

# Formal validation of automated policy refinement in the management of network security systems

João Porto de Albuquerque · Heiko Krumm · Paulo Lício de Geus

Published online: 20 February 2010  
© Springer-Verlag 2010

**Abstract** Policy hierarchies and automated policy refinement are powerful approaches to simplify administration of security services in complex network environments. A crucial issue for the practical use of these approaches is to ensure the validity of the policy hierarchy, i.e. since the policy sets for the lower levels are automatically derived from the abstract policies (defined by the modeller), we must be sure that the derived policies uphold the high-level ones. This paper builds upon previous work on Model-based Management, particularly on the Diagram of Abstract Subsystems approach, and goes further to propose a formal validation approach for the policy hierarchies yielded by the automated policy refinement process. We establish general validation conditions for a multi-layered policy model, i.e. necessary and sufficient conditions that a policy hierarchy must satisfy so that the lower-level policy sets are valid refinements of the higher-level policies according to the criteria of consistency and completeness. Relying upon the validation conditions and upon axioms about the model representativeness, two theorems are proved to ensure compliance between the resulting system behaviour and the abstract policies that are modelled.

**Keywords** Policy refinement · Model-based management · Formal validation · Security policies · Policy based management · Network security

## 1 Introduction

Current enterprises heavily rely upon network infrastructures that are connected to the internet in order to effectively perform their business. In these environments, a great variety of security technologies and mechanisms are employed to offer protection against network-based attacks. Whilst significant progress has been made on improving network security technology in recent years, research still remains to be done to offer a proper abstraction, integration, and tool support for the management of the configuration of security services. In practice, a security administrator must nowadays deal with a great number of complex and heterogeneous configuration syntaxes, most of which are unintuitive and in some cases even misleading. This error-prone process is a threat to the security of those environments, since a single maladjustment between two mechanisms can leave the whole system vulnerable to attacks.

### 1.1 Previous work

Within this context, *policy-based network management* offers a promising approach, since it describes the behaviour of different mechanisms by means of abstract and uniform policies [23]. Model-based Management (MBM) [16, 17] is a policy-based approach that employs an object-oriented layered model. It aims to provide a smooth transition from an abstract view of the system to be managed and the policies that apply to it down to reaching a detailed system representation at the most inferior layer. The modeller thus defines the

---

J. P. de Albuquerque (✉)  
School of Arts, Sciences and Humanities, University of Sao Paulo,  
Rua Arlindo Bétio, 1000, Ermelino Matarazzo, São Paulo,  
SP 03828-000, Brazil  
e-mail: joao.porto@usp.br

H. Krumm  
Department of Computer Science, Technical University  
of Dortmund, 44221 Dortmund, Germany  
e-mail: Heiko.Krumm@udo.edu

P. L. de Geus  
Institute of Computing, University of Campinas,  
Campinas, SP 13083-852, Brazil  
e-mail: paulo@ic.unicamp.br

system in each abstraction level, but the policies are specified only at the most abstract layer. After the model is complete, an automated policy refinement process takes place that generates policy sets for the lower levels, ultimately achieving the derivation of configuration parameters for all the security mechanisms of the system. In a further development, the scalability of the approach was improved by means of an additional layer, the *Diagram of Abstract Subsystems* (DAS), in order to cope with large-scale, complex network environments [7–9].

## 1.2 Current work

A subject that was not sufficiently explored up until then is the correctness of the automated policy refinement in MBM, which was briefly introduced in Porto de Albuquerque et al. [9]. Indeed, since the policy sets for the lower levels are automatically derived from the abstract policies (defined by the modeller), we must be sure that the derived policies uphold the high-level ones, i.e. we must assure that the refinement algorithms always produce a correct refinement from the policies and system objects given by the modeller. Only in this case the configuration generated from the lower levels can be expected to be in conformance with the abstract policies that were specified. The present work is thus dedicated to examine this issue.

