAS edges and connector-connected paths.

Condition 6.4.3.1. Let $v_1, v_2 \in V$ be two DAS elements that pertain to adjacent subsystems, such that they are linked by a pair of connectors, i.e. the DAS contains he path $\langle v_1, c_1, c_2, v_2 \rangle$. Suppose that $pc_1, pc_2 \in Pc$ are processes such that pc_1 refines v_1 and pc_2 refines v_2. Thus, a *connector-connected* path must exist that goes from pc_1 to pc_2. Formally stated:

$$\begin{aligned}&\forall\ v_1, v_2 \in V,\ c_1, c_2 \in C,\ pc_1, pc_2 \in Pc :\\ &\qquad \langle v_1, c_1, c_2, v_2 \rangle \in P \wedge (pc_1, v_1), (pc_2, v_2) \in RPc \Rightarrow \exists\ p = \langle pc_1, \ldots, pc_n \rangle : cconn[p]\end{aligned}$$

Condition 6.4.3.2. Let pc_1 and pc_2 be two processes that are connected by a connector-connected path p. Thus, two DAS nodes $v_1, v_2 \in V$ must exist such that pc_1 refines v_1, pc_2 refines v_2, and the model contains the DAS path $\langle v_1, c_1, c_2, v_2 \rangle$. Formally stated:

$$\begin{aligned}&\forall\ p = \langle pc_1, \ldots, pc_2 \rangle : cconn[p] \Rightarrow\ \exists\ v_1, v_2 \in V, c_1, c_2 \in C :\\ &\qquad (pc_1, v_1), (pc_2, v_2) \in RPc \wedge \langle v_1, c_1, c_2, v_2 \rangle \in P\end{aligned}$$

6.4.4 Generalisation Theorems and Lemma

In order to prove that the local and composition conditions are sufficient to validate the abstract refinement from the DAS to the ES level, we must consider now these layers as a whole, thus achieving a generalised result from the previous considerations (as illustrated in Fig. 6.1).

Firstly, we define a generalised predicate with basis on Defs. 6.4.2.2 and 6.4.3.1.

Definition 6.4.4.1. A path is considered *connected* if it is either *locally connected* (Def. 6.4.2.2) or *connector connected* (Def. 6.4.3.1). This fact is denoted by the predicate $conn : EP \rightarrow \{0, 1\}$, formally defined as: $conn[p] \Leftrightarrow\ lconn[p]\ \vee\ cconn[p]$.

We can then proceed to generalising the previous results, by means of the following theorems.

Theorem 6.4.4.1 (GT1)**.** *Let dp be an expanded path in the model, such that it is contained in some allowed expanded path (i.e. $dp \subseteq aep \in AEP$) and assume $pc_1, \ldots, pc_m$ is the sequence of processes in dp. In this case, the model contains a connected path between each pair of successive processes in the sequence $pc_1, \ldots, pc_m$. Formally stated:*

$$\begin{aligned}&\forall\ dp \subseteq aep \in AEP :\ pc_1, \ldots, pc_m \textit{ is the sequence of processes in } dp \Rightarrow\\ &\qquad \exists\ p = \langle pc_i, \ldots, pc_{i+1} \rangle : conn[p] \textit{ for } 1 \leq i < m\end{aligned}$$

Theorem 6.4.4.2 (GT2)**.** *Let $pc_a, pc_b \in Pc$ be two processes such that there is a connected path p in the model from pc_a to pc_b. In this case, the model contains an allowed expanded path aep* *that encloses p; i.e. aep allows the connection from pc_a to pc_b through exactly the same processes along the path p. Formally stated:*

$$\forall\ pc_a, pc_b \in Pc : \exists\ p = \langle pc_a, \ldots, pc_b \rangle \wedge\ conn(p) \Rightarrow\ \exists\ aep \in AEP :$$
$$pc_1, \ldots, pc_m \textit{ is the sequence of processes in aep}$$
$$\wedge\ pc_a = pc_i\ \wedge\ pc_b = pc_{i+1} \textit{ for some } 1 \leq i < m$$

Since the proofs for these theorems are quite long, they are presented in Appendix B for the sake of legibility.

Security Assumption Lemma

Analogously to the DAS level, we must now consider the security assumption of an AEP. The auxiliary function esa' is defined to denote the *effective security assumption* that processes and connectors can assure in the context of an AEP (similarly to the function esa of Sect. 6.3.1).

Definition 6.4.4.2. Let e be a process or connector that is contained in the path $aep \in AEP$, and let $atp \in ATP$ be the *ATPermission* related to *aep*; i.e. $(aep, atp) \in RAEP$. The function $esa : Pc \cup C \times AEP \rightarrow SC$ denotes the *effective security assumption* that e can assure in the context of *aep* and is defined as follows.

$$esa'[e, aep] = \begin{cases} esa[e, atp] & \text{if } e \in C \\ esa[v, atp] & \text{if } e \in Pc, \text{ where } (e, v) \in RPc \end{cases}$$

Notice that Cond. 6.4.2.3 implies that a related DAS element v must exist to each ES process, thus assuring a value for the function esa' in the second case of the definition above.

The security class that can be assured in the context of an AEP is denoted by the function psa' (similarly to the function psa of Def. 6.3.1.5).

Definition 6.4.4.3. The *path security assumption* of an AEP represents the security class that all elements along the path can provide. It is given by the function $psa' : AEP \rightarrow SC$. Let *aep* be an element of AEP, such that $e_1, \ldots, e_m$ is the sequence of elements of *aep* that pertain to $Pc \cup C$ (i.e. only processes and connectors along the path), thus:

$$psa'[aep] = \prod_{1 \leq i \leq m} esa'[e_i, aep]$$

Now, we are able to prove the following lemma (as the proof for the lemma is small, it is presented here directly after the statement).

Lemma 6.4.4.3. *Let $aep \in AEP$ be an allowed expanded path and $atp \in ATP$ an* ATPermission. *If aep is a refinement of atp then both can assure the same security levels; i.e. the path security assumption of aep is equal of that of atp. Formally stated:*

$$\forall\ aep \in AEP,\ atp \in ATP : (aep, atp) \in RAEP \Rightarrow psa'[aep] = psa[atp]$$

Proof. Let us consider without loss of generality an allowed expanded path $aep \in AEP$ and an *ATPermission* $atp \in ATP$, such that $(aep, atp) \in RAEP$. Let $atp = \langle v_1, \ldots, v_n \rangle$ and let $e_1, \ldots, e_m$ be the sequence of processes and connectors of *aep*. According to Def. 6.4.4.3 thus follows that:

$$psa'[aep] = \prod_{1 \le j \le m} esa'[e_j, aep] \tag{6.1}$$

Since *aep* is related to *atp*, from Cond. 6.4.1.2 results that $expand[aep, atp]$ must be valid. Then relying upon the definition of *expand* (Def. 6.4.1.1), we conclude that for each e_j in the equation above, if it is a process then there is an associated v_j in *atp* – i.e. $(e_j, v_j) \in RPc$ – and then the definition of esa' implies $esa'[e_j, aep] = esa[v_j, atp]$. On the other hand, if e_j is a connector, then $e_j = v_j$ and $esa'[e_j, aep] = esa[v_j, atp]$ also holds. Therefore, Eq. (6.1) can be rewritten as:

$$psa'[aep] = \prod_{1 \le j \le m} esa[v_j, atp]$$

And this is exactly what results from the definition of $psa[atp]$ (Def. 6.3.1.5). □

6.5 Model Representativeness Axioms

In this section we consider the relationship between a model instance and the environment in the real world that the model aims at representing. The intent is to identify the underlying assumptions one must postulate about the *model representativeness*; i.e. the expected capability of the model to accurately reflect the real environment it depicts. Said on another way, our concern here is to determine key aspects in the real world that must be correctly captured by the model, so that the policy refinement process produces a trustworthy result.

To accomplish this goal, the first step is the formalisation of the relevant entities of the real world that are not present in the model. Subsequently, axioms define the required relations between the formalised real entities and modelled objects.

6.5.1 Accesses and their abstract representations

As explained in Sect. 6, the main interest of this validation relies upon examining the ultimate output of the policy refinement process; i.e. the accesses that might take place in the real-world environment. In this respect, the relevant sets are formalised by the following definition.

Definition 6.5.1.1. The real environment comprises the following sets:

- Ac, the set of all potential accesses in the real-world environment that must be considered;
- $EAc \subseteq Ac$, the subset that contains those accesses that are *enabled* by the security system.

The expression *potential access in the real world that must be considered* means a potential communication flow in the real-world environment that starts from an initiator process and is directed to another (responding) process – it is henceforth simply called *access in the real* *environment*, for the sake of conciseness. An access *enabled* by the security system, in turn, indicates a communication flow which is allowed to pass trough all security mechanisms along the network path between the initiator process and the responding process.

The explanation above already anticipates the relation between access in the real world and the lowest model level (ES). Indeed, if an access in the real environment stands for a communication flow between processes, these processes should be represented in some expanded subsystem of the corresponding model instance. Moreover, within the scope of a given access, the initiator process is actually acting on behalf of a user, which is identified in the real environment by means of a *user credential*. The responding process of the same access, on the other hand, is performing some operation on a particular *system object*. As such, user credentials and system objects are also connection points between accesses in the real environment and the model instance. These considerations are formalised by our first axiom as follows.

Axiom 1. Each access in the real environment is associated to a 4-tuple (uc, pc_i, pc_r, so), where $uc \in Uc$ is the user credential associated to the initiator process $pc_i \in Pc$, and $pc_r \in Pc$ is the responding process that in turn handles the system object $so \in So$. Thus, the set Ac can be represented as a relation on the respective ES-level sets:

$$Ac \subseteq Uc \times Pc \times Pc \times So$$

It is worthwhile to remember here that a process in the ES level represents a *prototype* for actual processes that might be started in the real environment (Sect. 4.1.4). As such, the relation established by Axiom 1 between processes and accesses in the real environment imposes that the later will also stand for the *prototype* of actual accesses, or as we put it above, for potential accesses in the system that share some common properties. These common properties are thus each of the 4-tuple dimensions mentioned in Axiom 1.

Hence, the axiom establishes that each access to be considered can be described in terms of system objects in this way: the processor pc_i on behalf of the user credential uc communicates with the process pc_r that manipulates the system object so. But we can just as well describe the access by means of the corresponding abstract entities like this: the user u logged in the system as the subject type st is using the service sv in order to access the resource r. And these two quite different expressions refer to one and the same phenomenon in the real world – an access – tough each one of them at a different abstraction level. Thus, we must establish a means of formalising this correspondence, by connecting accesses in the real environment to abstract entities of the model. This job is accomplished by the following definition.

Definition 6.5.1.2. The correspondence between an access in the real environment and the SR level is denoted by the function $\textsf{abstract-rep} : Ac \rightarrow U \times St \times Sv \times R$. The 4-tuple (u, st, sv, r) returned by the function is the abstract representation of a given access in the set Ac.

The reader may be asking his/herself at this point: why is the access mapped to a representation at the SR level? Why not another model level? This choice is not arbitrary; instead, it is due to the fact that the SR level is the topmost level considered in the validation (see the beginning Section 6). As such, the SR representation offer the reference for the expected system behaviour in the context of the present validation.

The use of a *function* to map this relation is also by no means accidental. The assumption behind it is that for each access in the real environment there is one unique abstract representation at the SR level. Conversely, each 4-tuple (u, st, sv, r) can be the abstract representation of one or more real accesses. Indeed, for each such 4-tuples there will be usually several corresponding potential accesses in the real environment; for an abstract entity represents, in general, a group of similar lower-level objects.

An additional point that is worth mentioning here is that the service dependencies of the SR level (see Sect. 6.3.1) are by purpose let out of analysis in this section and in the validation theorems of the next section. This is done in order to simplify the assumptions and proofs, such that one can more clearly perceive their intent and meaning. Nevertheless, these results and assumptions could be simply changed to cover dependencies by the adding of extra conditions to the statements. Therefore, letting dependencies out does not consist of a limitation for the validation presented here.

From this background, we can reinterpret the set SP of policies in the SR level. Since its elements are 4-tuples in the form (u, st, sv, r) that represent positive authorisation policies, the set can be seen to enclose the abstract representations of those accesses that must be enabled in the real environment. As such, each element of SP must have a corresponding potential access (otherwise the policy could not be enforced); i.e. each service permission must be an abstract representation of some element of Ac. This fact is formalised by the following axiom.

Axiom 2. Each element of SP is the abstract representation of at least one access in the real environment in the set Ac. Formally stated:

$$\forall\ sp \in SP \Rightarrow \exists\ ac \in Ac : sp = \textsf{abstract-rep}[ac]$$

In addition to that, we must now analyse the model structure that reflects the relation of abstract representation defined above. In order to yield valid results, the model structure must correctly represent the correspondence between each access in the real environment and its abstract representation at the SR level. For each 4-tuple (uc, pc_i, pc_r, so) that stands for an access in the real environment, the model structure must thus connect the objects in each dimension of that tuple with the objects of the corresponding abstract representation in the SR level; i.e. with the objects of the corresponding 4-tuple (u, st, sv, r). The connection is established through actor and targets as illustrated in Figure 6.5. Hence, an actor a is connected to both the pair user credential and initiator process (uc, pc_i), and to the pair user and subject type (u, st) – thus establishing the correspondence between them. Analogously, a target t is connected both to the pair responding process and system object (pc_r, so) and to the pair service and resource (sv, r).

Therefore, a well-defined model instance must contain in its structure one such connection between each access and the corresponding abstract representation. This fact is formalised by the following axiom.

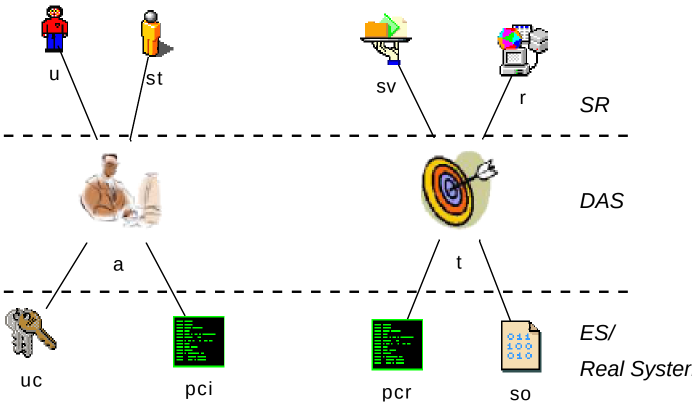


Figure 6.5: Correspondence between accesses in the real environment and the SR level

Axiom 3. The model structure represents correctly the correspondence between accesses in the real environment and their abstract representations. Let $(uc, pc_i, pc_r, so) \in Ac$ be an access in the real environment, such that the 4-tuple (u, st, sv, r) is its abstract representation at the SR level. The model structure must thus contain:

(i) an actor that is connected to both the pair user credential and initiator process (uc, pc_i), and to the pair user and subject type (u, st);

(ii) a target that is connected both to the pair responding process and system object (pc_r, so) and to the pair service and resource (sv, r);

(iii) actor a and target t must be compatible (see Def. 6.3.1.1);

(iv) a host that is connected both to the responding process pc_r and to the system object so.

Formally stated:

$$\begin{aligned}(u, st, sv, r) = \textsf{abstract-rep}\,[(uc, pc_i, pc_r, so)] \Leftrightarrow \\ \exists\, a \in A, t \in T : (a, u, st) \in RA \wedge (t, sv, r) \in RT \wedge atcomp[a, t] \\ \wedge (uc, a, u) \in RUc \wedge (pc_i, a), (pc_r, t) \in RPc \wedge (so, t) \in RSo \wedge\ samehost[pc_r, so]\end{aligned}$$

In order to simplify the notation, the axiom relies upon the predicate *samehost* defined as follows.

Definition 6.5.1.3. The predicate $samehost : Pc \times So \rightarrow \{0, 1\}$ denotes if a process and a system object are connected to the same host. It is defined as:

$$samehost[pc, so] \Leftrightarrow \exists\, h \in H : (h, pc), (h, so) \in W.$$

6.5.2 Authentication and Enabled Accesses

We must now examine the circumstances in which an access is enabled by the system in the real environment. For this purpose, let us first define some predicates in order to reflect the relevant facts in the real environment.

Definition 6.5.2.1. The following predicates map the system behaviour in the real environment:

- $\mathsf{can\text{-}use} : Uc \times Pc \rightarrow \{0, 1\}$ denotes whether a user credential can be used to launch a given process in the real environment. It is defined as:

$$\mathsf{can\text{-}use}[uc, pc] = \begin{cases} 1 & \text{if the credential } uc \text{ can be used to access process } pc; \\ 0 & \text{otherwise.} \end{cases}$$

- $\mathsf{can\text{-}handle} : Pc \times So \rightarrow \{0, 1\}$ denotes whether a process is able to handle a system object in the real environment. It is defined as:

$$\mathsf{can\text{-}handle}[pc, so] = \begin{cases} 1 & \text{if the process } pc \text{ can handle the system object } so; \\ 0 & \text{otherwise.} \end{cases}$$

In respect to the possibility of a user credential to start a process in the real environment, the assumption is that the actors in the model represent all such possibilities; i.e. a given credential is able to start a certain process in the real environment if, and only if, both are connected to the same actor in the model. The following axiom formalises this assumption.

Axiom 4. A credential $uc \in Uc$ is able to start the process $pc \in Pc$ in the real environment if, and only if, there is an actor $a \in A$ in the model such that uc and pc are both connected to a. Formally stated:

$$\mathsf{can\text{-}use}[uc, pc] \Leftrightarrow \exists\ a \in A, u \in U : (uc, a, u) \in RUc \wedge (pc, a) \in RPc.$$

Note that this axiom requires that the system be able to identify and authenticate correctly the user credential, and that the privileges for starting processes be enforced in conformance with the modelled actors. As in most access control models (as pointed out by Sandhu [96]), this work assumes that identification and authentication of users take place in a secure and correct manner, and the main concern is with what happen afterwards.

Analogously, a process is assumed to be able to access a system object if, and only if, both are connected to the same host and to the same target in the model. Here the criterion of the same host has to be introduced to avoid pairs of process and system object which tough connected to the same target, are placed in different machines. The following axiom thus formalises the assumption.

Axiom 5. A process $pc \in Pc$ can manipulate a certain system object $so \in So$ in the real environment if, and only if, both pc and so reside in the same host and are connected to the same target $t \in T$ in the model.

$$\mathsf{can\text{-}handle}[pc, so] \Leftrightarrow \mathit{samehost}[pc, so] \wedge \exists\ t \in T : (pc, t) \in RPc \wedge (so, t) \in RSo.$$

We can now proceed to analyse accesses that are enabled by the system in the real environment. In this respect, the system is assumed to enforce correctly the system view defined at the lowest model level (ES); i.e. the mechanisms of the real-world environment are expected to only permit the passing through of those communication flows that correspond to allowed expanded paths in the model. These assumption requires the correct implementation of the configuration according to the lowest-level policies in the model. It is formalised by the following axiom.

Axiom 6 (Correct Implementation)**.** Suppose the real-world system enables an access $eac \in EAc$, such that $eac = (uc, pc_1, pc_n, so)$. Let $pc_1, \ldots, pc_n$ be the sequence of processes along the communication path between the initiator process pc_1 and the responding process pc_n. The following propositions must hold:

(i) the credential uc can be used to access the process pc_1;

(ii) process pc_n can handle the system object so;

(iii) there is an allowed expanded path $aep \in AEP$ in the model, such that $aep = \langle pc_1, \ldots, pc_n \rangle$; i.e. aep contains exactly the same process sequence $pc_1, \ldots, pc_n$ of eac.

Formally stated:

$$\forall\ eac \in EAc : eac = (uc, pc_1, pc_n, so) \Leftrightarrow \\ \mathsf{can\text{-}use}[uc, pc_1] \wedge \mathsf{can\text{-}handle}[pc_n, so] \wedge \exists\ aep \in AEP : aep = \langle pc_1, \ldots, pc_n \rangle$$

To complete the analysis of an access in the real environment, one must also consider the security class within which the access takes place. This security class depends on the security assumptions of all the processes along the communication flow between the initiator process and the responding process. First, we must then define a function to map these security assumptions as follows.

Definition 6.5.2.2. The function $\mathsf{pc\text{-}esa} : Pc \times Eac \rightarrow SC$ gives the effective security assumption of a process in the context of an enabled access in the real environment.

Subsequently, since Axiom 6 implies that for each enabled access (say eac) there is a corresponding allowed expanded path in the model (say aep), each process along the communication path of eac is assumed to enforce the same security class as that assured by the corresponding process object in the context of aep – i.e. the effective security assumption of Def. 6.4.4.2 in Sect. 6.4.4. This assumption follows from a correct attribution of security classes to objects in the model. The model must be consistent with respect to the classification of model objects, so that the effective security assumption of processes correspond to the security properties of the entities in the real environment. The following axiom formalises this assumption.