This paper addresses this problem by proposing validation criteria for the automated policy refinement in MBM. We present here a general result that establishes conditions for the application of a policy refinement algorithm to always comply with the general validation criteria for policy hierarchies of *consistency* and *completeness*. Additionally, in order to reason about the effect of the application of the automatically generated configuration to the real environment, a series of axioms are defined to capture the assumptions that are implicit in the modelling. We thus prove that the defined conditions are sufficient and necessary to guarantee the validity of the refinement.

The rest of the paper is organised as follows: The first two sections summarise previous works, where Sect. 2 presents the main elements of the modelling technique that was employed, and Sect. 3 analyses the policy support offered in the MBM approach and explains the policy refinement process. Then, Sect. 4 presents the validation approach proposed in this paper, including an overview of the validation approach as a whole in Sect. 4.1, in order to guide the reader throughout the section. In Sect. 5, the practical results achieved are presented and a simple use example of the approach is given. Section 6 discusses the validation results, making considerations about the validation soundness. Lastly, Sect. 7 discusses related work and Sect. 8 summarises the contributions of this paper and points out to future work.

## 2 Modelling framework

The modelling framework of our approach is structured in three layers, as depicted in Fig. 1. Each of the layers is a refinement of the superior one in the sense of a “policy hierarchy” [18,26], 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.

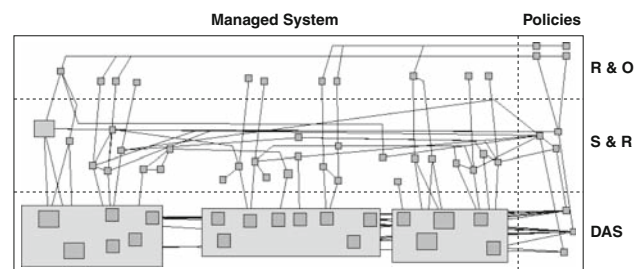
Thus, the horizontal dashed lines of Fig. 1 delimit the abstraction levels of the model: *Roles & Objects* (RO), *Subjects & Resources* (SR), and *Diagram of Abstract Subsystems* (DAS). As for the vertical subdivision, it differentiates between the model of the actual managed system (on the left-hand 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 [16,17] and extended. The RO level is based on concepts from Role-Based Access Control (RBAC) [11,21], and the second level (SR in Fig. 1) offers a system view defined on the basis of the services that will be provided. They are both briefly described in Sect. 2.1.

The third model layer (DAS) is where the main contribution of this work is applied. It aims at offering a modular description of the system’s *overall structure*, thereby improving the scalability of the modelling technique.

### 2.1 RO and SR layers

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 [11]. The main classes in this level thus represent *Roles* in which people who are working in the modelled environment act; *Objects* that should be subjected to access control; and *AccessModes*, i.e. ways to access 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 2 shows the graphical representation for each of these classes as well as for the remaining classes used in



**Fig. 1** Meta-model overview















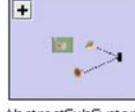









	Elements	Policies
RO	 Role  AccessMode  Object	 AccessPermission
SR	 User  SubjectType  Service  Resource  ServiceDependency	 ServicePermission
DAS	 Actor  Mediator  Target  Connector  AbstractSubSystem	 ATPermission
ES	 Process  Host  SystemResource  IP  Connector  Interface  Link	 ADP

Fig. 2 Classes of the model

our modelling framework, which are described in the next sections.

The classes in the RO level correspond to and convey the same semantics of the equally named entities in the RBAC terminology (see [21]). 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 Ferraiolo et al. [12]). 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. 1) offers a system view defined on the basis of the services that will be provided. Objects of these classes represent the following: (a) people working in the modelled environment (*Users*); (b) 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.

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. These two classes add information about each user that is authorised to act in a certain role and the possible subjects that may take place on the system. 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  $RBAC_0$  [21]. The subjects (also named sessions in the RBAC literature) refer to a mapping of one user to possibly many roles that are activated simultaneously [21]. As for the “type” suffix, it is appended in MBM to indicate that

the objects do not represent all the possible subjects during the execution of the system as in RBAC; rather they stand for the basic types of these subjects that may occur at run-time (see also [15]).