Axiom 7. Let $eac \in EAc$ be an enabled access in the real environment, such that $eac = (uc, pc_1, pc_n, so)$ and $pc_1, \ldots, pc_n$ is the sequence of processes along the communication path between the initiator process pc_1 and the responding process pc_n. Assume $aep \in AEP$ to be an allowed expanded path in the model that corresponds to eac; i.e. both eac and aep have exactly the same process sequence $pc_1, \ldots, pc_n$ (Axiom 6 guarantees the existence of such a path). Therefore, each process in the sequence is assumed to enforce the same security class in the real environment as that of the effective security assumption assured by the corresponding process object in the context of aep. Formally stated:

$$\forall\ eac \in EAc, aep \in AEP : eac = (uc, pc_1, pc_n, so), aep = \langle pc_1, \ldots, pc_n \rangle \Rightarrow \\ \textsf{pc-esa}[pc_j, eac] = esa'[pc_j, aep] \quad \text{for } 1 \leq j \leq n$$

Finally, we must consider the security class one can expect of an access as a whole. Analogous to the functions psa and psa' (Defs. 6.3.1.5 and 6.4.4.3), the security assumption of the access must reflect the security class that all the elements along the path are able to enforce. This is denoted by the following function.

Definition 6.5.2.3. The function $\textsf{acc-psa} : Eac \rightarrow SC$ gives the *path security assumption* of an enabled access in the real environment; i.e. the security class that all elements within the scope of an allowed communication flow in the real environment can provide. Let eac be an element of EAc, such that $eac = (uc, pc_1, pc_n, so)$ and $pc_1, \ldots, pc_n$ is the sequence of processes in the communication flow of eac, thus:

$$\textsf{acc-psa}[eac] = \prod_{1 \leq j \leq n} \textsf{pc-esa}[pc_j, eac]$$

6.6 Validation Theorems

With basis on the axioms of the previous section, the validation theorems proposed in Section 6.2 are formalised and proved in the next sections.

6.6.1 Proof of VT1

Validation Theorem 1 (VT1)**.** *For each policy in the SR level, the system enables all accesses in the real environment that correspond to the policy. Formally stated:*

$$\forall\ sp \in SP,\ ac \in Ac : sp = \textsf{abstract-rep}[ac] \Rightarrow\ ac \in EAc$$

Proof. To prove the general theorem's assertion, let us consider, without loss of generality, a service permission sp_1 and its related set of accesses in the real environment A^{sp1}, such that:

$$sp_1 \in SP,\ sp_1 = (u_1, st_1, sv_1, r_1),\ Ac^{sp1} := \{ac \in Ac \mid sp_1 = \textsf{abstract-rep}[ac]\}$$

Axiom 2 implies that Ac^{sp1} is not empty, so let us consider, again without loss of generality, an element ac_1 of Ac^{sp1}, such that $ac_1 \in Ac^{sp1}$, and $ac_1 = (uc_1, pci_1, pcr_1, so_1)$. Our goal is to prove that $ac_1 \in EAc$; i.e. that the access will be enabled by the system. Therefore, as the element were generally chosen, the result can be generalised, and VT1 will be thereby proved.

Since $sp_1 = \textsf{abstract-rep}(ac_1)$, Axiom 3 thus implies:

$$\exists\ a_1 \in A, t_1 \in T : (a_1, u_1, st_1) \in RA \wedge (t_1, sv_1, r_1) \in RT \wedge atcomp[a_1, t_1] \tag{6.2}$$

$$\wedge (uc_1, a_1, u_1) \in RUc \wedge (so_1, t_1) \in RSo \wedge samehost[pc, so] \tag{6.3}$$

$$\wedge\ (pci_1, a_1), (pcr_1, t_1) \in RPc \tag{6.4}$$

By applying Cond. 6.3.1.1 to Eq. (6.2) it thus follows that there is an *ATPermission* between the actor a_1 and target t_1:

$$\exists\ atp_1 \in ATP, atp_1 = \langle a_1, \ldots, t_1 \rangle : (sp_1, atp_1) \in RATP$$

Therefore, Cond. 6.4.1.1 implies that from the expression above follows that:

$$\forall\ w_1 \in Pc,\ w_m \in Pc :\ (w_1, a_1) \in RPc \wedge\ (w_m, t_1) \in RPc \Rightarrow \\ \exists\ aep \in AEP,\ aep = \langle w_1, \ldots, w_n \rangle : expand[aep, atp_1]$$

Hence, considering Eq. (6.4) it results in that the universal quantifier in the previous expression also applies to the processes pci_1 and pcr_1, so that:

$$\exists\ aep_1 \in AEP,\ aep_1 = \langle pci_1, \ldots, pcr_1 \rangle : expand[aep_1, atp_1] \tag{6.5}$$

Axiom 4 implies that from $(uc_1, a_1, u_1) \in RUc$ in Eq. (6.3), and $(pci_1, a_1) \in RPc$ in Eq. (6.4) it thus follows:

$$\textsf{can-use}(uc_1, pci_1) \tag{6.6}$$

Analogously, Axiom 5 implies that from $(so_1, t_1) \in RSo \wedge samehost[pc, so]$ in Eq. (6.3) and $(pcr_1, t_1) \in RPc$ in Eq. (6.4) we can conclude:

$$\textsf{can-handle}(pcr_1, so_1) \tag{6.7}$$

By applying Axiom 6 to Eqs. (6.5), (6.6), and (6.7) it thus comes that $ac_1 \in EAc$; i.e. the access is enabled by the system in the real environment. □

6.6.2 Proof of VT2

Validation Theorem 2 (VT2)**.** *For each enabled access in the real environment there is a corresponding policy at the SR level. Formally stated:*

$$\forall\ ac \in EAc \Rightarrow\ \exists\ sp \in SP : sp = \textsf{abstract-rep}[ac] \wedge sr[sp] \leq \textsf{acc-psa}[ac]$$

Proof. Let us consider, without loss of generality, an enabled access in the real world $ac_1 \in EAc$, such that $ac_1 = (uc_1, pci_1, pcr_1, so_1)$. First, we must prove that the abstract representation of ac_1 pertains to the service permissions in SP. Thereafter, the security class of ac_1 must be shown to comply with the security requirement of the corresponding service permission. In order to demonstrate the first goal, from Axiom 6 we conclude that:

$$\textsf{can-use}[uc_1, pci_1] \tag{6.8}$$

$$\textsf{can-handle}[pcr_1, so_1] \tag{6.9}$$

$$\exists\ aep_1 \in AEP : aep_1 = \langle pci_1, \ldots, pcr_1 \rangle \tag{6.10}$$

By applying Cond. 6.4.1.2 to Eq. (6.10) it follows that a corresponding *ATPermission* must exist in the model:

$$\exists!\ atp_1 \in ATP,\ atp_1 = \langle a_1, \ldots, t_1 \rangle :\ expand[aep_1, atp_1] \tag{6.11}$$

On the other hand, from Eq. (6.8) and Axiom 4 it comes that:

$$\exists\ a \in A, u \in U : (uc_1, a, u) \in RUc \wedge (pci_1, a) \in RPc$$

The definition of the predicate *expand* (Def. 6.4.1.1) and Eq. (6.11) imply that $(pci_1, a_1) \in RPc$. But Cond. 6.4.2.3 asserts that there is only one *Actor* associated to each process, so that the expression above can only be true for $a = a_1$ and $u = u_1$. Thus it can be reformulated as follows.

$$(uc_1, a_1, u_1) \in RUc \wedge (pci_1, a_1) \in RPc \tag{6.12}$$

By applying Cond. 6.4.2.4 to actor a_1 and user u_1 it follows that there must be a related subject type in the model:

$$\exists\ st \in St : (a_1, u_1, st) \in RA \tag{6.13}$$

Considering now Eq. (6.9), Axiom 5 implies that:

$$\exists\ t \in T : (pcr_1, t) \in RPc\ \wedge (so_1, t) \in RSo\ \wedge\ samehost[pcr_1, so_1]$$

Analogously to the actor argument, Def. 6.4.1.1 and Eq. (6.11) imply that $(pcr_1, t_1) \in RPc$. Since Cond. 6.4.2.3 asserts that there is only one *Target* associated to each process, the expression above can only be true for $t = t_1$, so that it can be rewritten as follows.

$$(pcr_1, t_1) \in RPc\ \wedge (so_1, t_1) \in RSo\ \wedge\ samehost[pcr_1, so_1] \tag{6.14}$$

Cond. 6.3.2.2 then asserts that for the *Target* t_1 there are related service and resource objects at the SR level:

$$\exists\ sv \in Sv, r \in R : (t_1, sv, r) \in RT \tag{6.15}$$

Now, by selecting the subject type $st_1 \in St$ to satisfy Eq. (6.13), and the service $sv_1 \in Sv$ and target $t_1 \in T$ from Eq. (6.15) we conclude that:

$$(a_1, u_1, st_1) \in RA \wedge (t_1, sv_1, r_1) \in RT \tag{6.16}$$

By applying Cond. 6.3.1.2 to Eqs. (6.16) and (6.11) it comes that there is a related service permission in the model:

$$\exists\ sp_1 \in SP, sp = (u_1, st_1, sv_1, r_1) : (sp_1, atp_1) \in RATP$$

Therefore, Cond. 6.3.1.3 implies that the pair actor-target of atp_1 is compatible (i.e. $atcomp[a_1, t_1]$). This fact together with Eqs. (6.16), (6.13), (6.12), and (6.14) fulfil all the conditions of Axiom 3. We then conclude that sp_1 is the abstract representation of the enabled access ac_1:

$$sp_1 = \mathsf{abstract\text{-}rep}[ac_1] \quad \blacksquare$$

Only the compliance with the security requirements remains yet to be proved; i.e. the second part of the theorem: $sr[sp_1] \leq \mathsf{acc\text{-}psa}[ac_1]$. In this respect, Def. 6.5.2.3 implies that:

$$\mathsf{acc\text{-}psa}[ac_1] = \prod_{1 \leq j \leq n} \mathsf{pc\text{-}esa}[pc_j, ac_1] \tag{6.17}$$

Where $pc_1, \ldots, pc_n$ is the process sequence of both the communication flow of access ac_1 and the allowed expanded path aep_1 (Axiom 6), with $pc_1 = pci_1$ and $pc_n = pcr_1$. Axiom 7 thus implies that for each process in the sequence:

$$\mathsf{pc\text{-}esa}[pc_j, ac_1] = esa'[pc_j, aep_1] \quad \text{for } j \leq i \leq n$$

By substituting this expression in Eq. (6.17) it results in:

$$\mathsf{acc\text{-}psa}[ac_1] = \prod_{1 \leq j \leq n} esa'[pc_j, aep_1] \tag{6.18}$$

But according to Def. 6.4.4.3:

$$psa'[aep_1] = \prod_{1 \leq i \leq m} esa'[e_i, aep_1]$$

Where $e_1, \ldots, e_m$ is the sequence of elements of aep_1 that pertain to $Pc \cup C$; i.e. only the processes and connectors along the path. If we separate the two types of elements that are considered in this expression, it becomes:

$$psa'[aep_1] = \prod_{1 \leq j \leq n} esa'[pc_j, aep_1] \quad \sqcap \quad \prod_{1 \leq k \leq o} esa'[c_k, aep_1] \tag{6.19}$$

Where $c_1, \ldots, c_o$ is the sequence of connectors in the expanded path aep_1. Thus, since the operator $\sqcap$ always select the minimum values for each security class dimension (see Def. 6.1.0.6 in Sect. 6.1), by comparing Eqs. (6.18) and (6.19) we conclude that $psa'[aep_1]$ is either equal $\mathsf{acc\text{-}psa}[ac_1]$, if all connectors have greater values than the minimum levels among the processes; or otherwise $psa'[aep_1]$ will be less than $\mathsf{acc\text{-}psa}[ac_1]$, that is:

$$psa'[aep_1] \leq \mathsf{acc\text{-}psa}[ac_1] \tag{6.20}$$

On the other hand, Lemma 6.4.4.3 implies that $psa'[aep_1] = psa[atp_1]$. Since Cond. 6.3.1.3 imposes that $sr[sp_1] \leq psa[atp_1]$, from Eq. (6.20) it results in that $sr[sp_1] \leq \mathsf{acc\text{-}psa}[ac_1]$. $\square$

6.7 Validation Soundness

Analysing the two validation theorems demonstrated in the previous section, one can conclude that they ensure the compliance of the policy sets with the validation criteria defined in the beginning of Sect. 6. Indeed, **VT1** implies that the real system enables all accesses in the real world that correspond to the SR level policies specified, such that the *completeness* criterion is thereby satisfied. As for the satisfaction of *consistency*, VT2 asserts that for each possible access in the real world, a corresponding policy exists at the SR level model. These criteria are cited in the literature as the most important to assure the refinement *correctness* in a multi-layered policy hierarchy (see, for instance, [1] and [105]). Indeed, in the particular context of this work, the criteria ensure that the special meaning conveyed by policies and system model—i.e. at the same time representing the positive and negative prescriptions for system duties and for user privileges (see Sect. 5.1.1)—be propagated and maintained from from one abstraction level to its immediate inferior neighbour.

Therefore, the formal proofs mean that: provided a model (including system and policy representation after a process refinement) fulfils the defined condition sets, and this model accurately represents a real-world environment (i.e. the axioms hold), one can conclude that the generated *output*—i.e. the possible system behaviour that results from using the generated configuration—is exactly in conformance with the *input*—i.e. the system view and the policies at the most abstract level considered (SR). As such, the whole system can be seen to act as a reference monitor in the access control terminology [98], which allows only those accesses enabled by the access control policies.

Note that the accesses in the real world are not represented in the model, since the modelling technique used in this work is based on a static picture of the system. As such, policy semantics and system behaviour are not explicitly/formally modelled (as in the work of Burns et al. [16], analysed in Sect. 2.3), but rather implicitly assumed. The model representativeness axioms of Sect. 6.5 thus serve to making explicit the assumptions that are implicit in the modelling, thereby allowing us to draw conclusions about the system functioning behind the model. In this manner, the axioms reflect assumptions concerning both the *correct modelling* of the real world entities, and the *correct implementation* of the normative model elements.

Security goals and requirements, in turn, should be actually seen as a prescription of *modes* in which accesses may take place. They complement the access control policy adding contextual information, having thus a subordinate character, i.e. in isolation, they do not have a meaning for their own, but they only acquire significance together with an access permission by modifying (restricting) the modelled authorisation. As such, they act just like adverbs in the natural language, which are used to imprint a different connotation to verbs and adjectives[2]. Consequently, the security requirements and assumptions in this work are not to be directly compared with clearances and labels of traditional mandatory policies in lattice-based access control models [96], as the latter have a much stronger, independent character (although there is off course some resemblance between the two concepts).

[2] I thank Prof. Krumm for suggesting me this precise and beautiful metaphor.

Furthermore, the requirement support is let flexible enough, so that it is possible to address the needs of different scenarios, depending on how fine-grained and comprehensively one would like to verify the requirement assurance by the system. For instance, a modeller may expand the categories above to encompass all functional classes prescribed by the Common Criteria [22], adjusting the vector dimension and the level range in order to map families and components in each one of that classes. For the purpose of the present work tough, the simplified approach adopted is sufficient.

6.8 Refinement and Validation Analysis

From the standpoint of the practical application, the consistency conditions defined in Sect. 6 can be sorted into three groups. The first condition group is checked on the fly as the modeller defines the objects and associations in the diagram. Each condition in this group reflects a simple relation between objects in the model and it is checked either: a) whenever a object is created; b) whenever a new association between two objects is defined—i.e. a new edge is drawn in the diagram; c) whenever the modeller launches the menu option "Check Model" in the tool. As such, this first group consists of the *pre-conditions* for the algorithm of the respective refinement step (SR/DAS or DAS/ES). Furthermore, since these conditions are tested together with the model input, the computational cost for their verification is insignificant (in comparison with the time that the modeller needs to draw the model).

The second condition group comprises conditions that must be present in the model after the execution of the refinement algorithm in a given step. As such, they are the *post-conditions* for the respective refinement algorithm and thus assumed to be achieved by the algorithm. Consequently, the effort for their "verification" is in fact the computational complexity of the refinement algorithm itself.

The third group is composed by additional conditions that must be verified after the refinement algorithm is performed. Therefore, the computational effort to test the conditions in this third group is actually the real cost of the validation here proposed; i.e. the validation overhead.

In the following sections, each of the conditions defined in Sect. 6 is assigned to one of the three aforementioned groups, and the verification complexity of the conditions of the third group is analysed in detail, as well as the running time for the refinement algorithms. This is performed in two parts, in such a way that each part corresponds to a refinement step through the abstraction layers. In this manner, the next section presents the analysis of the refinement SR/DAS. Subsequently, the conditions and algorithms concerning the refinement DAS/ES are then examined in Sect. 6.8.2. Finally, the scalability of the whole checking process is evaluated in Sect. 6.8.3.

6.8.1 Analysis of the SR/DAS phase

In this phase, the following conditions can be assigned to the first two groups:

- *Pre-conditions*: structural conditions 6.3.2.1, and 6.3.2.2

- *Post-conditions*: refinement conditions 6.3.1.1, 6.3.1.2, 6.3.1.3.

Thereafter, only Condition 6.3.2.3 must be extra verified after the execution of the refinement algorithm of this phase. The checking consists, for each edge of a DAS, of ensuring that this edge is used in at least one of the ATPs in the set generated policy set. As such, if we make the refinement algorithm mark, during its execution, the DAS edges it uses(with no extra time penalty), the verification effort for this condition will be simply to check for each edge whether it is marked—thus linear to the amount of DAS edges, i.e. $O(|E|)$.

As for the refinement algorithm SR/DAS itself, it consists of a variation of the *modified Dijkstra algorithm* for single-source shortest path discovery [23, Ch. 25], including particularities of the DAS representation such as the fact that a path may traverse mediators but not actors and targets (the refinement algorithms are listed in Appendix A). Since a path discovery must be performed for each compatible pair *Actor-Target*, we can overestimate the running time for the algorithm by considering that a search is executed for all pairs of actors and targets in the model. Thus, the algorithm is $O(|A| \cdot |T| \cdot E \text{ lg } V)$, as the cost for each path discovery with the modified Dijkstra algorithm is $O(E \text{ lg } V)$ [23, p. 530].

6.8.2 Analysis of the AS/ES phase

For the AS/ES phase, the algorithm must be in conformance with the following conditions:

- *Pre-conditions*: local structural conditions 6.4.2.1, 6.4.2.2, 6.4.2.3 and 6.4.2.4;

- *Post-conditions*: refinement conditions 6.4.1.1 and 6.4.1.2.

In order to implement the conditions that must be additionally tested—namely, conditions 6.4.2.5, 6.4.2.6, 6.4.3.1, and 6.4.3.2—let us consider an auxiliary data structure called *process connection matrix* $PCM = (x_{ij})$ that consists of a $n \times n$ matrix in which n is the number of processes and each element x_{ij} represents the fact the process pc_i is connected to the process pc_j in the ES model; i.e. $conn[pc_i, pc_j]$. From the definition of the predicate *conn* (Def. 6.4.4.1), one can conclude that the matrix will include processes that are both locally connected and those connected through connectors.

A PCM matrix will be then constructed for each expanded subsystem separately. The first step is to take an ES subgraph for a particular subsystem s and generate an auxiliary graph $ES' = (W', F')$ that encloses all elements of s and additionally also the processes in the adjacent subsystems that are linked through connectors to elements of s. As such, the PCM matrix is in fact a subset of the *transitive closure* of ES' that only contains the processes. It can be thus calculated using the Floyd-Warshall algorithm in time $O(|W'|^3)$ [23, Ch. 26].

Using these auxiliary structures, the verification of the validation conditions is quite straightforward. We must just take each pair of processes in PCM and, if they are connected there must be a corresponding edge in DAS between the related objects (Conds. 6.4.2.6 and 6.4.3.2); otherwise, the edge must not exist—thus satisfying Conds. 6.4.2.5, 6.4.3.1, as all involved elements in DAS must be connect to at least one process from Conds. 6.4.2.1 and 6.4.2.2. Therefore, the cost for this checking is $O(n^2)$, since n^2 lookups for edges in DAS will be performed (which we can consider to be executed in constant time).

The total running time for the validation of one subsystem is determined by the cost of the PCM construction $O(|W'|^3)$, since n is the number of processes which is always a subset of W' (thus $n^2 = o(|W'|^3)$). The process described above must be applied to each subsystem s_i in the system, so that the total time required will be bounded by the sum:

$$\sum |W'_i|^3 , \tag{6.21}$$

where $|W'_i|$ is the number of elements of the auxiliary graph of the subsystem s_i. Notice that the processes of adjacent subsystems must be only added to the auxiliary graph of one of them (i.e. they "flow" in just one direction through connectors), so that the sum above is overestimated.

Let us analyse now the refinement algorithm of this phase. It consists of, for each edge of the ATPs like (v_1, v_2), finding a local path between the ES processes that refine v_1 and v_2. But we can use the same Floyd-Warshall algorithm of the validation above to calculate at the same time a matrix with the local shortest-paths between each pair of processes in a subsystem, with the same asymptotic behaviour as before (in fact, just with a very tight additional constant time, see [23, Ch. 26]). Hence, the path determination is performed with constant time in the refinement phase (just looking at the matrix generated by the validation), so the DAS/ES refinement running time is $O(E)$.

6.8.3 Scalability Improvements

Comparing the two refinement phases previously analysed, we observe that the SR/DAS phase has a global character with respect to subsystems—i.e. paths that span the whole DAS must be found, and the extra condition is related to all edges in the graph—whilst the DAS/ES step is performed locally—i.e. only local paths and local connections are considered, except from the immediate vicinity of the connectors. This asymmetry is due to the fact that, for the DAS/ES step, the theorems and the lemma of Sect. 6.4.4 ensure that the local and composition conditions can be generalised to the whole ES level. Moreover, since the results are proved formally, it is not needed to check them for each particular model instance, but only the local and compositions conditions. This asymmetric character supplies the basis for the scalability gains achieved by this approach, as explained in the following sections.