### 2.1.1 Formalisation

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. The possible connections between two or more classes in the meta-model are 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. 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 along the paper.

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

**Definition 2.1** The SR level has the following components:

- $U, St, Sv, R$ , disjoint sets respectively encompassing objects of the *User*, *SubjectType*, *Services*, and *Resources* classes;
- $SvDep \subseteq Sv \times Sv \times R$ , a partially ordered relationship to express that a service depends on another to access a resource.

## 2.2 The DAS level

The main objective of the Diagram of Abstract Subsystems (DAS) is to describe the *overall structure* of the system in a modular fashion, i.e. to cast the system into its building blocks and to indicate their interconnections. As such, this diagram is conceived to provide security administrators and designers with an intelligible view of the system's architecture, by means of which the system configuration can be effectively managed.

A DAS is, formally speaking, a graph comprised of Abstract Subsystems (ASs) as nodes, and edges that represent the possibility of bi-directional communication between two ASs. An AS, in turn, contains an abstract view of a certain system segment, i.e. a simplified representation of a given group of system components that may rely on the following types of elements:

*Actors*: groups of individuals which have an *active* behaviour in a system, i.e. they initiate communication and execute mandatory operations according to *obligation policies* (see Sect. 3.1).

*Mediators*: elements that intermediate communication, in that they receive requests, inspect traffic, filter and/or transform the data flow according to the *authorisation policies*; they can also perform mandatory operations based on *obligation policies*, such as registering information about data flows.

*Targets*: *passive* elements; they contain relevant information, which is accessed by *actors*.

*Connectors*: represent the interfaces of one AS with another; i.e. they allow information to flow from, and to, an AS.

Each element of the types *Actors*, *Mediators* or *Targets* represents a group of system elements that have a relevant behaviour for a global, policy-oriented view of the system. As for the *Connectors*, they are related to the physical interfaces of an AS (for a detailed elaboration on the modelling of abstract subsystems we refer to the study by Porto de Albuquerque et al. [8]). The DAS is formally defined as follows.

**Definition 2.2** The DAS level comprises the following elements:

- $A, M, T, C, Su$ , sets respectively enclosing *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);
- $sub : V \rightarrow Su$ , a function that gives the subsystem to which a certain element of  $V$  is assigned in the model.

**Definition 2.3** The associations between elements of SR and DAS are formalised as follows:

- $RA \subseteq A \times U \times St$ , 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*.

## 2.3 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 the goals are defined at the RO level by *SecurityGoal* objects. 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 Critical,  $24 \times 7$  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 by Westinen et al. [25]).

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 [14]. 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*. 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.

An additional type of object is included in the model in order to express the security properties that an entity is assumed to *assure*: the *SecurityAssumptions*. 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. to represent the security levels one can assume in a communication that involves that service. Analogously, at the DAS level a modeller may assign a *SecurityAssumption* both to subsystems—representing the security class that the elements inside an AS are assumed to share—and individually to *Actors*, *Targets*,

and *Mediators*—when a particular object is assumed to have different security properties than those of the other elements in the subsystem.

**Definition 2.4** The security requirements and assumptions are formally defined as follows:

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

**Definition 2.5** 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 (the security classes in  $SC$  together with the partial order  $\leq$  form a lattice):

- $sc_1 \leq sc_2$  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  $i = 1, 2, 3, 4$ .

### 2.4 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 the following: 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; and *network connection* entities, such as protocols, interfaces, and network segments. 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.

We formalise the model elements of this layer and their associations with the DAS level in the following definitions.

**Definition 2.6** The ES level has the following elements:

- $Uc, Pc, H, So, Nc$ , sets correspondingly enclosing objects of types: *UserCredentials*, *Processes*, *Hosts*, *SystemObjects*, and *NetworkConnections* (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 2.7** 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;
- $sub : W \rightarrow Su$ , overriding of function *sub* to map the association of ES nodes to subsystems.

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.

### 2.5 Simple model example

A model example is shown in Fig. 3, in which a DAS (at the bottom) is represented together with the RO and SR levels. This model represents a typical network environment, for which three *AccessPermissions* are defined at the uppermost level (RO), in order to regulate the access rights of the users in the internal network with respect to e-mail transactions, as well as to allow the users to receive e-mails from the Internet. At the bottom of this figure, the DAS for this environment illustrates the concepts previously explained in this section.

Figure 4 shows the Expanded Subsystem for the AS “internal network” as an example. Comparing the simplified view (in the DAS of Fig. 3) and the detailed one (Fig. 4), it can be observed that modelling through abstract subsystems offers concrete advantages in the conciseness and understandability of the model, as well as providing an intelligible view of the system architecture.

## 3 Policy support and automated refinement

In MBM, after the input of a valid model instance, 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

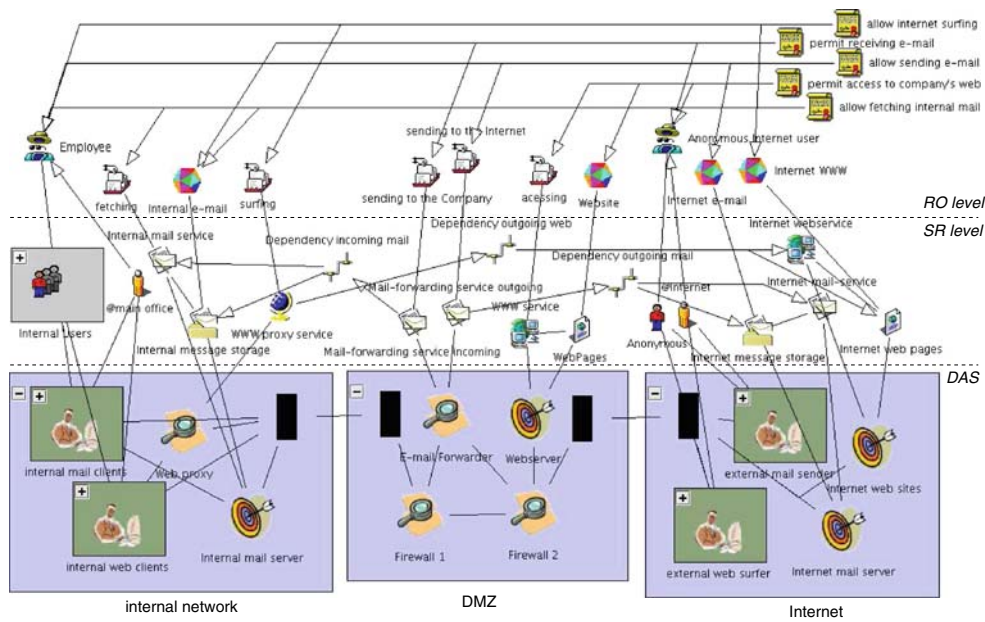


Fig. 3 Three-layered Model

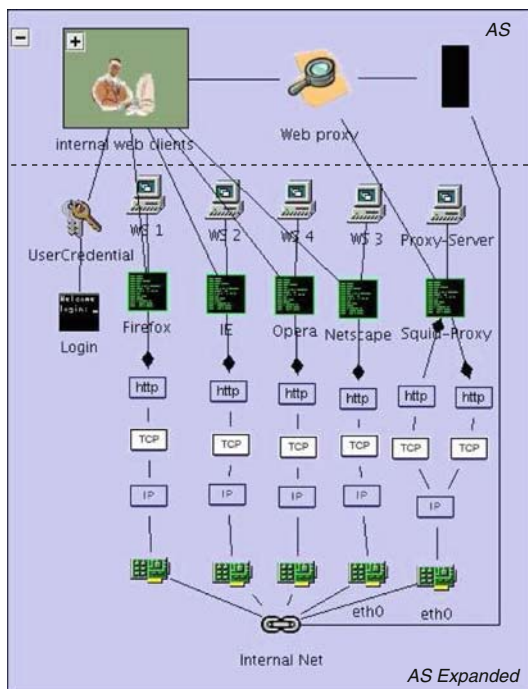


Fig. 4 “internal network” abstract and expanded subsystem (Fig. 3)