Scalability on the Refinement

The first type of scalability gain achieved by our approach can be shown by comparing the refinement algorithms. In this respect, the phase SR/DAS is quite costly: $O(|A| \cdot |T| \cdot E \lg V)$, since paths must be discovered in the whole DAS graph (see Sect. 6.8.1). As we discussed in Sect. 6.8.2, the refinement effort for the DAS/ES step is only $O(E)$, as long as results from the validation process are exploited (the validation costs will be analysed later on in the next section). Therefore, the total running time for the refinement is dominated by the costly path discoveries of the SR/DAS phase. However, this path discoveries must only consider the abstract and less numerous DAS elements instead of dealing with all the elements contained in the expanded subsystems.

To make the advantage of the DAS approach clear, let us compare this result with the procedure without using the DAS level, as that employed by previous work on model-based management of security systems like [68, 37]. In this case, the set of service permissions (at the SR level) has to be refined into the PH level, which corresponds to a graph comprising all ESs in our methodology. A path discovery procedure similar to that described in Sect. 6.8.1 is executed, in order to find out the allowed paths between processes, but it must now consider all elements of the detailed view, so that the cost for each path discovery is $O(W \lg F)$ using the modified Dijkstra algorithm. Analogously to Sect. 6.8.1, we can thus estimate the total running time as $O(Pc_a \cdot Pc_t \cdot W \lg F)$, where Pc_a and Pc_t are the number of processes in the system that act as actors and targets, respectively[3].

Therefore, the scalability gain in the refinement process achieved by using a DAS is determined by the following relation:

$$\frac{|A| \cdot |T| \cdot E \lg V}{Pc_a \cdot Pc_t \cdot F \lg W} \tag{6.22}$$

The number of vertices and edges in a DAS, and particularly the number of actors and targets, will heavily depend on intrinsic characteristics of the modelled environment, such as the entities to be modelled and the possibility of subdividing and grouping them, so that we cannot establish a general coefficient from the aforementioned expression. However, observing that each DAS element is usually defined to group various objects in the ES level, it is clear that the refinement in a DAS will consider a smaller amount of elements (this hypothesis is confirmed by the experimental data presented in Sect. 7.3.1).

Scalability on the Validation

The total cost of the validation process is dominated by the local validation DAS/ES (Sect. 6.8.2) which is much more expensive than the simple SR/DAS checking (Sect. 6.8.1)—i.e. quite the opposite situation in contrast with the refinement costs. As in the previous section though, this fact also contributes positively for the scalability of the approach. Indeed, the local consistency conditions are restricted to consider only the elements inside the boundaries of one subsystem in turn—with the exception of the elements of the neighbour subsystems which are directly linked to connectors—so the analysis process can be split up and can be performed independently on each subsystem, improving thereby the scalability of the validation.

In contrast, in the case in which the DAS layer does not exist, the analysis must consider all the elements of the detailed view, so that the verification effort using the same Floyd-Warshall algorithm (i.e. analogously to Sect. 6.8.2) would be bounded by $|W|^3$. To compare this with the total validation cost for the AS/ES validation given by Eq. 6.21, we must observe that $|W|$,

[3] Actually, in [37] the path discovery is performed on an auxiliary graph in which each protocol stack is grouped in a single node. This reduces a little the number of elements that must be considered in the search. Nevertheless, since this reduction affects only protocol objects and do not influence the overall model growth behaviour, I will not take it into account here.

i.e. the number of ES elements, is actually the sum of the elements in each ES $|W'_i|$ (with a small overlap of some processes which is not considered here). Thus the relation between the validation costs with and without the DAS is bounded by:

$$\frac{|W'_1|^3 + |W'_2|^3 + \cdots + |W'_s n|^3}{(|W'_1| + |W'_2| + \cdots + |W'_s|)^3}, \tag{6.23}$$

where s is the number of subsystems in the model. Although the asymptotic behaviour of both terms is cubic, one can perceive from Eq. 6.23 that the more subsystems a model has, the faster is the validation performed in relation to the approach without the DAS intermediary layer. Furthermore, with DAS the analysis of subsystems can be parallelised, achieving even more significant performance gains.

6.9 Chapter Summary

This chapter offers a formal approach to the validation of policy hierarchies that use DAS for the configuration management of network security systems. In the pursuit of this goal, two criteria were selected to ensure the correctness of the refinement: the *completeness* and *consistency* among the different policy sets in the hierarchy. Based on these criteria, a number of condition sets were defined for the refinement phases from the SR level to the Diagram of Abstract Subsystems (DAS), and from the DAS to the Expanded Subsystems. As regards to this last phase, the structural conditions must only concern elements inside the same subsystem, for their validity in the whole level is assured by means of generalisation theorems and a lemma.

Subsequently, *model representativeness axioms* were defined that formalise the implicit assumptions behind the model. Relying upon axioms and conditions, two *validation theorems* were thus proved that ensure the compliance between the resulting system behaviour and the system view and the policies at the most abstract level considered (SR). The meaning and soundness of the validation approach were also discussed in Sect. 6.7. The two theorems correspond to the elected validation criteria of *completeness* and *consistency*, so that once a model complies with the conditions defined and the axioms hold true, the refinement process is *correct*.

Furthermore, the computational effort required by the verification of the defined conditions and by the refinement algorithms have been analysed. The DAS level is expected to bring scalability gains both to the refinement and to the validation.

Chapter 7

Application Examples and Experimental Results

This chapter analyses the practical application of the research approach developed in this work. Firstly, two simple examples are completely explained. Thereafter, a comprehensive case study of a fictitious yet realistic environment is described and experimental results are analysed. The environment is in fact an extension of the simple test scenario of Sect. 4.2, in order to address an enterprise network, composed of a main office and branch office, that is connected to the Internet.

7.1 Simple Examples

Two basic examples are following presented to illustrate the back-end functions implemented in the context of this work: the OpenBSD pf packet filter, and the VPN daemon isakpmd for the same operating system. They are presented in the next sections in turn.

7.1.1 Firewall Example

Figure 7.1 shows a simple firewall model. The RO level allows the employees to surf in the internet, and the security goal and associated requirements are low, such that they have no influence on the refinement process. Thus, the actor "Client" is allowed by the ATPermission to access the target "Webserver" via the mediator "Firewall". The AEP (*allowed expanded path*) is derived from *ATPermission* (see Sect. 5.1), so that the generated configuration for the pf packet filter on the right hand column of Fig. 7.1 reflects the information contained in AEP. In the "VARIABLES" section, the network interface names are associated to aliases that will be later used in the filter rules of section "FILTER". In this section, one rule for each interface is created, in order to allow the traversing of packets from IP `192.168.1.2` to the Webserver with address `192.168.2.2` and port number `80`. The only difference between the two rules is the connection direction: `$interfaceNo1` (em0) must accept the incoming connections (pass in), whereas `$interfaceNo2` (em1) must enable these connections to pass out. Since `pf` has stateful capabilities, i.e. it is a dynamic packet filter (Sect. 2.1.1), the option `keep state` makes the packets in the reverse direction that belong to the same connection to be automatically allowed.

7.1.2 VPN Example

As in the previous example, the intent here is to enable employees to access the company's internal web server. In this case though, the server is located at another network that is connected to the workstations' network via Internet, as shown in Fig. 7.3. The security requirements associated to the service and resource of this example require security levels that are not fulfilled when the communication is performed directly through the Internet (see Sect. 6.3.1), since the security assumption of the AS "Internet" *(1,1,1,1)* is lower than the requirements *(2,2,2,2)*. Thus, the tool automatically chooses the path that connects the main office with the branch office via a VPN tunnel established by the two VPN gateways.

In the case the requirements would be fulfilled without the VPN tunnel—for example, if the requirements were set to (1,1,1,1)—the refinement algorithm would choose the shortest path from actor "intern client" to target "webserver", which bypasses the two VPN gateways. As such, no configuration for the processes "isakmpd 1" and "isakmpd 2" would be generated. The processes will be configured only when there is an allowed communication flow that passes through them.

In the example of Fig. 7.3, the path <*intern Client*, *VPN Gateway 1*, C_{branch}, C_{inet}, C_{main}, *VPN Gateway 2*, *Webserver*> is able to fulfil the requirements due to the Tunnel <*VPN Gateway 1*, C_{branch}, C_{inet}, C_{main}, *VPN Gateway 2*>, which is established by the "EncryptionService".

In order to generate the configuration files from the derived AEP, the properties of the involved objects must be provided as follows.

- For the *Certificate*, the property *distinguished Name* must be set with the name of the *Certificate Authority* that signs the certificate. In the example:

  ```
  /C=de/ST=NRW/L=Dortmund/CN=RootCA
  ```

- The *IP address* used for the communication with the other gateways must be inputted in the corresponding property of the *IPSec* protocol object that is associated to the VPN process. The value for the process "isakmpd 1" in the example is: `192.168.1.1`.

- The *network address* and *network mask* of the network to which the VPN gateway is connected must be provided in the corresponding properties. In the example, for the process "isakmp1" the values are respectively: `10.1.0.0, 255.255.0.0`.

Figure 7.2 shows the configuration file `isakmpd.config` generated for the process "isakmp1". Additionally, the tool generates further information that is presented in the hidden tabs of the configuration in Fig. 7.2: (1) the tab *Read me* contains instructions for activating the VPN gateway; (2) the tab *certify* contains a script that generates certificates; (3) the tab *Policy* sets

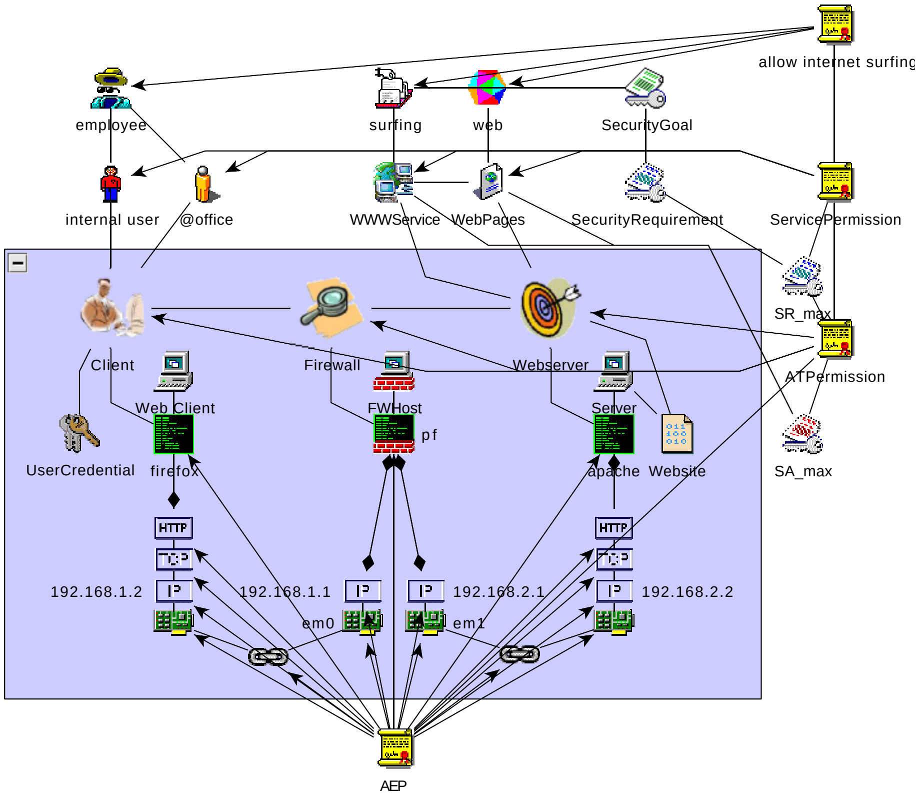


Figure 7.1: Simple firewall model after the refinement

```
# file: pf.conf

### VARIABLES ###########
loopback = "lo0"   # Loopback Device
interfaceNo1 = "em0"
interfaceNo2 = "em1"


### OPTIONS #############
set block-policy return
set debug urgent
scrub in all


### FILTER ##############
pass in quick on $interfaceNo1 proto tcp
↩   from 192.168.1.2 to 192.168.2.2 port
80 ↩   keep state flags S/SA
pass out quick on $interfaceNo2 proto tcp
↩   from 192.168.1.2 to 192.168.2.2 port
80 ↩   keep state flags S/SA
```

the distinguished name; (4) tab *pf.conf* disables all the traffic that do not correspond to the allowed VPN communication to pass through the VPN gateway.

7.2 Case Study

The considered scenario in the case study is that of an enterprise network, composed of a main office and branch office, that is connected to the Internet. Our main goal is to assist the security administrator in the task of designing the configuration for the security mechanisms that are required to enable and control web-surfing and e-mail facilities for the company's office employees. The employees are allowed to use the computers in the main and branch offices in order to surf on the Internet and to access their internal e-mail accounts, as well as to send e-mails to internal and external addresses. In addition to that, they may also retrieve their e-mails from home. As for the ongoing communication from the Internet, the security system must enable any external user to access the corporate's web site and to send e-mails to the internal e-mail accounts.

7.2.1 Network Environment

The network environment considered encompasses the following main segments:

Main Office The company's headquarters, where a pool of 8 *workstations* is available for common use of employees. Furthermore, the company has an internal mail server, a web proxy (for logging of web activity and performance optimisation), and a LDAP directory server that contains information about remote access users;

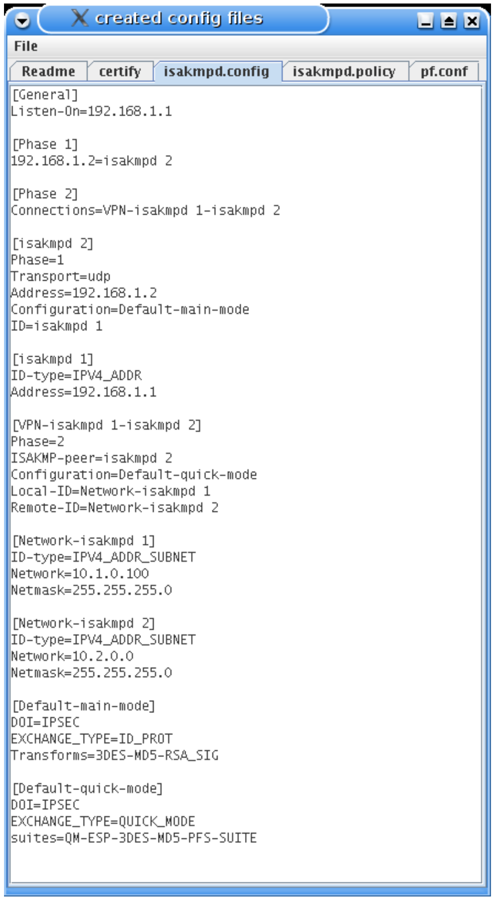
DMZ The *Demilitarised Zone* contains servers that should be accessed from outside. A *Mail Forwarding Server* is used to forward mail from inside to outside and vice versa. A cluster of 5 *webservers* is employed to guarantee availability and performance of the content made available for public access via web. Additionally, a *Radius* server is also placed at the DMZ to allow the employees to access internal network facilities via Internet. Two firewalls are used to control, respectively, the accesses from the internal network to the DMZ, and from this to the Internet.

Branch Office At the branch office, another pool of workstations is available for employees. The branch office network is protected by a simpler *firewall* and a *VPN gateway* that logically melts the branch with the main office.

Remote Access Points Employees can access the internal network from home or elsewhere via Internet. This access shall be performed under a VPN connection.

7.2.2 High-level Policies

The highest-level security policies for this environment can be summarised in the following statements:



```
[General]
Listen-On=192.168.1.1

[Phase 1]
192.168.1.2=isakmpd 2

[Phase 2]
Connections=VPN-isakmpd 1-isakmpd 2

[isakmpd 2]
Phase=1
Transport=udp
Address=192.168.1.2
Configuration=Default-main-mode
ID=isakmpd 1

[isakmpd 1]
ID-type=IPV4_ADDR
Address=192.168.1.1

[VPN-isakmpd 1-isakmpd 2]
Phase=2
ISAKMP-peer=isakmpd 2
Configuration=Default-quick-mode
Local-ID=Network-isakmpd 1
Remote-ID=Network-isakmpd 2

[Network-isakmpd 1]
ID-type=IPV4_ADDR_SUBNET
Network=10.1.0.100
Netmask=255.255.255.0

[Network-isakmpd 2]
ID-type=IPV4_ADDR_SUBNET
Network=10.2.0.0
Netmask=255.255.255.0

[Default-main-mode]
DOI=IPSEC
EXCHANGE_TYPE=ID_PROT
Transforms=3DES-MD5-RSA_SIG

[Default-quick-mode]
DOI=IPSEC
EXCHANGE_TYPE=QUICK_MODE
suites=QM-ESP-3DES-MD5-PFS-SUITE
```

Figure 7.2: The configuration window for *isakmpd 1*

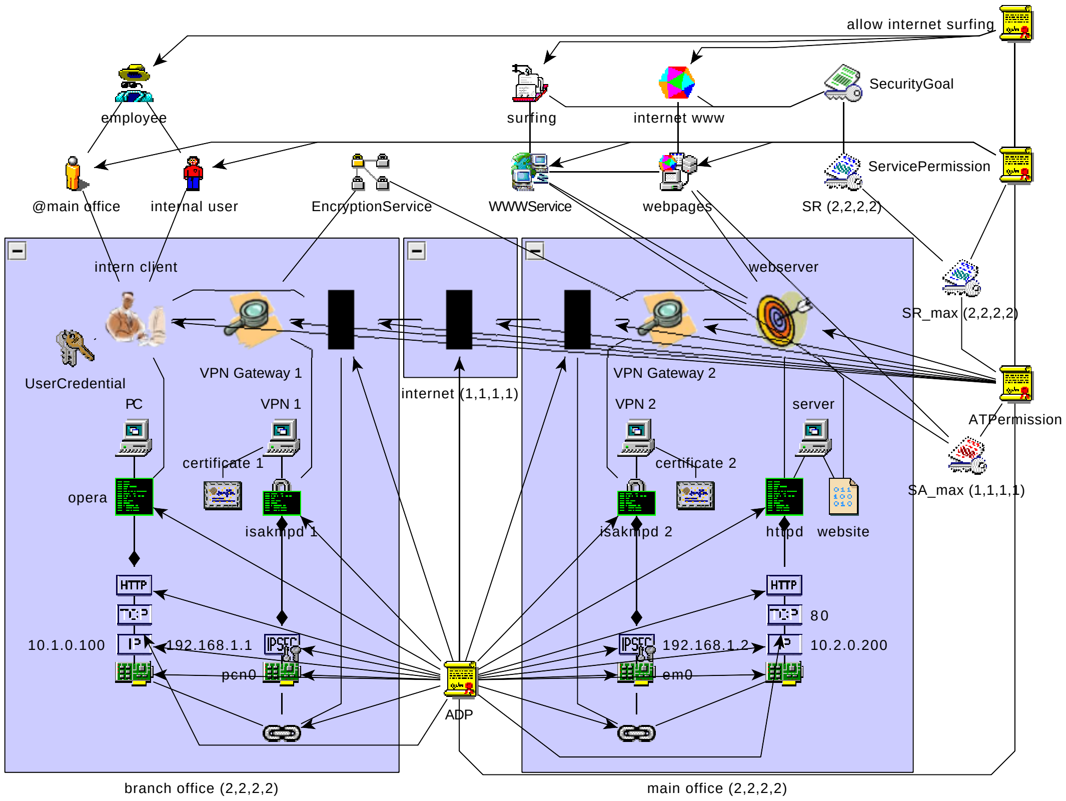


Figure 7.3: Simple VPN model after the refinement

P1: The employees may surf on the Internet from the computers in the main and branch offices;

P2: The employees may read their internal e-mails from the main office, from the branch office, and from home;

P3: The employees may send e-mail both to external and internal addresses;

P4: Users on the Internet may access the corporate web server;

P5: Users on the Internet may send e-mail to internal company's mail addresses.

7.2.3 Incremental Configuration Design Process

Having as input the above abstract policy statements and previously described scenario, the configuration design process evolves through the following steps:

1. modelling of the abstract policy (at RO level of fig. 4.1);
2. definition of the system services and their relations to the abstract elements of the preceding step (SR level of fig. 4.1);
3. modelling of the relevant system elements and their abstract representations (i.e. the Abstract Subsystems) as well as their relations to the objects of the service-oriented system's view;
4. refinement of the high-level policies at the RO level through the other abstraction levels, culminating with the generation of the low-level configuration parameters for the security mechanisms.

This steps can be performed according to the spiral development process [93]: a given subset of system and of the policies is firstly chosen and the four steps above are executed considering only that subset. The development proceeds by iterating the steps in order to consider an increasing part of the system and policies, so that at the end a complete model is thereby produced. In the context of this work, the incremental process can be based on the subdivision into *Abstract Subsystems*, such that the designer starts with a given subsystem and the local policies to it—i.e. the policies for which the *actor*, *target* and *mediators* are enclosed in the considered subsystem. Thereafter, one subsystem is added in turn, together with the policies related to the set of subsystems considered so far, up to reaching a complete model.

The next sections describe the model resulting from this process for the case study environment.

RO level

The top of Figure 7.4 shows the resulting model at the RO level for our considered scenario. The basic objects are: the *Roles* “Employee” and “Anonymous Internet User”, and the *Objects* “Internal e-mail”, “Website”, “Internet e-mail” and “Internet WWW”. These objects are

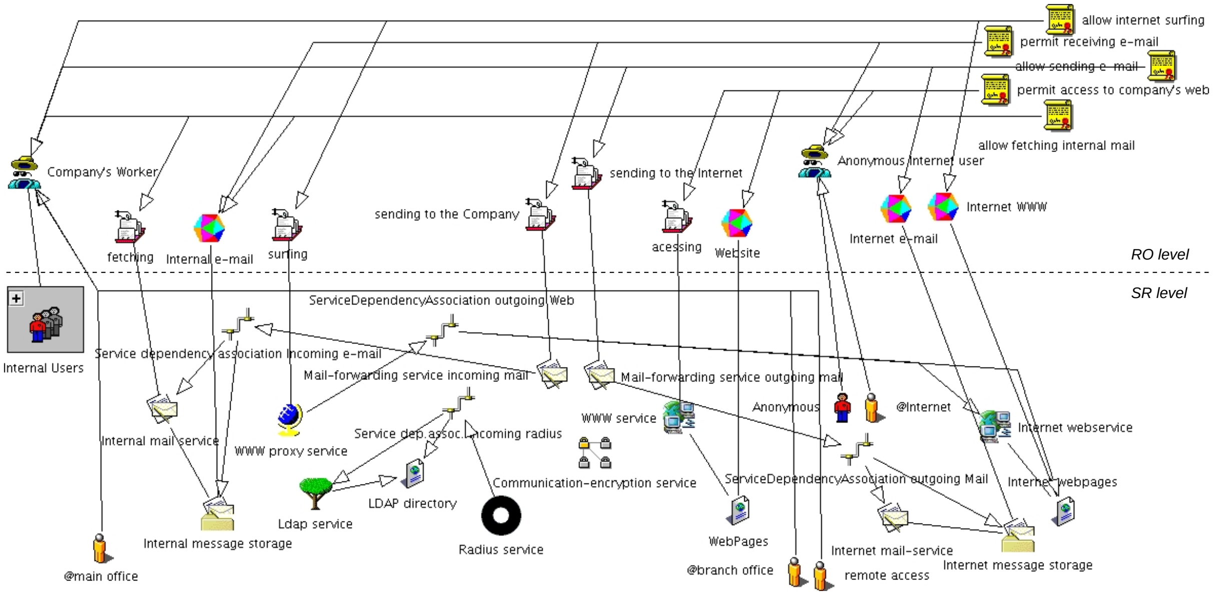


Figure 7.4: Case study model of the levels RO and SR

associated with *AccesModes* by means of five *AccessPermissions* (at the top, on the right of fig. 7.4), each of the latter corresponding to one of the abstract policy statements of the previous section. Thus, for instance, the *AccessPermission* “allow Internet surfing” models the policy statement **P1**, associating the role “Employee” to “surfing” and “Internet WWW”. The other policy statements are analogously modelled by the remaining *AccessPermissions*.

SR level

In our example, the *User* “Anonymous” and the *SubjectType* “@Internet” are defined in association with the role “Anonymous Internet User”. For the role “Employee”, several *User* objects are grouped in the *TypedFolder* “Internal Users” (see Sect. 5.3.1), and three *SubjectTypes* are defined: “@main office”, “@branch office” and “@remote access”. These objects map the three types of session that can be established by an employee in the considered scenario, depending on his physical location.

Regarding the modelling of services and resources, in turn, the *AccessMode* “sending to the company” (RO level) is refined by the *Service*“Mail-forwarding service incoming e-mail”. This service must rely on the “Internal mail service” in order to provide access to the resource “Internal message store” (which is associated with the RO *Object* “Internal e-mail”). This fact is then represented by the connection of the two services to “Dependency incoming e-mail”. Similar situations occur with the services “Mail-forwarding service outgoing” and “WWW Proxy service”, which depend correspondingly on “Internet mail-service” and “Internet web service”. On the other hand, a quite simple situation occurs with the *AccessMode* “accessing” and the *Object* “Website” (RO level); in these cases the service “WWW Service” and the resource “Web pages", which are correspondingly associated to those objects, are directly connected to each other.

Additionally, services and resources are also defined in order to represent more technical features of the system that are not modelled in the RO level. In our example the services "Communication-encryption service", "Radius service" and "LDAP service" fall into this category.

DAS and ES levels

The bottom of Figure 7.5 shows the expanded view of the ASs "internal network" (left) and "dmz" (right). In the latter, there are objects representing hosts and processes for each security mechanism to be used in this segment: two firewalls, a mail forwarder (to avoid exposition of the internal mail server), a radius server (to provide remote access), a VPN gateway and two web servers. Each one of these processes is connected through the adequate protocol stack—modelled by a series of interconnected corresponding protocol objects—to their network interfaces. The host objects representing the mail forwarder and the web servers are additionally connected to objects that map the physical assets they store; i.e. mail files and web pages. The object "UserCredential" represents a group of credentials to which the "X509Certificate" is assigned in order to map the cryptographic certificate required by the VPN process "Encryption Process". The expanded view of the "internal network" follows the same pattern, as do the other ASs that are omitted in Figure 7.5.

Subsequently, the abstract view of each AS must be defined by the creation of objects for Actors, Mediators, Targets and Connectors, and their associations to the objects of the SR level must be established. This is accomplished by classifying the behaviour of the elements of the expanded view into one of these classes; i.e. by recognising which role they play in the security policies.

The *Actors* "internal mail clients" and "internal web clients" (see fig. 7.5) are created in the "internal network" to map the processes of this subsystem with active behaviour in our example (they are the agents of the policies **P1** and **P2** respectively). They are also both connected to the objects in the SR level that map the same behaviour: the *TypedFolder* "Internal Users" and the *SubjectType* "@main office". Due to their intermediary or supporting functions, the *Mediators* "Web proxy" and "LDAP Server" are created and connected to the corresponding processes in the expanded view; they also connect these processes with the appropriate services in the SR level (i.e. the objects "WWW proxy service" and "LDAP Service"). The *Target* "Internal mail server" is then used to represent the passive character of its related objects "Mail Server" and "Internal Mail Files", also associating these to the service "Internal mail service" and the resource "Internal message store" in the SR level. As for the Connectors, they are inserted in order to represent communication possibilities between ASs and thus correspond to the physical interfaces of the actual system.

Proceeding in an analogous manner for all of the remaining ASs in our example, a complete DAS is achieved and the whole system model is complete. Figure 7.6 presents the DAS obtained

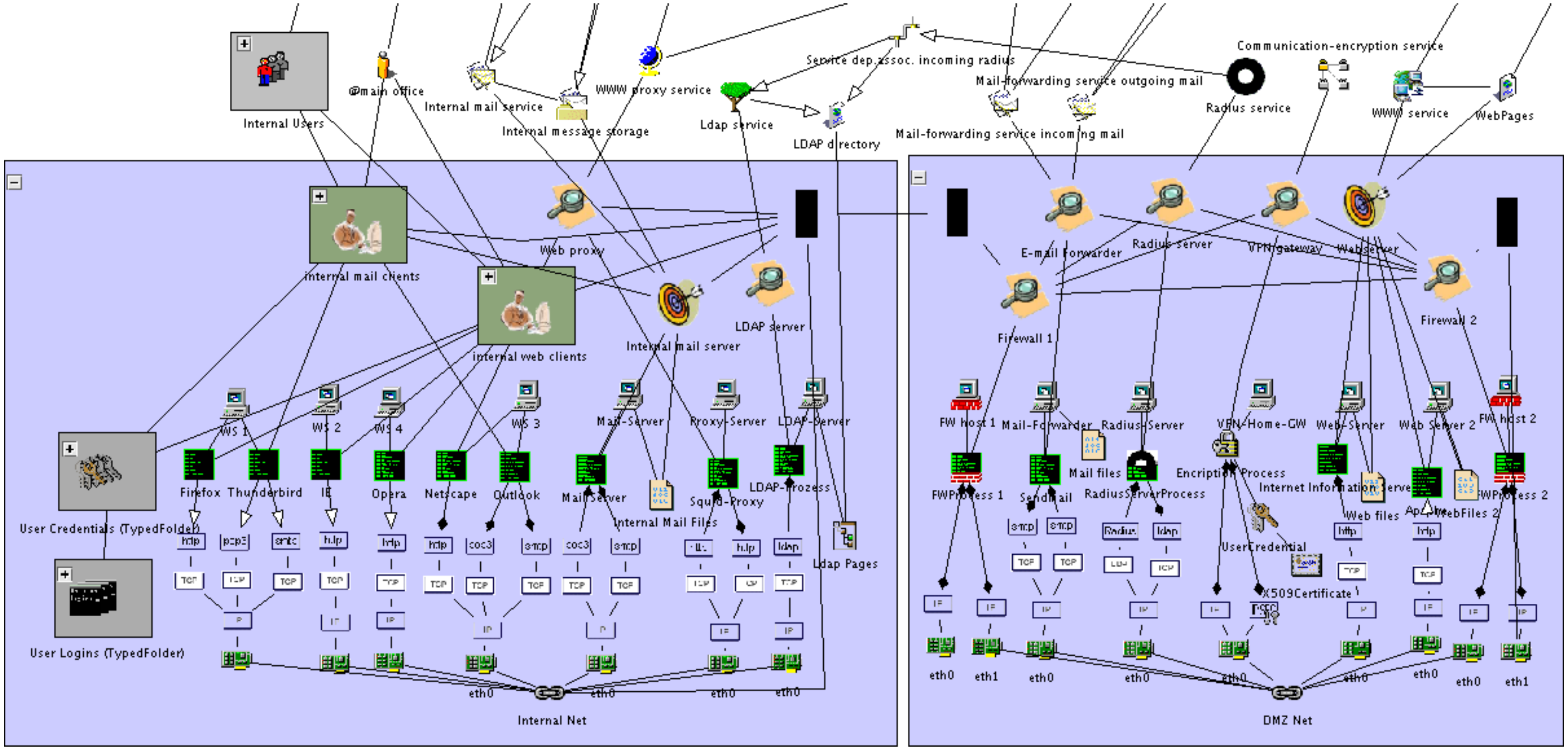


Figure 7.5: Subsystems internal network and dmz from the case study and their relation to the SR level

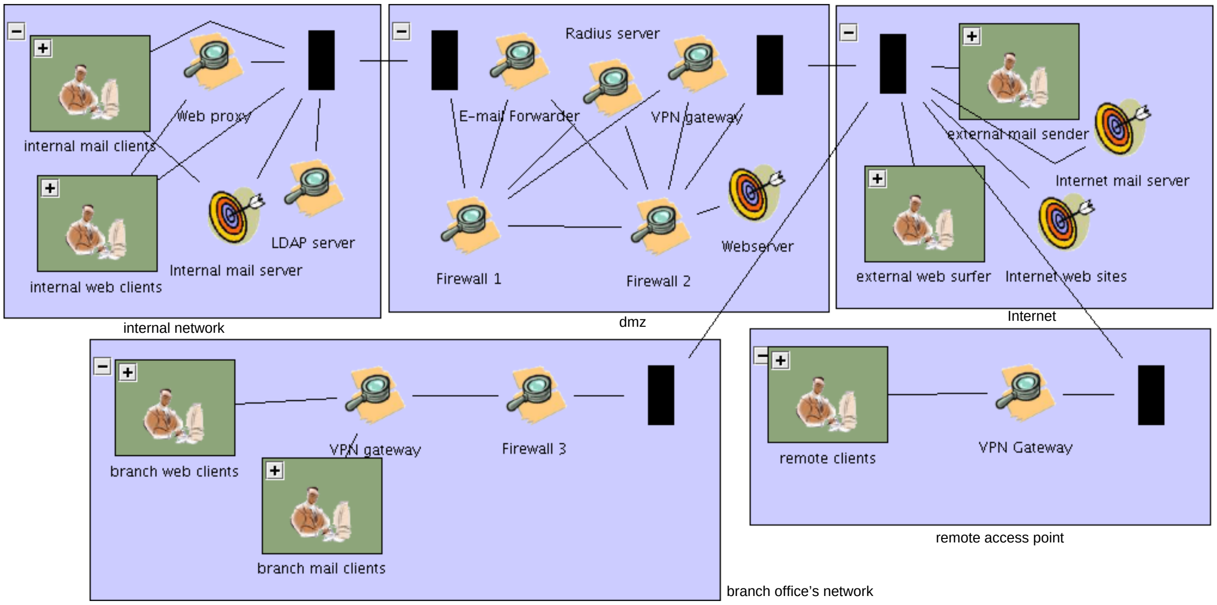


Figure 7.6: DAS for the case study environment

for this environment. It is composed by five ASs, representing the logical network segments of the described scenario: “internal network”, “dmz”, “Internet”, “branch office’s network” and “remote access point”. The *Actors*, *Targets* and *Mediators* of each of these AS and their interconnections through *Connectors* and structural connections are also depicted in Fig. 7.6. Pictures of the model’s overview for the application case and of the*Expanded Subsystems* for the ASs “internal network” and “dmz” are given in the Appendix C.

7.3 Analysis of the Experimental Results

The following sections analyse in turn two types of enhancements achieved by the methodology proposed in this work: the improvements related to the modelling and representation of complex systems, and those improvements concerning the handling of large models.

7.3.1 Representation Improvements

Analysing the model for this realistic application case (Fig. 7.6), one can clearly perceive the advantages brought forth by the DAS level. Although the detailed information of the ES level encompasses more than 500 objects—representing, for instance, 8 hosts and credentials for 30 users in the internal network, 8 hosts in the branch office’s network and a cluster of 5 web servers in the “dmz” (see Appendix C)—the abstract representation of the DAS (Fig. 7.6) consists of only 32 objects altogether. Therefore, it can be noticed that the modelling through abstract subsystems offers concrete advantages in the conciseness and understandability of the model.

Furthermore, since the environment in this application case is an extension of the test scenario used in Sect. 4.2, it is possible to compare them, in order to identify the growth behaviour of a DAS. The number of ES objects (which depict the mechanisms of the actual system, and thus reflect the growth of the system itself) increased from 95 in the simple test scenario to 540 in the realistic application case (i.e. a growth of almost 470%). The number of DAS elements, in contrast, rose from 19 (Fig. 4.8) to 32 (Fig. 7.6)—i.e. a growth of only less than 69%. We thus conclude that the size of a DAS does not increase in the same pace as the number of elements in the ES level (and thus as the system's mechanisms), but rather the DAS's growth is much slower. This makes clear the scalability gain afforded by the DAS in the support of large models.

Since the number of elements both in the DAS and in the ES levels heavily depend on intrinsic characteristics of the environment modelled (such as the entities to be modelled and the possibility of subdividing and grouping them), an unrestricted generalisation of these quantitative results is not possible. Nevertheless, in qualitative terms, similar gains can be expected in the modelling of other large-scale networked environments; for they are similar to the typical scenarios presented here.

7.3.2 Editing, Navigation and Visualisation Improvements

In order to edit specific parts of large models (such as the one used for our case study), a user must rely on model cut-out enlargement techniques; i.e. on zooming. However, with the standard method of zooming that is based on linear enlargement of a fixed-size model cut-out, the model navigation and visualisation are problematic. The problems do not concern the linear enlargement itself, but rather the steps required by the user. In this respect, the benefits brought forth by the fisheye view technique can be best seen by considering the execution of a simple design task. Let us first consider what happens when we use a zoom based on linear enlargement, then subsequently consider the use of a non-linear enlargement and fisheye view.

The task consists of connecting an object in the area of the model that is currently edited to another located at the opposite extreme of a large model. In this case, "large" means that the whole model does not fit into the screen when scaled to a size that makes its editing possible for the user. With the standard zooming method the user needs to perform the following steps:

1. Scale down, in order to be able to see the entire model;

2. Estimate the locations of the target and source objects in the out-zoomed view, and the angle of the edge needed to connect them;

3. Enlarge the area around the source object, in order to be able to select it;

4. Select the source;

5. Drag a new edge from the source into the direction of the previously estimated angle (the enlarged region of the model moves automatically following the mouse);

6. Stop the dragging once the target can be seen on the screen;

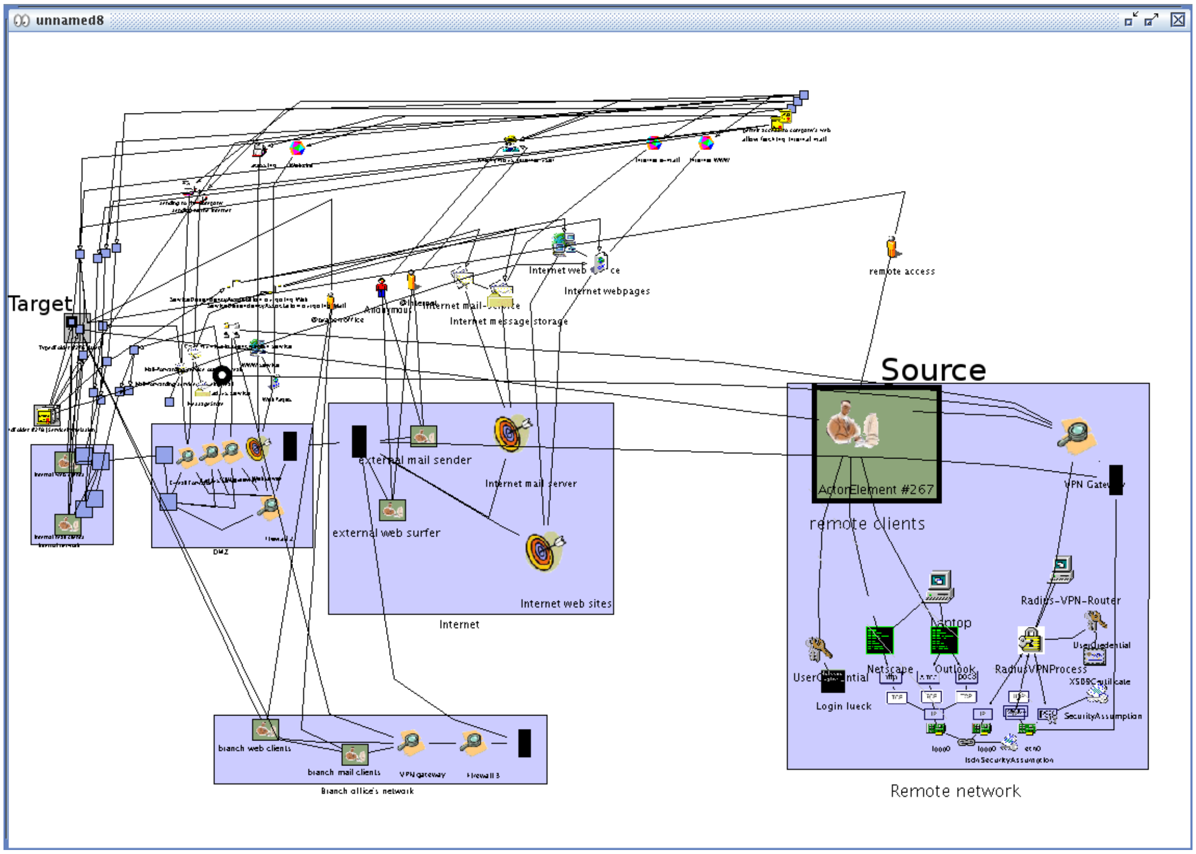


Figure 7.7: Fisheye view with focus centred at the source node

7. Drop the edge on the target object to mark it as the endpoint.

On the other hand, with the use of a fisheye view only the following steps are needed:

1. Activate the fisheye view mode;

2. Select the source object, which is displayed inside the enlarged focus area (Figure 7.7);

3. Drag a new edge from the source into the direction of the target, whose location can be simultaneously seen in the down-scaled surroundings; (the focus follows the mouse during this process, so that the target can eventually be seen in detail);

4. Drop the edge on the target object (the area around the target is enlarged as shown in Figure 7.8).

Therefore, using a fisheye view reduces the number of steps needed by almost half and has an even greater reduction of the amount of time required by the user to accomplish the task. The latter is due to the fact that abrupt switching between the different scaled views requires a certain time for the user to orientate him/herself. Furthermore, the usage is immediately

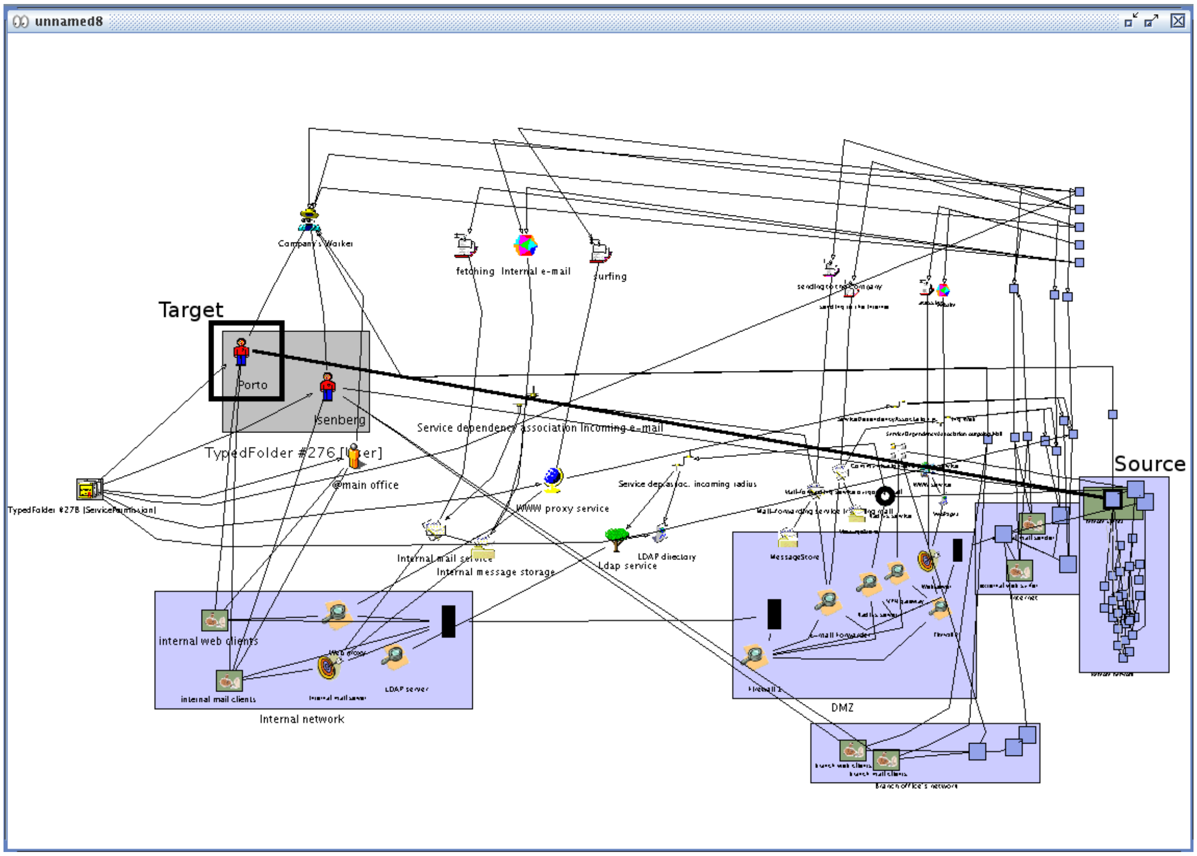


Figure 7.8: Fisheye view with focus centred at the target node

intuitive, since the user never loses of sight the full context of the model and the non-linear adaptive scaling seems "natural" in contrast to the sharp border between linear enlarged cut-outs and their surroundings.

Combining Semantic Zooming and Fisheye View

Since the two focus & context techniques previously introduced (see Sect. 5.3) operate on orthogonal subjects—namely, fisheye view on graphics and semantic zooming on structure—their combined use is reasonable. They do not influence each other and have no side effects that could be detrimental to the use of the other.

We perform this combination by using the scale factor resulting from the fisheye transformation function to set a *level of detail* attribute on each compound element of the model (i.e. ASs and *TypedFolders*). The value of this attribute is then used to adjust the abstraction level which is used to display a given element. The result of this is that the greater the difference between the scale of the focused area and that used to display the element, the higher is the non-linearity in the display of that element. As the distance between the focus and a certain

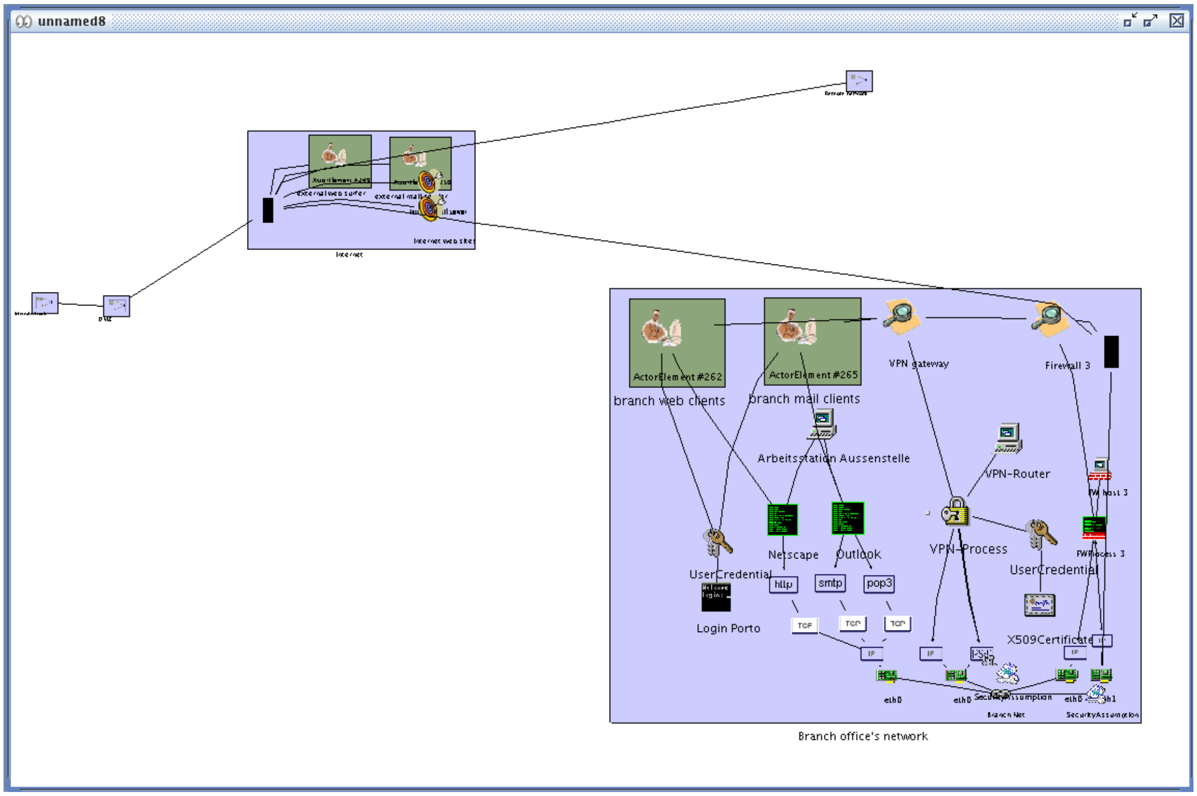


Figure 7.9: Fisheye view in combination with semantic zooming

object increases, this object is gradually represented in a more abstract view—which *per se* has a smaller graphical representation—and also graphically miniaturised by the fisheye view function. Thus, a larger focused area is made possible even if the context is still visible, which leads to an optimisation of the screen space.

Figure 7.9 shows the combined use of the techniques applied to the DAS of the model in our example. The AS "branch office's network" has the focus and is displayed in a higher scale and in its full level of details, allowing the edition of its elements. The AS "Internet" pertains to a close context and is shown in its abstract representation, gradually smaller in size; whereas the miniaturised and "closed" views of the ASs "dmz", "internal network" and "remote access" save screen space while still enabling the user to perceive their existence.

7.4 Chapter Summary

This chapter has explained the practical application of the methodology developed in this work by means of two simple examples and a realistic case study. The examples cover the two mechanisms for which back-end functions were developed in order to generate configuration parameters: the pf packet filter and the VPN daemon isakpmd, both from the OpenBSD operating system.

For the case study, an incremental design process was proposed that is centred around the system administrator. We have followed the application of the four steps this process entails to an example scenario, which consists of a realistic case study.

The experimental results achieved have shown that concrete gains in the understandability and scalability of the modelling of large-scale systems can be expected from the employment of the approached developed in the present work. The scalability was improved both in respect to the representation and to the visualisation and manipulation of large models.

Chapter 8

Conclusion

This work has proposed a design process for the configuration management of network security systems that is especially concerned with large-scale environments, aiming to bridge the gap between high-level security policies and the configuration parameters of mechanisms. This process is based on the model-based management paradigm and relies upon a multi-layered system model that contains different abstraction levels, by means of which the abstract policy representation is gradually brought into a more concrete view of the system.

The next sections discuss successes, partial successes and limitations of the present work and relate them to the goals set out to be pursued (Sect. 3.2).

8.1 Contributions of this work

The main contributions of this research work can be summarised as follows:

- a modelling technique for the management of security systems that achieves scalability by the segmentation of the system in *Abstract Subsystems*, enabling the process of model development and analysis to be performed in a modular fashion;

- tool support for comfortable model editing that implements *focus and context* techniques to improve the handling of large models, allowing the designer to define in detail a certain model part without losing sight of the system as a whole;

- a multi-layered modelling process in which the abstract policy representation is gradually brought into a more concrete system view, bridging the gap between high-level security policies and real system implementations;

- an automated policy refinement process that suits the needs of large-scale, complex environments and derives lower-level policies from the abstract policy set defined by the modeller, eventually yielding configuration parameters for security mechanisms;

- a scalable validation approach for the policy refinement, which includes a formalism of the model, the establishment of consistency conditions, and the proof of the refinement correctness;
- a realistic case study that shows the practicality and the concrete advantages brought forth by this work.

8.1.1 Modelling Framework

The main novelty introduced by this work is the application of the *divide-and-conquer* approach to the modelling of network security systems through the so-called *Abstract Subsystems*. The modelling of a given network segment is thereby subdivided into three steps: (1) the detailed definition of the network mechanisms in the *Expanded Subsystem*; (2) the establishment of an abstract view of those mechanisms by grouping them into *actor*, *target*, and *mediator* objects; (3) the definition of the interaction among abstract subsystems in a *Diagram of Abstract Subsystems* (DAS). As such, the modeller must deal with the mechanisms' complexity only on a local basis for each subsystem—i.e. in steps (1) and (2)—whereas in step (3) the system as a whole is considered by means of an understandable and abstract overall representation of the system's architecture, which directly corresponds to an administrator's mental model of the system. Whilst the first two steps rely upon *modularisation* to address complexity, the third step reflects the *composition* of the modules.

Modularisation has been used to improve programming of large-scale systems since a long time. The modelling framework proposed here builds upon the well-known principle of *information hiding*, in which "every module (...) is characterised by its knowledge of a design decision which it hides from all others. Its interface or definition [is] chosen to reveal as little as possible about its inner workings"—as stated by Parnas in his classical paper *On the Criteria to be Used in Decomposing Systems into Modules* [80]. Indeed, in the present work each *Abstract Subsystem* hides details about its inner features (i.e. network topology, hosts, processes, protocols, etc.), providing instead a policy-oriented view of them which allows the administrator to reason about the distribution of policy participants over the system. Moreover, the sorting out of network objects into *Actors*, *Mediators* and *Targets* fulfils a general user requirement detected by Yurcik et al. [118] while evaluating state-of-the-art security operation tools: the profiling of distinguishable classes of the computers on the network by their activity type.

Therefore, it is my belief that the modelling framework proposed here contributes towards bringing the benefits of modularisation to the management of network security systems. Still according to Parnas's work [80], the benefits expected from modularisation are:

(1) managerial—development time should be shortened because separate groups would work on each module with little need for communication;

(2) product flexibility—it should be possible to make drastic changes to one module without a need to change others;

(3) comprehensibility—it should be possible to study the system one module at a time. The whole system can therefore be better designed because it is better understood.

This work has not directly addressed the first of these benefit types. Though the modelling framework offers the modularisation features required for independent groups to work on the same model, the technical instruments to fully enable the managerial benefits are let as future work. On the other hand, the experimental results gathered in this work show that the modelling with *Abstract Subsystems* do bring benefits of the second and third types. The flexibility gain is exemplified by the incremental development process proposed in Sect. 7.2.3: while working on a particular subsystem, the designer can firstly concentrate only on the elements of the subsystem, and after that define the abstract interactions with the others. The same applies for maintaining the system model, since local changes can be made without altering the global abstract view.

The advantages on the *comprehensibility* mentioned by Parnas can also be clearly seen in the case study analysed in this work. The DAS not only provided a compact and understandable representation of the architecture of a large system, but the abstract representation has been shown to grow at a pace slower than that of network objects (Sect. 7.3.1). Just as in programming though, the benefits of modularisation depend on the characteristics of the problem treated, and on a reasonable dose of the modeller's design competence. However, I hope the examples and discussion in this thesis can serve as parameters for achieving in future models benefits similar to those achieved here.

8.1.2 Graphical Representation

The tool prototype that was implemented also combines the modelling framework developed with *focus & context* techniques, substantially improving the navigation and visualisation of large models. Indeed, the combined use of the techniques *fisheye view* and *semantic zooming* addresses the primary interface requirement of security operators discovered by Yurcik et al. while evaluating state-of-the-art security operation tools: the need for an overall situational awareness of an entire network [118]. Section 7.3.2 shows how these techniques were associated to the modelling framework developed in order to improve the navigation and visualisation of large system models, thereby allowing the designer to define in detail a certain part of the model without losing sight of the system as a whole.

While the graphical system representation developed here provides the administrator with an understandable and scalable view of the system architecture, a partial success has been achieved in the representation of policies. In DAS, policies are represent by allowed paths in the graph from an *actor* to a *target* (*ATPermissions*), allowing the administrator to have a comprehensible view of the allowed network traffic. As the number of allowed path grows though, it is hard to find a reasonable graphical representation that do not overload the screen and makes it not ununderstandable. In this work, the option chosen was to filter out *ATPermissions*, presenting only one path at once—which I think its a step in the right direction. However, it would be nice if the tool could automatically select and present at the same time all the paths that are derived from the same policy of the upper levels (*AccessPermissions* and *ServicePermissions*), enabling the administrator to trace which allowed paths correspond to a certain abstract policy. In the current stage, this is only manually possible. As regards permissions at the ES level (*allowed expanded paths*), the situation is yet more complex, as they are much more numerous. The enhancement of the policy graphical representation is thus let for future research, and some ideas about it are presented later on.

8.1.3 Refinement Algorithms and Validation

This work has successfully developed a policy refinement process appropriate to the needs of large-scale environments. The automated derivation of lower-level policies was proved to be tractable by algorithms with reasonable running times. Moreover, the refinement algorithms developed here are expected to perform faster than the ones used in previous work on MBM, since they must only consider the abstract and thus less numerous DAS elements, instead of dealing with all the elements contained in the expanded subsystems (Sect. 6.8.3).

As mentioned previously, the exact gain attained by the abstract representation depends on idiosyncrasies of the modelled environment, so that it is not possible to achieve general quantitative results in this respect. Nevertheless, the analysis shows that relying upon data from the environments modelled so far—and assuming that this data are significant representatives for other similar environments—the introduction of the DAS layer can be expected to make the refinement process to perform faster and to scale better than the approach without using DAS.

An important accomplishment of the present work is the model formalisation and the establishment of conditions for the *correctness* of the policy refinement; i.e. for ensuring that the generated lower-level policies uphold the abstract policies defined by the user. The criteria of *completeness* and *consistency* were selected, so that the normative meaning conveyed by policies and system model in this work (analysed in Sect. 5.1.1) can be propagated from one abstraction level to its immediate lower neighbour. Based on these criteria, a number of condition sets were established that concern: (i) the relation between elements of policy sets of two adjacent abstraction levels (*refinement conditions*); and (ii) system objects at a given abstraction level as compared both with the system objects at the immediate superior level, and with policies at the same layer (*structural consistency conditions*). Such conditions were defined for the refinement phases from the SR level to the DAS level, and from DAS to the Expanded Subsystems.

The tool checks a group of these validation conditions on-the-fly as model objects are defined and on user's demand whenever she or he chooses the menu option 'Check Model'. A second condition group is tested along with the refinement process, and yet another group is applied after the refinement. These conditions are thus capable of validating the multi-layered structural architecture of the system *vis-à-vis* the security policies prescribed. As a matter of fact, experience has shown that in practice the modelled network systems are frequently not capable of enforcing the given high-level policies. In such cases, the validation conditions defined here cannot be satisfied, and the tool indicates the model elements which violate the policies. In this manner, the modeller receives valuable information to do the necessary modifications on the system in order to make it congruous to the policies.

Furthermore, the computational effort required by the verification of the defined conditions has been analysed. In this respect, the DAS level affords the splitting up of the system analysis, i.e. it enables the validation to be performed in a modular fashion, which is more appropriate when dealing with large models. Indeed, the structural conditions DAS/ES have a local scope and are thus restricted to consider only the elements inside the boundaries of one subsystem in turn, so that the analysis process can be performed independently on each subsystem. The system-wide validity of those local conditions is proved by means of generalisation theorems and a lemma, which do not have to be checked for each particular model instance. Therefore, the more complex a system is, the more beneficial will be the introduction of DAS in the validation cost, since the complexity of all subsystems but one is hidden during the execution of each local test.

In order to be able to reason about the system behaviour that results from the application of the generated configuration, *model representativeness axioms* were also defined that formalise the implicit assumptions behind the model. These axioms state assumptions concerning both the mapping of real world entities by the model objects (*correct modelling*) and the effective application of normative aspects in the model to the real system (*correct implementation*).

Relying upon axioms and conditions, two *validation theorems* have been proved to ensure the compliance between the resulting system behaviour and the system view and the policies at the most abstract level considered (SR). The two theorems directly correspond to the elected validation criteria of *completeness* and *consistency.* Consequently, the formal proofs mean that, provided that a model fulfils the defined condition sets, and that this model accurately reflects a real-world environment (i.e. the axioms hold), the system behaviour enabled by the generated configuration is expected to be exactly in conformance with the system view and the policies at the most abstract level. This is an important result for the reliability of the configuration produced, and for the meaningfulness of the approach in general. It makes sure that the system mechanisms are configured in a proper way, so that the combination of their particular actions does yield the global enforcement of the abstract policies, i.e. that they collectively behave as an integrated system.

8.1.4 Management Process and Implementation

Along with the improvements discussed above concerning the modelling and the graphical representation, the approach used also makes possible a practical management process. This process has a twofold input: (a) a set of high-level policies; and (b) information about the network resources and their use. Having these two elements on hand, the administrator is able to develop the model incrementally, adding one subsystem in turn, up to completing the model. During the model editing, the user is assisted by the tool with the visualisation, handling, and checking features previously discussed. The incremental model development proposed by this work can be used not only in a top-down fashion (as described in detail here) but also bottom-up: sometimes the existence of a network element might be the evidence of the need for an additional policy not yet modelled.

This approach has proved its practical relevance in the case study analysed. It has been successfully employed to the integrated configuration management of packet filters and VPN gateways of a realistic, complex network environment. Back-end functions were implemented for the corresponding mechanisms of the OpenBSD operating system (pf and isakmpd), covering the basic functions of these mechanisms.

It is also anticipated that the modelling effort will be reduced in applications of the technique to other environments, since the models here achieved can be used as templates. After some effort with diverse environments, a library of templates could be gathered and allow a typical modeller only to adapt the expanded view of subsystems to meet the intended network environment, possibly having to add some specific policies to the existing ones.

8.2 Future Work

This work could be extended and enhanced in a number of ways, including the following.

Modelling Features The high-level policy modelling proposed here is based on the simplest *Role-Based Access Control* model that exists. It could thus be extended to enclose features of more comprehensive RBAC models, such as role hierarchies and constraints, and even to incorporate more elaborated extensions of RBAC such as the $UCON_{ABC}$ usage control model [79] and the Generalised-RBAC (GRBAC) [24]. The SIRENA project [47, 45] shows that the MBM approach can be profitably used with the GRBAC to address requirements of dynamic environment conditions. The combination of these extensions with the scalable modelling framework proposed here could thus yield interesting results.

Policy Representation As mentioned in the previous section, the representation of policies at the lower levels of the model could be improved. Since these policies are generated automatically by the tool, a promising direction to investigate is the grouping of lower-level policies based on the high-level policies they refine. I think an automated navigation mechanism that allows the administrator to track the effect on the network traffic caused by a given high-level policy would be a most valuable resource. This should not be difficult to achieve from the current stage of the tool, although its execution offers interesting challenges.

Usability Currently there is a growing concern about bringing nearer the research communities of security and human-computer interfaces (HCI). I believe that this is a very promising research direction, and that the present work makes a step into the direction of making the security management closer to the human administrator. Indeed, the models used here are intuitive and directly match the administrators' system mental model. An interesting extension of the present work would be to further investigate this in an inter-disciplinary approach, by performing usability studies to identify the needs of administrators, and by running tests in order to evaluate our prototype tool and approach.

Topology Discovery The gathering of information about network devices, links, and workstations could be partially automated by interrogating the system relying upon management protocols such as SNMP. This would improve the modelling by eliminating possible errors in this respect, and would reduce the modeller's effort, which is especially interesting in large-scale environments when combined with good lay-out algorithms for producing the final graphical representation.

Coverage The approach here presented could be extended to cover other functionalities of firewalls, VPNs, and other security mechanisms, such that they would be simultaneously and uniformly managed with the modelling framework proposed. Previous work on Model-Based Management indicates that the approach is able to cover both more firewall functionalities (e.g. *Network Address Translation*) and other mechanisms (e.g. Kerberos, web proxies). In fact, since the modelling framework represents policies at the lowest-level as allowed network flows, it should be quite straightforward to convert the flows into configuration parameters for most security services which perform some sort of access control based on certain communication characteristics.

Chapter 9

Conclusão

Este trabalho propôs uma abordagem para o gerenciamento da configuração de sistemas de segurança de redes que é especialmente concebido para suprir as necessidades de ambientes complexos e de larga escala. O processo se pauta pelo paradigma do gerenciamento baseado em modelos, utilizando um modelo em camadas do sistema que contém diferentes níveis de abstração, e através do qual uma representação em alto nível da política de segurança é gradualmente vertida em uma visão mais concreta do sistema. Dessa forma, preenche-se a lacuna existente atualmente entre as políticas de segurança de uma organização expressas em alto nível e os parâmetros de configuração dos mecanismos.

9.1 Principais Contribuições

Os principais resultados alcançados no conjunto dos trabalhos que integram esta pesquisa podem ser sumarizados nos seguintes pontos:

- desenvolvimento de uma técnica de modelagem para o gerenciamento de sistemas de segurança, a qual obtém escalabilidade mediante a segmentação do sistema em subsistemas abstratos (*Abstract Subsystems*), possibilitanto que os processos de desenvolvimento e análise sejam executados de maneira modular;
- implementação de um protótipo de ferramenta gráfica para edição de modelos, a qual incorpora técnicas de *foco e contexto* para a melhorar o tratamento de grandes modelos, permitindo que o projetista defina em detalhes uma certa parte do modelo sem perder de vista o sistema como um todo;
- um processo automático de refinamento de políticas adequado às necessidades de ambientes complexos de larga escala, o qual deriva políticas de mais baixo nível a partir do conjunto de políticas em alto nível definidas pelo usuário, produzindo afinal parâmetros de configuração para mecanismos de segurança;

- uma abordagem escalável para a validação do refinamento de políticas, que inclui a formalização do modelo, o estabelecimento de condições de consistência, e a prova formal da corretude do refinamento;
- um processo de modelagem em camadas, no qual uma representação abstrata das políticas é vertida gradualmente um uma visão mais concreta do sistema, preenchendo assim a lacuna entre políticas de segurança de alto nível e as configurações de mecanismos de segurança nos sistemas reais;
- um estudo de caso realista que evidencia a relevância prática e as vantagens concretas proporcionadas por este trabalho.

As próximas seções discutem resumidamente os sucessos totais ou parciais e as limitações dos resultados obtidos no presente trabalho.

9.1.1 Técnica de Modelagem

A principal novidade introduzida por este trabalho é a aplicação do princípio de divisão e conquista (*divide et impera*) na modelagem de sistemas de segurança de redes, através dos assim chamados *Subsistemas Abstratos* (*Abstract Subsystems*). A modelagem de um certo segmento de rede é então subdividida em três etapas: (1) a definição detalhada dos mecanismos de rede nos *Subsistemas Expandidos*; (2) o estabelecimento de uma visão abstrata daqueles mecanismos, agrupando-os em objetos dos tipos *actor*, *target* e *mediator*; (3) a definição d interação entre subsistemas em um *Diagrama de Subsistemas Abstratos* (DAS na sigla em inglês). Dessa forma, o usuário lida com a complexidade dos mecanismos apenas localmente em cada subsistema — isto é, nos passos (1) e (2) — enquanto que no passo (3) o sistema como um todo é apreciado mediante uma representação geral compreensível e abstrata de sua arquitetura, a qual corresponde diretamente ao modelo mental que o administrador tem do sistema. Enquanto que as duas primeiras etapas empregam *modularização* para lidar com a complexidade, a terceira corresponde à composição dos módulos — técnicas tradicionais usadas na programação de sistemas de larga escala.

Assim, o presente trabalho traz benefícios da modularização para o gerenciamento de sistemas de segurança de redes — particularmente, para o projeto da configuração de mecanismos —, tais como:

Flexibilidade No processo de desenvolvimento incremental (ver Seção 7.2.3), o projetista pode concentrar-se primeiramente apenas nos elementos de um subsistema em particular, e depois definir a interação deste subsistema com os outros de maneira abstrata. O mesmo se aplica para a manutenção posterior do modelo, pois mudanças locais podem ser implmentadas sem alterar a visão abstrata do sistema todo.

Compreensibilidade O nível DAS provê uma representação compacta e inteligível da arquitetura de um sistema de grande porte, que, além disso, provou ter crescimento assintótico mais lento do que a taxa de crescimento dos objetos de rede (Seção 7.3.1).

Não obstante, assim como em programação, os benefícios da modularização dependem de características do problema tratado, e em uma boa dose de competência de *design* por parte do projetista. Esse trabalho espera, entretanto, ter oferecido em seus exemplos e discussões parâmetros para que se obtenha benefícios similares àqueles obtidos aqui em futuros modelos.

9.1.2 Representação Gráfica e Protótipo

A ferramenta protótipo implementada combina a técnica de modelagem desenvolvida com técnicas de *foco e contexto*, melhorando substancialmente a navegação e a visualização de grandes modelos. Com efeito, o uso combinado das técnicas *fisheye view* (visão olho-de-peixe) e *semantic zooming* (zoom semântico) e da técnica de modelagem aqui apresentada permite que o projetista defina em detalhes uma certa parte do sistema sem perder de vista a arquitetura do sistema como um todo, ou seja, mantendo-se consciente do contexto maior em que está trabalhando. Ao término do desenho do sistema, a ferramenta executa, então, um processo de refinamento que produz automaticamente arquivos de configuração para os mecanismos de segurança.

Enquanto que a representação gráfica desenvolvida oferece ao administrador uma visão escalável e compreensível do sistema, em relação a políticas um sucesso parcial foi alcançado. No DAS as políticas são representadas por caminhos no grafo (*ATPermissions*) — cada um dois quais autoriza um *actor* a acessar um *target* —, proporcionando ao administrador uma representação compreensível do tráfego de rede permitido. No entanto, na medida em que o número de caminhos permitidos cresce, torna-se difícil encontrar uma representação gráfica razoável, ou seja, que não sobrecarregue a tela e que se mantenha inteligível. Neste trabalho, a opção escolhida foi filtrar *ATPermissions*, apresentando apenas um caminho por vez — o que nos parece ser um passo na direção correta. Contudo, seria interessante se a ferramenta pudesse seleciona automaticamentee apresentar ao mesmo tempo todos os caminhos que são derivados de uma mesma política de nível mais alto (*AccessPermissions* ou *ServicePermissions*), possibilitando ao administrador rastrear que caminhos permitidos na rede correspondem a certas políticas abstratas. No estado atual de implementação isso é possível, porém apenas manualmente. Em relação a permissões no nível ES (*allowed expanded paths*), a situação é ainda mais complexa, pois eles tendem a ser mais numerosos. Um aperfeiçoamento da representação gráfica de políticas é deixado, portanto, para trabalhos futuros, e algumas idéias a esse respeito são apresentadas adiante (Seção 9.2).

9.1.3 Algoritmos de Refinamento e Validação

O presente trabalho desenvolveu um processo de refinamento de políticas apropriado às necessidades de ambientes de rede de larga escala. A derivação automática de políticas de nível mais baixo provou ser tratável por algoritmos com tempo de execução razoável. Além disso, espera-se que os algoritmos de refinamento desenvolvidos tenham desempenho melhor do que aqueles usados em trabalhos anteriores em gerenciamento baseado em modelos. Isso se deve ao fato de que, no presente trabalho, os algoritmos necessitam apenas considerar os elementos abstratos, e por isso mesmo menos numerosos, do nível DAS, ao invés de lidar com todos os elementos contidos nos subsistemas expandidos (Seção 6.8.3).

Como mencionado anteriormente, o ganho exato advindo da técnica de modelagem depende das idiossincrasias do ambiente modelado, de forma que não se pode obter resultados quantitativos gerais a esse respeito. Entretanto, a análise mostra que, com base em dados dos ambientes modelados até o momento — e assumindo que esses dados provêem de representantes significativos para outros ambientes similares —, pode-se esperar que a introdução do nível DAS faça com que o processo de refinamento seja executado mais rapidamente do que a abordagem sem usar o DAS, e tenha seu desempenho menos afetado pelo crescimento do sistema, isto é, tenha maior escalabilidade.

Um resultado importante obtido por este trabalho é a formalização do modelo e o estabelecimento de condições para a *corretude* do refinamento de políticas, ou seja, para assegurar que as políticas de baixo nível geradas sejam conformes às políticas abstratas definidas pelo usuário. Os critérios de *completude* e *consistência* foram selecionados, de forma que o significado normativo adquirido por políticas e pelo modelo do sistema neste trabalho (analisados na Seção 5.1.1) possa ser propagado de um nível de abstração para seu vizinho imediatamente inferior. Baseado nesses critérios, grupos de condições foram estabelecidas concernentes: (i) à relação entre elementos de conjuntos de políticas de dois níveis de abstração adjacentes (condições de refinamento); e (ii) à comparação entre objetos do sistema frente aos objetos de sistema no nível superior, assim como frente às políticas do mesmo nível (condições de consistência estrutural). Tais condições foram definidas para as fases de refinamento do nível SR para o DAS, e deste para os *Subsistemas Expandidos* do nível ES.

A ferramenta verifica algumas dessas condições *on-the-fly*, à medida que os objetos do modelo são definidos, e sob demanda do usuário, sempre que este aciona a opção correspondente no menu. Um segundo grupo de condições é testado durante o processo de refinamento, enquanto que um último grupo é aplicado depois do refinamento. Essas condições são, assim, capazes de validar a arquitetura em camadas do sistema *vis-à-vis* as políticas de segurança prescritas. Na verdade, a experiência tem mostrado que na prática os sistemas modelados são freqüentemente incapazes de assegurar o cumprimento das políticas de alto nível definidas. Nesses casos, as condições de validação definidas não são satisfeitas e a ferramenta indica os elementos do modelo que violam as políticas. Dessa forma, o projetista recebem nesse processo informações valiosas sobre as modificações necessárias para tornar o sistema congruente com as políticas.

Além disso, o esforço computacional requerido para verificação das condições foi analisado. Nesse respeito, a camada DAS proporciona uma segmentação da análise do sistema, isto é, ela possibilita que a validação seja executada de forma modular — a qual é mais apropriada para lidar com grandes modelos. De fato, as condições estruturais da relação entre os níveis DAS/ES possuem escopo local, restringindo-se portanto a considerar apenas os elementos contidos nas fronteiras de um subsistema por vez (com exceção apenas daqueles diretamente relacionados a conectores), de forma que o processo de análise pode ser realizado independentemente para cada subsistema. A generalização da validade dessas condições locais para o sistema todo foi provada através de teoremas e lemas, os quais obviamente não necessitam ser testados para cada instância de modelo em particular. Por conseguinte, quanto mais complexo for um sistema, maior será o benefício advindo da introdução do DAS no custo total da validação, pois durante a aplicação de um teste local em um subsistema em particular, a complexidade de todos os demais subsistemas é desconsiderada.

Com o intuito de poder analizar o comportamento do sistema resultante da aplicação da configuração gerada, axiomas sobre a representatividade do modelo (*model representativeness axioms*) foram definidos para formalizar as assunções implícitas. Esses axiomas refletem premissas concernentes tanto ao mapeamento de entidades no mundo real por objetos do modelo (modelagem correta), quanto à efetiva aplicação dos aspectos normativos do modelo no sistema real (implementação correta). Com base nos axiomas e nas condições, dois teoremas foram provados para assegurar a conformidade do comportamento resultante do sistema com a visão do sistema e as políticas do nível mais abstrato considerado (nível SR). Os dois teoremas correspondem diretamente ao critérios de validação escohlidos, a saber *completude* e *consistência*. Logo, as provas formais significam que, dado que um modelo satisfaça as condições definidas e que esse modelo reflita corretamente o ambiente do mundo real (ou seja, que os axiomas valham), pode-se esperar que o comportamento do sistema possibilitado pela configuração gerada esteja em plena conformidade com a visão do sistema e as políticas no nível mais alto de abstração. Este resultado é importante para a confiabilidade das configurações produzidas e para a significância da abordagem em geral, pois ele garante que os mecanismos do sistema sejam configurados de maneira adequada. Assim, a combinação de das ações de cada mecanismo em particular resultará efetivamente na aplicação global das políticas de segurança abstratas, ou seja, os mecanismos agirão coletivamente como um verdadeiro sistema integrado.

9.1.4 Processo de Gerenciamento e Implementação

Além das vantagens concernentes à modelagem e à representação gráfica que foram discutidas acima, a abordagem utilizada possibilita também um processo prático de gerenciamento. Esse processo como entrada: (a) um conjunto de políticas de alto nível; e (b) informações sobre os recursos de rede e seu uso. De posse desses dois elementos, o administrador pode desenvolver o modelo de maneira incremental, adicionando um subsistema por vez até completar o modelo. Durante a edição do modelo, o usuário é assistido pela ferramenta com as capacidades de visualização, manipulação e verificação descritas previamente.

Essa abordagem teve sua relevância prática confirmada no estudo de caso analizado. Ela foi aplicada com sucesso para o gerenciamento integrado das configurações de filtros de pacote e VPNs (redes privadas virtuais, na sigla em inglês) em um ambiente de rede realista e complexo. Módulos *backend* especiais foram implementados para gerar parâmetros de configuraçao para os mecanismos correspondentes no sistema operacional OpenBSD (pf e isakpmd), cobrindo as funcionalidades básicas desses mecanismos.

9.2 Trabalhos Futuros

Este trabalho pode ser estendido e aperfeiçoado de diversas formas, incluindo as seguintes.

Modelagem A modelagem de políticas de alto nível proposta aqui se baseia no mais simples modelo de Controle de Acesso Baseado em Papéis (*Role-Based Access Control*, RBAC) existente. Ela poderia, então, ser estendida para abarcar características de modelos mais abrangentes, como hierarquias de papéis e restrições (*constraints*), e até mesmo para incorporar extensões mais elaboradas do RBAC como o modelo de controle de utilização $UCON_{ABC}$ [79] e o *Generalised-RBAC* (GRBAC) [24]. O projeto SIRENA [47, 45] mostra que o paradigma MBM pode ser empregado em conjunto com o GRBAC para lidar com requerimentos de condições de ambiente dinâmicas. A combinação dessas extensões com a modelagem escalável proposta no presente trabalho poderá produzir resultados interesssantes.

Representação de Políticas Como mencionado na seção anterior, a representação de políticas nos níveis inferiores do modelo pode ser aperfeiçoada. Como essas políticas são geradas automaticamente pela ferramenta, uma direção de investigação promissora é o agrupamente de políticas de mais baixo nível com base nas políticas de alto nível que as primeiras refinam. Um mecanismo de navegação automático que permita o admistrador rastrear o efeito no tráfego de rede causado por uma certa política de alto nível seria um recurso de grande valia. Isso não deve ser de difícil obtenção a partir o estado atual da ferramenta, embora sua execução ofereça desafios interessantes.

Usabilidade Atualmente há uma preocupação crescente em aproximar as comunidades de pesquisa de segurança e de interfaces humano-computador. Essa é uma direção de pesquisa muito promissore, e o presente trabalho dá um passo na direção de torna o gerenciamento da segurança mais próximo do administrador humano. Com efeito, os modelos desenvolvidos aqui são intuitivos e correspondem diretamente ao modelo mental que administradores têm do sistema. Uma possível extensão do presente trabalho poderia investigar essa relação em uma abordagem interdisciplinar, através de estudos de usabilidade para identificar as necessidades de administradores e avaliar a ferramenta protótipo e a abordagem.

Topologia A obtenção de informações sobre dispositivos de rede, conexões e máquinas pode ser parcialmente automatizada, mediante buscas na rede utilizando protocolos de gerenciamento como o SNMP. Além de reduzir o esforço do projetista, essa automatização eliminaria a possibilidade de erros na modelagem. Isso seria especialmente produtivo em ambientes de larga escala, desde que combinado com bons algoritmos de *layout* automátco para produzir a representação gráfica final.

Cobertura A abordagem proposta pode ser estendida para cobrir outras funcionalidades de *firewalls* e VPNs e outros mecanismos de segurança, de forma que fossem eles gerenciados simultaneamente e uniformemente através da modelagem desenvolvida aqui. Trabalhos anteriores em Gerenciamento Baseado em Modelos (MBM) indicam que a abordagem é capaz de cobrir não somente outras funções de firewalls (e.g. *Network Address Translation* em [37]), como também outros mecanismos de segurança (e.g. Kerberos em [94]). Na verdade, como a técnica de modelagem representa políticas no nível mais baixo em forma de fluxos de rede permitidos, deveria ser relativamente fácil converter esses fluxos em parâmetros de configuração para a maioria dos serviços de segurança que executam algum tipo de controle de acesso com base em certas características da comunicação.

Bibliography

[1] M. Abrams and D. Bailey. Abstraction and refinement of layered security policy. In M. Abrams, S. Jajodia, and H. Podell, editors, *Information Security : An Integrated Collection of Essays*, pages 126–136. IEEE Computer Society Press, Los Alamitos, California, USA, 1994.

[2] A. Adams and M. A. Sasse. Users are not the enemy. *Communications of ACM*, 42(12):40–46, 1999.

[3] R. J. Anderson. Why cryptosystems fail. *Communications of ACM*, 37(11):32–40, 1994.

[4] S. Anderson-Redick, et al. *System Policy Editor*. O'Reilly and Associates, 2000.

[5] Associação Brasileira de Normas Técnicas, Rio de Janeiro. *NBR ISO/IEC 17799:2000, Tecnologia da informação—Código de prática para a gestão da segurança da informação*, 2000.

[6] Y. Bartal, A. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. In *Proc. 20th IEEE Computer Society Symposium on Security and Privacy*, 1999.

[7] Y. Bartal, A. J. Mayer, K. Nissim, and A. Wool. Firmato: A novel firewall management toolkit. *ACM Transactions on Computer Systems*, 22(4):381–420, November 2004.

[8] D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations and model. Technical Report M74–244, MITRE Corporation, Bedford, MA, USA, 1975.

[9] S. M. Bellovin. Distributed firewalls. *;login: magazine, special issue on security*, November 1999.

[10] M. Bishop. *Computer Security: Art and Science*. Addison Wesley, 2002.

[11] M. Blaze, J. Feigbaum, J. Ioannidis, and A. Keromytis. The keynote trust management system version 2. RFC 2704. Internet Engineering Task Force, September 1999.

[12] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, Oakland, CA, May 1996.

[13] M. Blaze, J. Ioannidis, and A. D. Keromytis. Trust management for IPSec. *ACM Transactions on Information and System Security (TISSEC)*, 5(2):95–118, May 2002.

[14] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[15] D. Box, F. Curbera, M. Hondo, C. Kaler, D. Langworthy, A. Nadalin, N. Nagaratnam, M. Nottingham, C. von Riegen, and J. Shewchuk. *Web Services Policy Framework (WS-Policy). Version 1.1*, September 2004.

[16] J. Burns, A. Cheng, P. Gurung, S. Rajagopalan, P. Rao, D. Rosenbluth, A. Surendran, and D. M. Jr. Automatic management of network security policy. In *DARPA Information Survivability Conference and Exposition (DISCEX II'01)*, volume 2, 2001.

[17] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, 1996.

[18] S. K. Card, J. D. Mackinlay, and B. Shneiderman, editors. *Readings in Information Visualization: Using Vision to Think*. Series in Interactive Technologies. Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[19] D. W. Chapman and A. Fox. *Cisco Secure PIX Firewalls*. Cisco Press, 2001.

[20] Check Point Software Technologies Ltd., Redwood City, Calif. *Check Point Software Revolutionizes Internet Security Management*, January 2000. Checkpoint press release, see: `http://www.checkpoint.com/press/2000/vpe012400.html`.

[21] W. R. Cheswick, S. M. Bellovin, and A. D. Rubin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, 2nd edition, 2003.

[22] Common Criteria Project. *Common Criteria for Information Technology Security Evaluation (CC 2.2), Part 2: Security functional requirements*, January 2004.

[23] T. H. Cormen, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[24] M. J. Covington, M. J. Moyer, and M. Ahamad. Generalized role-based access control for securing future applications. In *23rd National Information Systems Security Conference Proceedings*, 2000.

[25] F. Cuppens, N. Cuppens-Boulahia, T. Sans, and A. Miége. A formal approach to specify and deploy a network security policy. In *Formal Aspects in Security and Trust (FAST 2004)*, 2004.

[26] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. Ponder: A language for specifying security and management policies for distributed systems—the language specification.

Technical Report DoC 2000/1, Imperial College of Science, Technology and Medicine, London, UK, 2000.

[27] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In E. L. M. Sloman, J. Lobo, editor, *Proc. IEEE Workshop Policy 2001: Policies for Distributed Systems and Networks*, number 1995 in Lecture Notes in Computer Science, Heidelberg, 2001. Springer Verlag.

[28] N. Damianou, N. Dulay, E. Lupu, M. Sloman, and T. Tonouchi. Tools for domain-based policy management of distributed systems. In *IEEE/IFIP Network Operations and Management Symposium (NOMS2002)*, pages 213–218, Florence, Italy, 2002.

[29] Distributed Management Task Force (DMTF). *Common Information Model (CIM) Version 2.9*, January 2005. `http://www.dmtf.org/standards/cim/cim_schema_v29`.

[30] P. Dourish and D. Redmiles. An approach to usable security based on event monitoring and visualization. In *NSPW '02: Proceedings of the 2002 workshop on New security paradigms*, pages 75–81. ACM Press, 2002.

[31] Enterprise Management Associates. *Solsoft NP: Putting security policies into practice. White Paper*, 2000. `http://www.solsoft.com/library/ema_profiler.pdf`.

[32] D. Ferraiolo and R. Kuhn. Role-based access control. In *Proceeedings of 15th NIST-NCSC National Security Computer Conference*, Baltimore, MD, 1992.

[33] D. F. Ferraiolo, J. F. Barkley, and D. R. Kuhn. A role-based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information and System Security*, 2(1):34–64, February 1999.

[34] G. W. Furnas. Generalized fisheye views. In *Proceedings of ACM CHI'86 Conference on Human Factors in Computing Systems*, Visualizing Complex Information Spaces, pages 16–23, 1986.

[35] Firewall builder. `http://www.fwbuilder.org`, 2006. Accessed in March, 2006.

[36] S. Garfinkel and G. Spafford. *Practical Unix & Internet Security.* O'Reilly & Associates, Sebastopol, CA, 2nd edition, 1996.

[37] G. Geist. Model-based management of security services: Integrated enforcement of policies in company networks. Master's thesis, University of Dortmund, Germany, 2003. in German.

[38] J. D. Guttman. Filtering postures: local enforcement for global policies. In *SP '97: Proceedings of the 1997 IEEE Symposium on Security and Privacy*, page 120, Washington, DC, USA, 1997. IEEE Computer Society.

[39] D. Haixin, W. Jianping, and L. Xing. Policy-based access control framework for large networks. In *ICON '00: Proceedings of the 8th IEEE International Conference on Networks*, pages 267–272, Washington, DC, USA, 2000. IEEE Computer Society.

[40] C. Hansen. Modellbasiertes Management: Korrelation von Firewall-Logs mit abstrakten Policies. Master's thesis, University of Dortmund, Germany, 2002.

[41] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[42] S. Hinrichs. Policy-based management: Bridging the gap. In *Proceedings of the 15th Annual Computer Security Applications Conference*, Phoenix, AZ, USA, 1999.

[43] HLFL: High-level Firewall Language. `http://www.hlfl.org/`, 2006. Accessed in March, 2006.

[44] ICSA Labs. Certified firewall products. `http://www.icsalabs.com/html/communities/firewalls/certification/rxvendors/index.shtml`, 2006. Accessed in March, 2006.

[45] S. Illner, H. Krumm, I. Lück, A. Pohl, A. Bobek, H. Bohn, and F. Golatowski. Model-based management of embedded service systems - an applied approach. In *Proc. 20th Int. IEEE Conf. on Advanced Information Networking and Applications (AINA2006)*, volume 2, pages 519–523, Vienna, April 2006. IEEE Computer Society Press.

[46] S. Illner, H. Krumm, A. Pohl, I. Lück, D. Manka, and T. Sparenberg. Policy controlled automated management of distributed and embedded service systems. In T. Fahringer and M. H. Hamza, editors, *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks*, pages 710–715, Innsbruck, Austria, February 2005. IASTED, ACTA Press.

[47] S. Illner, A. Pohl, and H. Krumm. Security service adaptation for embedded service systems in changing environments. In *Proceedings of the 2nd IEEE International Conference on Industrial Informatics (INDIN'04)*, pages 457–462, Berlin, Germany, 2004.

[48] Institute of Electrical and Electronics Engineers, New York. *IEEE Standard Glossary of Sof tware Engineering Terminology*, 1990.

[49] International Organization for Standardization. *ISO/IEC 17799:2000, Information technology—Code of practice for information security management*, 2000.

[50] S. Ioannidis, S. M. Bellovin, J. Ioannidis, A. D. Keromytis, and J. M. Smith. Design and implementation of virtual private services. In *WETICE '03: Proceedings of the Twelfth International Workshop on Enabling Technologies*, page 269, Washington, DC, USA, 2003. IEEE Computer Society.

[51] S. Ioannidis, A. D. Keromytis, S. M. Bellovin, and J. M. Smith. Implementing a distributed firewall. In *ACM Conference on Computer and Communications Security*, Athens, Greece, 2000.

[52] isakmpd - ISAKMP/Oakley a.k.a. IKE key management daemon. OpenBSD Manual Pages. Available at: `http://www.openbsd.org/`, 2006. Accessed in March, 2006.

[53] H. Isenberg. Skalierbarer werkzeuggestützter Entwurf der Sicherheitsdienste großer vernetzter IT-Systeme. Master's thesis, University of Dortmund, March 2005. English title: Scalable Tool-assisted Design of Security Services in Large-scale Computer Networks.

[54] ITU-T Telecommunications Standardization Sector of International Telecommunication Union. *X.810 – Security Frameworks for Open Systems: Overview*, November 1995.

[55] R. Jeruschkat. Modellbasierte Policy-Verfeinerung für Sicherheitsdienste großer vernetzter IT-Systeme. Master's thesis, University of Dortmund, March 2006. English title: Model-based Policy Refinement for Security Services in Large-scale Computer Networks.

[56] B. Kempter and V. A. Danciu. Generic policy conflict handling using a priori models. In J. Schönwälder and J. Serrat, editors, *Ambient Networks: 16th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2005*, volume 3775 of *Lecture Notes in Computer Science*, pages 84–96, Berlin Heidelberg, Germany, October 2005. Springer-Verlag.

[57] S. Kent and R. Atkinson. Security architecture for the internet protocol. RFC 2401. Internet Engineering Task Force, 1998.

[58] A. Keromytis. *STRONGMAN: A Scalable Solution to Trust Management in Networks*. PhD thesis, University of Pennsylvania, Pennsylvania, November 2001.

[59] J. Kohl and C. Neuman. The kerberos network authentication service (V5). RFC 1510, Internet Engineering Task Force, Sept. 1993.

[60] O. Köth and M. Minas. Structure, abstraction, and direct manipulation in diagram editors. In M. Hegarty, B. Meyer, and N. H. Narayanan, editors, *Diagrammatic Representation and Inference, Second International Conference (Diagrams 2002)*, volume 2317 of *Lecture Notes in Computer Science*, Callaway Gardens, GA, USA, 2002. Springer.

[61] I. Lück, C. Schäfer, and H. Krumm. Model-based tool-assistance for packet-filter design. In E. L. M. Sloman, J. Lobo, editor, *Proc. IEEE Workshop Policy 2001: Policies for Distributed Systems and Networks*, number 1995 in Lecture Notes in Computer Science, pages 120–136, Heidelberg, 2001. Springer Verlag.

[62] I. Lück, M. Schönbach, A. Mester, and H. Krumm. Derivation of backup service management applications from service and system models. In B. S. R. Stadler, editor, *Active*

Technologies for Network and Service Management, Proc. DSOM'99, number 1700 in Lecture Notes in Computer Science, pages 243–255, Heidelberg, 1999. Springer Verlag.

[63] I. Lück, S. Vögel, and H. Krumm. Model-based configuration of VPNs. In R. Stadtler and M. Ulema, editors, *Proc. 8th IEEE/IFIP Network Operations and Management Symposium NOMS 2002*, pages 589–602, Florence, Italy, 2002. IEEE.

[64] T. K. Lee, S. Yusuf, W. Luk, M. Sloman, E. Lupu, and N. Dulay. Compiling policy descriptions into reconfigurable firewall processors. In *FCCM '03: Proceedings of the 11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, page 39, Washington, DC, USA, 2003. IEEE Computer Society.

[65] J. Leiwo and Y. Zheng. A framework for the management of information security. In *Information Security – Proceedings of the First International Workshop*, number 1396 in Lecture Notes in Computer Science, Heidelberg, DE, 1997. Springer–Verlag.

[66] J. Lobo, R. Bhatia, and S. Naqvi. A policy description language. In *Proc. 16th Nat. Conf. on Artificial Intelligence (AAAI-99)*, pages 291–298. MIT Press, 1999.

[67] *LONGMAN Dictionary of Contemporary English.* Pearson ESL, 4th edition, 2003.

[68] I. Lück. *Model-based Security Service Configuration.* PhD thesis, University of Dortmund, Germany, To Appear in July 2006.

[69] L. Lymberopoulos, E. Lupu, and M. Sloman. Ponder policy implementation and validation in a CIM and differentiated services framework. In *IFIP/IEEE Network Operations and Management Symposium (NOMS 2004)*, Seoul, Korea, April 2004.

[70] Massachusetts Institute of Technology. Kerberos: The network authentication protocol (current release: krb5-1.4.3). `http://web.mit.edu/kerberos/www/`, 2006. Accessed in March, 2006.

[71] D. McBride. Successful deployment of it service management in the distributed enterprise. White Paper, Hewlet-Packard Company, 1998.

[72] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *Software Engineering*, 26(1):70–93, 2000.

[73] J. D. Moffett and M. S. Sloman. Policy hierarchies for distributed system management. *IEEE JSAC Special Issue on Network Management*, 11(9), 11 1993.

[74] M. Mont, A. Baldwin, and C. Goh. POWER prototype: Towards integrated policy-based management. In J. Hong and R. Weihmayer, editors, *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS2000)*, pages 789–802, Hawaii, USA, 2000.

[75] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen. *Policy Core Information Model (PCIM), Version 1: Specification.* Internet Engineering Task Force, February 2001. RFC 3060.

[76] B. Musial and T. Jacobs. Application of focus + context to UML. In T. Pattison and B. Thomas, editors, *Australian Symposium on Information Visualisation, (invis.au'03)*, volume 24 of *Conferences in Research and Practice in Information Technology*, pages 75–80, Adelaide, Australia, 2003. ACS.

[77] Linux Netfilter homepage. `http://www.netfilter.org`, 2006. Accessed in March, 2006.

[78] D. Oppenheimer, A. Ganapathi, and D. Patterson. Why do internet services fail, and what can be done about it. In *4th USENIX Symposium on Internet Technologies and Systems (USITS'03)*, 2003.

[79] J. Park and R. Sandhu. The $ucon_{ABC}$ usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, 2004.

[80] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, 1972.

[81] C. N. Payne. *Handbook for the Computer Security Certification of Trusted Systems*, chapter The Security Policy Model. NRL Technical Memorandum 5540:080A. Naval Research Laboratory, Washington, D. C., USA, Jan 1995.

[82] PF: The OpenBSD packet filter. `http://www.openbsd.org/faq/pf/`, 2006. Accessed in March, 2006.

[83] J. Porto de Albuquerque. Scalable model-based policy refinement and validation for network security systems. Technical Report IC-06-005, Institute of Computing, University of Campinas. Available at: `http://www.ic.unicamp.br/~jporto/report_formalisation.pdf`, April 2006.

[84] J. Porto de Albuquerque and P. L. de Geus. Agentes 'proxy': Conceitos e técnicas de implementação. In *III Simpósio de Segurança em Informática - SSI'2001*, São José dos Campos, SP, October 2001. [*Proxy Agents: Concepts and Implementation Techniques*].

[85] J. Porto de Albuquerque and P. L. de Geus. A framework for network security system design. *WSEAS Transactions on Systems*, 2:139–144, January 2003.

[86] J. Porto de Albuquerque, H. Isenberg, H. Krumm, and P. L. de Geus. Gerenciamento baseado em modelos da configuração de sistemas de segurança em redes de larga escala. In *V Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 174–187, Florianópolis, Brazil, September 2005.

[87] J. Porto de Albuquerque, H. Isenberg, H. Krumm, and P. L. de Geus. Improving the configuration management of large network security systems. In J. Schönwälder and J. Serrat, editors, *Ambient Networks: 16th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2005, Barcelona, Spain, October 24-26, 2005, Proceedings*, volume 3775 of *Lecture Notes in Computer Science*, pages 36–47, Berlin Heidelberg, Germany, October 2005. Springer-Verlag.

[88] J. Porto de Albuquerque, H. Krumm, and P. L. de Geus. Modelagem de sistemas de segurança em ambientes de redes de larga escala. In *Anais do 23o. Simpósio Brasileiro de Redes de Computadores (SBRC)*, Fortaleza, Brazil, May 2005.

[89] J. Porto de Albuquerque, H. Krumm, and P. L. de Geus. Modellierung von netzsicherheitsystemen umfangreicher vernetzter it-infrastrukturen. In A. B. Cremers, R. Manthey, P. Martini, and V. Steinhage, editors, *INFORMATIK 2005 - Informatik LIVE! Band 2, Beiträge der 35. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Bonn, 19. bis 22. September 2005*, volume 68 of *Lecture Notes in Informatics*, pages 633–637, Bonn, Germany, September 2005. GI.

[90] J. Porto de Albuquerque, H. Krumm, and P. L. de Geus. On scalability and modularisation in the modelling of security systems. In S. D. C. di Vimercati, P. F. Syverson, and D. Gollmann, editors, *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*, volume 3679 of *Lecture Notes in Computer Science*, pages 287–304, Berlin Heidelberg, Germany, September 2005. Springer-Verlag.

[91] J. Porto de Albuquerque, H. Krumm, and P. L. de Geus. Policy modeling and refinement for network security systems. In *6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005), 6-8 June 2005, Stockholm, Sweden*, pages 24–33, Washington, DC, USA, June 2005. IEEE Computer Society.

[92] J. Porto de Albuquerque, H. Krumm, P. L. de Geus, and R. Jeruschkat. Scalable model-based configuration management of security services in complex enterprise networks. *Submitted to Computer Networks, Elsevier*, July 2006.

[93] R. S. Pressman. *Software Engineering: A Practitioner's Approach.* McGraw-Hill, 5th edition, 2001.

[94] G. Rothmaier. Model-based security management: Abstract requirements, trusts areas, and configuration of security services. Master's thesis, University of Dortmund, Germany, 2001. in German.

[95] J. Rushby. Security requirements specifications: How and what? In *Symposium on Requirements Engineering for Information Security (SREIS)*, Indianapolis, IN, March 2001.

[96] R. S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11):9–19, 1993.

[97] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[98] R. S. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Communications*, 32(9), September 1994.

[99] M. Sarkar and M. H. Brown. Graphical fisheye views of graphs. In *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems*, Visualizing Objects, Graphs, and Video, pages 83–91, 1992.

[100] C. Schäfer. Modell- und Policy-basiertes Sicherheitsmanagement: Administration von Firewall-Konfigurationen. Master's thesis, University of Dortmund, Germany, 2000.

[101] C. L. Schuba. *On the Modeling, Design and Implementation of Firewall Technology.* PhD thesis, Department of Computer Sciences, Purdue University, December 1997.

[102] C. L. Schuba and E. H. Spafford. A reference model for firewall technology. In *ACSAC '97: Proceedings of the 13th Annual Computer Security Applications Conference*, page 133, Washington, DC, USA, 1997. IEEE Computer Society.

[103] C. Scott, P. Wolfe, and M. Erwin. *Virtual Private Networks.* O'Reilly & Associates, 2nd edition, 1998.

[104] M. Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2(4):333–360, 1994.

[105] M. Sloman and E. C. Lupu. Security and management policy specification. *IEEE Network, Special Issue on Policy-Based Networking*, 16(2):10–19, March/April 2002.

[106] Y. Snir, Y. Ramberg, J. Strassner, R. Cohen, and B. Moore. *Policy Quality of Service (QoS) Information Model. Request for Comments 3644.* Internet Engineering Task Force, November 2003.

[107] I. Sommerville. *Software Engineering.* Addison-Wesley, 6th edition, 2000.

[108] T. Sparenberg. Modellbasiertes Sicherheitsmanagement: Systematische Administration aktueller Firewall Produkte. Master's thesis, University of Dortmund, Germany, 2001.

[109] W. Stallings. *Cryptography and Network Security: Principles and Practice.* Prentice Hall, Inc., Upper Saddle River, NJ, USA, 2nd edition, 1998.

[110] D. F. Sterne. On the buzzword security policy. In *IEEE Symposium on Security and Privacy*, Oakland, California, USA, 1991.

[111] S. Vögel. Modellbasiertes Sicherheitsmanagement: Automatisierte Konfiguration von virtuellen privaten Netzen. Master's thesis, University of Dortmund, Germany, 2001.

[112] D. D. Welch-Abernathy. *Essential Checkpoint Firewall-1: An Installation, Configuration, and Troubleshooting Guide*. Addison-Wesley, 2002.

[113] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, and S. Waldbusser. *Terminology for Policy-Based Management*. Internet Engineering Task Force, November 2001. RFC 3198.

[114] R. Wies. Policies in network and systems management — formal definition and architecture. *Journal of Network and Systems Management*, 2(1):63–83, 1994.

[115] R. Wies. Using a classification of management policies for policy specification and policy transformation. In A. S. Sethi, Y. Raynaud, and F. Fure-Vincent, editors, *Integrated Network Management IV*, volume 4, pages 44–56, Santa Barbara, CA, 1995. Chapman & Hall.

[116] A. Wool. Architecting the lumeta firewall analyzer. In *10th USENIX Security Symposium*, pages 85–97, Washington D.C., 2001. USENIX.

[117] A. Wool. The use and usability of direction-based filtering in firewalls. *Computers & Security*, 23:459–468, 2004.

[118] W. Yurcik, J. Barlow, K. Lakkaraju, and M. Haberman. Two visual computer network security monitoring tools incorporating operator interface requirements. In *ACM CHI Workshop on Human-Computer Interaction and Security Systems (HCISEC)*, Fort Lauderdale, Florida, USA, April 2003.

[119] E. D. Zwicky, S. Cooper, and D. B. Chapman. *Building Internet Firewalls*. O'Reilly and Associates, Sebastopol, CA, 2nd edition, 2000.

Appendix A

Refinement Algorithms

Algorithm 1 defines the refinement process from SR to DAS; i.e. the generation of a set of *ATPs* for each service permission in SP. Hence, for each *Actor* that refines the *User* and *SubjectType* associated with the service permission *sp* being refined (line 2), the algorithm executes the auxiliary function $findpaths()$ (line 6) providing as parameter the set of *Targets* related to *sp*. This set comprises *Targets* in two different cases: a) if the *Service sv* (associated to *sp*) has direct access to the *Ressource r*, the related *Target* will refine these two objects through the relation *RT* (line 4); b) if there is a service dependency, then the related *Target* will refine *r* and another *Service sd*, on which *sv* depends to access *r* through the relation *SvDep* (line 5). Notice that, for the sake of notation conciseness, the service dependency is henceforth restricted to one level.

The function $findpaths(P, a, T)$, in turn, is a straightforward adaptation of the classical *modified Djikstra algorithm* for discovery of single-source shortest paths [23, Ch. 25]. It returns the set P of all paths between the initial node a and any node in the destination set T and is thus not shown in details here. In the sequence, the algorithm verifies each path found by invoking the function *testpath*(); if the path is considered valid, its security requirement is set as the one of the related service permission (line 9) and it is then added to the ATP set (line 10).

Algorithm 2 presents the *testpath*(p, sp) function, which takes as input the path p to be tested and the corresponding service permission *sp*. The path shall be tested into two different aspects: i) satisfaction of service dependencies; and ii) compatibility between *Actor* and *Target*.

In the *for*-loop of lines 2–6, the service dependencies related to the permission are checked. The loop then tests if, for each service dependency, a corresponding mediator is present in the path (line 3). In the negative case, the path is considered invalid.

The second test applied to the path verifies the compatibility of the related *Actor* (say a) and *Target* (say t); i.e. it tests whether the *protocol* and *connection orientation* defined for a in the AS expanded view matches the corresponding definitions for t. In the case there is a mediator along the path, the compatibility is tested first from the *Actor* to the *Mediator*, and then from this to the *Target*; for the *Mediator* can here act as a gateway. For instance, in Fig. 4.9 of Sect. 4.1.4, the process “Netscape” (of host “WS2”) is assigned to the protocol “http”,

Algorithm 1 Refinement $SR \rightarrow DAS$

```
for all $sp(u, st, sv, r) \in SP$ do
  for all $a : (a, u, st) \in RA$ do
    $P \leftarrow \emptyset$ {set of path candidates}
    $T \leftarrow \{t : (t, sv, r) \in RT\} \cup$
        $\{t : (t, sd, r) \in RT \wedge (sv, sd, r) \in SvDep\}$ {set of targets related to $sp$}
    $findpaths(P, a, T)$
    for all $p \in P$ do
      if $testpath(p, sp)$ then
        $sr[p] \leftarrow sr[sp]$
        $ATP \leftarrow ATP \cup p$
      end if
    end for
  end for
end for
```

Algorithm 2 Function $testpath(p, sp : (u, st, sv, r)) : boolean$

```
$sa[p] \leftarrow sa[sv]$; $testpath \leftarrow true$
for all $sd : (sv, sd, r) \in SvDep$ do {tests service dependencies}
  if $\nexists\ m :\ (m, sd) \in RM$ and $m \in p$ then
    $testpath \leftarrow false$
  end if
end for
if not $path\_compatible(p)$ then
  $testpath \leftarrow false$
end if
```

and the diamond near the latter indicates that the former will initiate the communication (i.e. has outgoing orientation). The corresponding *Actor* “internal web clients” is hence compatible with the *Mediator* “Webproxy”, since the process “Squid-Proxy” assigned to the latter is as well connected to a “http” protocol, but has an incoming orientation. The algorithm for this verification is trivial and is thus omitted here.

The interested reader may find a more detailed explanation of the refinement algorithms and of the configuration generation back-ends in [55].

Appendix B

Proof of the Generalisation Theorems

B.1 Proof of GT1

The proof of this theorem is based on an induction on the number of ASs for which the assertion is valid. For the basis, let us consider a path dp_1 that is a subpath of some $aep_1 \in AEP$, such that dp_1 is completely enclosed inside a single AS; i.e. $dp_1 \in LEP$ (Definition 6.1.0.9). Thus all processes along dp_1 pertain to the same AS, and the basis hypothesis can be stated as

$$dp_1 \subseteq aep_1 \in AEP: \; pc_1, \ldots, pc_m \text{ is the process sequence in } dp_1 \\ \wedge \; sub[pc_1] = sub[pc_2] = \cdots = sub[pc_m] \quad \text{(B.1)}$$

Hence, applying this hypothesis to Cond. 6.4.1.2 follows that there is a corresponding abstract path ap_1 in DAS, such that

$$ap_1 \subseteq atp_1 \in ATP, \; ap_1 = \langle v_1, \ldots, v_m \rangle : expand[dp_1, ap_1]$$

The definition of the predicate *expand* (Definition 6.4.1.1) implies that

$$(e_i, v_i) \in RPc \;\vee\; (e_i = v_i) \text{ for } 1 \leq i \leq n \quad \text{(B.2)}$$

Where $e_1, \ldots, e_n$ is the sequence of processes and connectors of dp_1. But the hypothesis in Eq. (B.1) states that all elements of dp_1 pertain to the same AS, so that the structure of AEP elements (described in Definition 6.1.0.9) implies that there are no connectors in dp_1 (i.e. $n = m$). Hence, the right-hand term of the disjunction in Eq. (B.2) is false for all elements—since only connectors can be contained in both an element of ATP and an element of AEP. Equation (B.2) then becomes

$$(pc_i, v_i) \in RPc \text{ for } 1 \leq i \leq m \quad \text{(B.3)}$$

Considering now a pair of subsequent DAS elements in ap_1, said v_j and v_{j+1}, Cond. 6.4.2.5 asserts that:

$$\forall\ pc_a, pc_b \in Pc : (pc_a, v_j) \in RPc\ \wedge\ (pc_b, v_{j+1}) \in RPc \Rightarrow \\ \exists\ p = \langle pc_a, \ldots, pc_b \rangle : lconn[p] \quad \text{(B.4)}$$

Clearly, the universal quantifier in Eq. (B.4) implies that also for pc_j and pc_{j+1} that are respectively connected to v_j and v_{j+1} by Eq. (B.3):

$$\exists\ p = \langle pc_j, \ldots, pc_{j+1} \rangle : lconn[p] \qquad \text{(B.5)}$$

Since pc_j and pc_{j+1} were chosen without loss of generality, Eq. (B.5) is valid for each pair of subsequent processes in dp_1. Definition 6.4.4.1 asserts that a local-connected path is also connected, so that we prove, as intended, that for the basis case:

$$\exists\ p = \langle pc_i, \ldots, pc_{i+1} \rangle : conn[p] \text{ for } 1 \leq i < m \quad \blacksquare$$

For the general case, let us assume:

$$dp_2 \subseteq aep \in AEP : \ pc_1, \ldots, pc_o \text{ is the process sequence in } dp_2, \ \ dp_2 \text{ spans } k \text{ ASs} \qquad \text{(B.6)}$$

The induction hypothesis then reads:

$$\textit{Theorem 6.4.4.1 is valid for all } dp \subseteq atp \in ATP : dp \textit{ spans up to } k-1 \textit{ ASs} \qquad \text{(B.7)}$$

Proceeding in analogy with the basis case, from Eq. (B.6), Cond. 6.4.1.2, and Definition 6.4.1.1 follows that there is a corresponding ap_2 in DAS, such that:

$$ap_2 \subseteq atp_2 \in ATP, \ ap_2 = \langle v_1, \ldots, v_p \rangle : \ (e_i, v_i) \in RPc\ \vee\ (e_i = v_i) \text{ for } 1 \leq i \leq p \qquad \text{(B.8)}$$

Where $e_1, \ldots, e_p$ is the sequence of processes and connectors of dp_2. Now let us consider that $e_r = v_r$ is the first connector in this sequence (thus $e_1 = pc_1, \ldots, e_{r-1} = pc_{r-1}$, since they are all processes), and dsp_1 is the subpath that contains the first elements of dp_2 until process pc_{r-1}. According to the Definition 6.1.0.9, dsp_1 is thus completely enclosed in one subsystem; i.e. $sub[pc_1] = sub[pc_2] = \cdots = sub[pc_{k-1}]$. Furthermore, the same Definition 6.1.0.9 implies $e_{r+1} = v_{r+1}$ is also a connector, and pertains to a different subsystem. Hence, we can apply the basis case to dsp_1, achieving that:

$$\exists\ p = \langle pc_j, \ldots, pc_{j+1} \rangle : conn[p] \text{ for } 1 \leq j < r-1 \qquad \text{(B.9)}$$

Now, consider dsp_2 the subpath of dp_2 that starts at process e_{r-1}, passes through the connectors e_r and e_{r+1} and ends at the subsequent process e_{r+2}. From Eq. (B.8) there is a corresponding DAS subpath in ap_2:

$$asp_1 = \langle v_{r-1}, v_r, v_{r+1}, v_{r+2} \rangle : \ (v_{r-1}, v_r) \in E\ \wedge\ (v_r, v_{r+1}) \in E\ \wedge\ (v_{r+1}, v_{r+2}) \in E \qquad \text{(B.10)}$$

And:

$$(e_{r-1}, v_{r-1}) \in RPc \ \wedge \ e_r = v_r \in C \ \wedge \ e_{r+1} = v_{r+1} \in C \ \wedge \ (e_{r+1}, v_{r+1}) \in RPc \tag{B.11}$$

Thus, applying Eq. (B.10) to Cond. 6.4.3.1 comes:

$$\forall \ pc_x, pc_y \in Pc : (pc_x, v_x) \in RPc \ \wedge \ (pc_y, v_y) \in RPc \Rightarrow \\ \exists \ p = \langle pc_x, \ldots, pc_y \rangle : cconn[p] \tag{B.12}$$

Clearly, the universal quantifier in Eqs. (B.12) and (B.11) imply that:

$$\exists \ p = \langle e_{r-1}, \ldots, e_{r+2} \rangle : cconn[p] \tag{B.13}$$

Since $e_{r-1} = pc_{r-1}$ and $e_{r+2} = pc_r$ (e_r and e_{r+1} are connectors), and relying upon Definition 6.4.4.1 we can then extend Eq. (B.9) to:

$$\exists \ p = \langle pc_j, \ldots, pc_{j+1} \rangle : conn[p] \text{ for } 1 \leq j < r \tag{B.14}$$

Now, let dsp_3 be the subpath of dp_2 that goes from $e_{r+2} = pc_r$ to the end (e_p). Since Eq. (B.6) states that dp_2 spans k subsystems, and dsp_3 is the result of removing the subpath contained in the first subsystem of dp_2 (i.e. the subpath $\langle e_1, \ldots, e_r \rangle$), dsp_3 thus spans $k-1$ subsystems. Hence, the induction hypothesis Eq. (B.7) can be applied to dsp_3 achieving that:

$$\exists \ p = \langle pc_l, \ldots, pc_{l+1} \rangle : conn[p] \text{ for } r \leq l < n \tag{B.15}$$

Therefore, from Eqs. (B.14) and (B.15) follows:

$$\exists \ p = \langle pc_j, \ldots, pc_{j+1} \rangle : conn[p] \text{ for } 1 \leq j < n \quad \square$$

B.2 Proof of GT2

Relying upon Definition 6.4.4.1, the theorems' assertion can be expanded into two subgoals to be proved:

$$\forall \ pc_x, pc_y \in Pc : \exists \ p_1 = \langle pc_x, \ldots, pc_y \rangle \wedge \ lconn(p_1) \Rightarrow \ \exists \ aep_1 \in AEP : \\ pc_1, \ldots, pc_m \text{ is the sequence of processes in } aep_1 \\ \wedge \ pc_x = pc_i \ \wedge \ pc_y = pc_{i+1} \text{ for } 1 \leq i < m \tag{B.16}$$

And:

$$\forall \ pc_z, pc_w \in Pc : \exists \ p_2 = \langle pc_z, \ldots, pc_w \rangle \wedge \ cconn(p_2) \Rightarrow \ \exists \ aep_2 \in AEP : \\ pc_1, \ldots, pc_o \text{ is the sequence of processes in } aep_2 \\ \wedge \ pc_z = pc_j \ \wedge \ pc_w = pc_{j+1} \text{ for } 1 \leq j < o \tag{B.17}$$

Beginning with Eq. (B.16), from Cond. 6.4.2.6 follows:

$$\exists\ v_x, v_y \in V : (pc_x, v_x) \in RPc\ \wedge\ (pc_y, v_y) \in RPc\ \wedge\ (v_x, v_y) \in E \tag{B.18}$$

Cond. 6.3.2.3 and Eq. (B.18) thus imply:

$$\exists\ atp_1 \in ATP : \langle v_x, v_y \rangle \subseteq atp_1 \tag{B.19}$$

Therefore, if we consider $atp_1 = \langle v_1, \ldots, v_n \rangle$ then $v_x = v_i$ and $v_y = v_{i+1}$ for some $1 \leq i < n$. Now, from Cond. 6.4.1.1 follows:

$$\begin{aligned}\forall\ w_1 \in Pc,\ w_m \in Pc :\ (w_1, v_1) \in RPc \wedge\ (w_m, v_n) \in RPc \Rightarrow \\ \exists\ aep \in AEP,\ aep = \langle w_1, \ldots, w_n \rangle : expand[aep, atp_1]\end{aligned} \tag{B.20}$$

We must analyse now four different subcases to cover the different positions that the elements v_i and v_{i+1} may occupy in atp_1. In the first subcase $i = 1$ and $i+1 = n$, thus from the definition of ATP (Definition 6.1.0.3) follows that $v_i = v_1 \in A$ (it is an *Actor*) and $v_{i+1} = v_n \in T$ (it is a *Target*). Thus, the universal quantifier in Eq. (B.20) implies, that also for pc_x and pc_y correspondingly related to $v_x = v_i$ and $v_y = v_{i+1}$ by Eq. (B.18):

$$\exists\ aep_1 \in AEP,\ aep_1 = \langle pc_x, \ldots, pc_y \rangle \quad \blacksquare$$

In the second case, $i > 1$ and $i + 1 < n$, and Definition 6.1.0.3 implies $v_i \in M$ and $v_{i+1} \in M$ (both are *Mediators*). Hence, Condition 6.4.2.2 implies that there is only one process related to each mediator, so that from Eq. (B.18), Eq. (B.20) and Definition 6.4.1.1 (definition of function *expand*) follows that:

$$\exists\ aep_1 \in AEP,\ aep = \langle w_1, \ldots, pc_x, \ldots, pc_y, \ldots, w_n \rangle \tag{B.21}$$

The third and fourth cases (namely $i = 1\ \wedge\ i+1 < n$; and $i > 1\ \wedge\ i+1 = n$) can be analogously demonstrated by a combination of the two previous cases. ∎

Now, to prove Eq. (B.17) we must apply Cond. 6.4.3.2, achieving:

$$\begin{aligned}\exists\ v_z, v_w \in V, c_1, c_2 \in C :\ (pc_z, v_z) \in RPc\ \wedge\ (pc_w, v_w) \in RPc \\ \wedge\ (v_z, c_1) \in E\ \wedge\ (c_1, c_2) \in E\ \wedge\ (c_2, v_w) \in E\end{aligned} \tag{B.22}$$

From Eq. (B.22) and Cond. 6.3.2.4 comes:

$$\exists\ atp_2 \in ATP : \langle v_z, c_1, c_2, v_w \rangle \subseteq atp_2 \tag{B.23}$$

Therefore, if we consider $atp_2 = \langle v_1, \ldots, v_n \rangle$ then $v_z = v_k$, $c_1 = v_{k+1}$, $c_2 = v_{k+2}$, and $v_w = v_{k+3}$ for some $1 \leq k < n - 3$. From Cond. 6.4.1.1 thus follows:

$$\begin{aligned}\forall\ w_1 \in Pc,\ w_o \in Pc :\ (w_1, v_1) \in RPc \wedge\ (w_o, v_n) \in RPc \Rightarrow \\ \exists\ aep \in AEP,\ aep = \langle w_1, \ldots, w_o \rangle : expand[aep, atp_2]\end{aligned} \tag{B.24}$$

Hence, four analogous subcases to the ones in the first part of this proof must be considered, in order to cover the different positions that v_z and v_w may occupy in atp_2 (namely i) $k = 1 \ \wedge k + 3 = n$; ii) $k > 1 \ \wedge k + 3 < n$; iii) $k = 1 \ \wedge k + 3 < n$; and iv) $k > 1 \ \wedge k + 3 = n$). Proceeding in the same manner as before, we can prove for all four subcases that:

$$\exists \ aep_2 \in AEP, \ aep_2 = \langle w_1, \ldots, pc_z, \ldots, pc_w, \ldots, w_o \rangle \quad \square$$

Appendix C

Models of the Application Case

We present here some of the models obtained for the application case analysed in Sect. 7.3. The growth in the complexity of the ES level is made clear from the comparison of the models of the AS "internal network" in the realistic application case (Fig. C.1) with that in the test scenario (left hand of Fig. 4.9), and similarly for the AS "dmz" (compare Fig. C.2 with the right hand of Fig. 4.9).

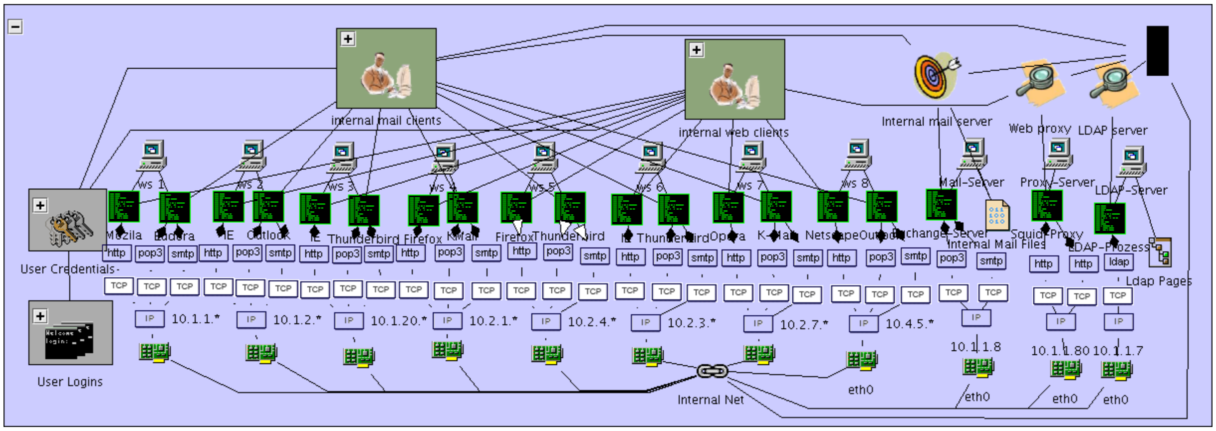


Figure C.1: Abstract and expanded view of subsystem "internal network"

In contrast, the superior levels of the modelling show a slower growth behaviour, and hence more scalability. Figure C.3 presents the three-layered model for the application case (compare with Fig. 4.8).

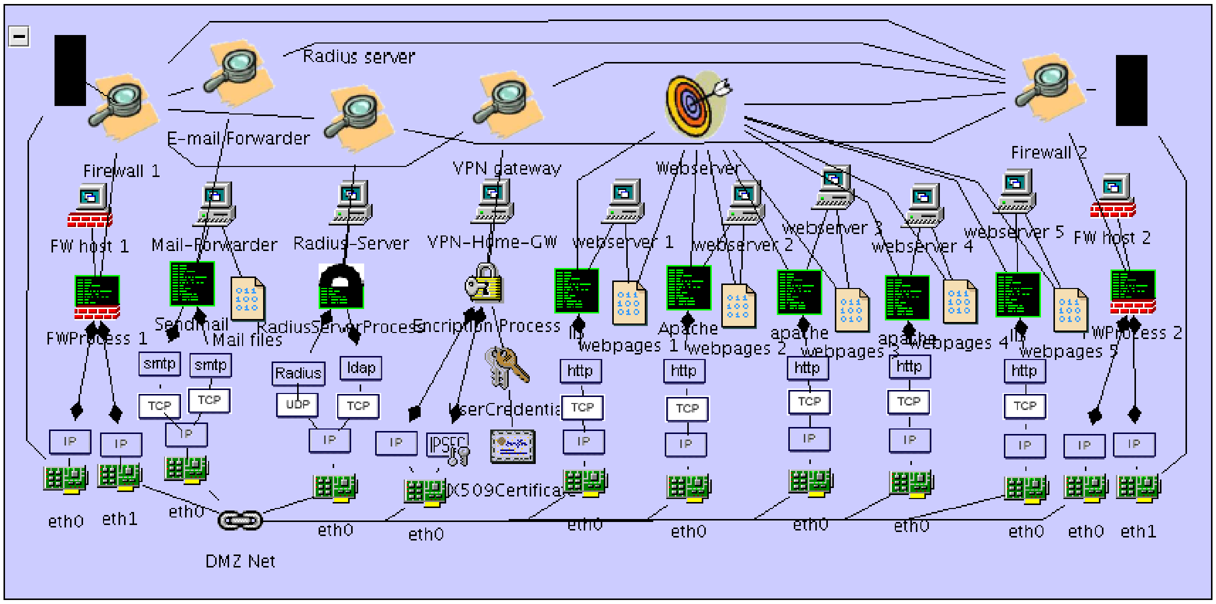


Figure C.2: Abstract and expanded view of subsystem “dmz”

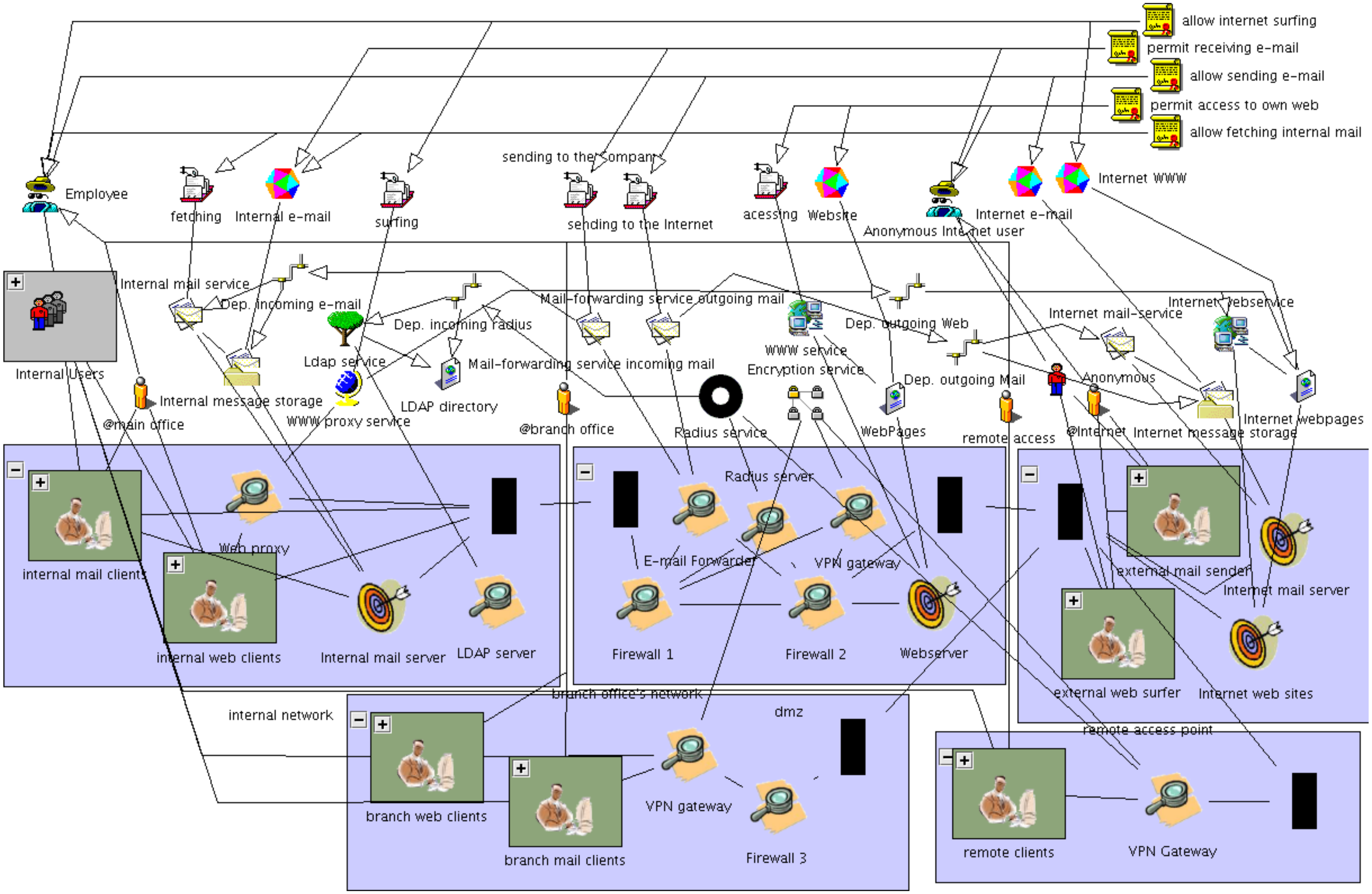


Figure C.3: Three-layered model for the case study