either *policy refinement* [1, 18] or *policy transformation* [25, 26]. In this work, we adopt the first notation.

The fully automated derivation of low-level, executable policies from a set of abstract specifications is, in the general case, not practical [24,26]. Nevertheless, as in MBM the system model is structured into 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.

Before presenting the refinement process in MBM, the next section analyses the policy support provided by the modelling and its semantics in comparison with a classification framework [24].

### 3.1 Policy support and semantics

According to Sloman and Lupu [24], there are two basic types of policies: *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 (IDSs), and watchdogs.

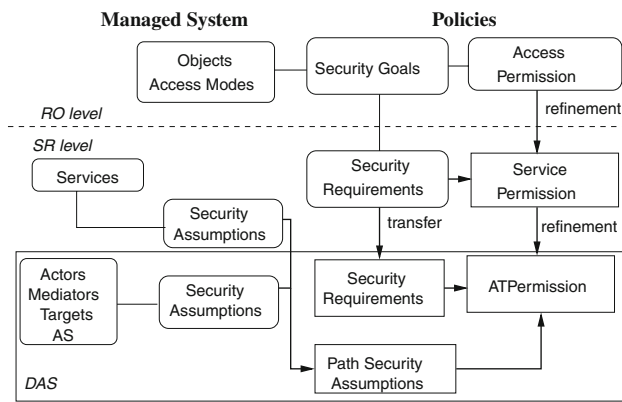


Fig. 5 Policies and security requirements in the system

An overview of the hierarchical structure of the policies supported in our modelling is presented in Fig. 5. 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 in the previous sections), the objects generated during the automated refinement process are depicted as normal rectangular boxes. As for the connecting lines, the thicker, arrowed ones represent associations automatically established during the refinement process, while the thinner lines with no arrows stand for the assignments given by the modeller. In the right hand columns of this figure, one can notice that the policy refinement is subdivided in two parallel tracks, corresponding to the two policy types supported by the modelling: *security goals and requirements*, and *authorisation policies*.

In the uppermost model level (RO), authorisation policies are represented by means of *AccessPermission* objects (Sect. 2.1). 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 [22], 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 a consequence, the positive and negative *authorisation policies* for users—i.e. the *user privileges*, or actions that users *may* or *may not* do—are also to be interpreted as *obligation policies* for the system—i.e. the *system duties* or actions 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. 2.1), which in its basic form does not enclose obligation policies (referred to as duties in [21]). However, besides the above policy elements, our modelling framework also includes an extension to the RBAC model by means of the *SecurityGoals*, *SecurityRequirements*, and *SecurityAssumptions* classes (Sect. 2.3). 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 to use either an encrypted tunnel or a plain text channel. 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 *SecurityRequirement* prescriptive associated to the *SecurityGoal* of its corresponding *AccessPermission*. The examples show that, although *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.

### 3.2 Automated refinement and configuration generation

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) and are defined as follows.

**Definition 3.1** The set of authorisation policies for the SR level is defined as:

- $SP \subseteq U \times St \times Sv \times R$ ;
- $sr : SP \rightarrow SC$ , a function that gives the security class required by a *ServicePermission* (see Sect. 2.3).

Along with the *SP* set of *positive* authorisation policies, we also define an auxiliary set of *negative* authorisation policies,  $\overline{SP}$ , which is the complement set of *SP*. The  $\overline{SP}$  set is not explicitly defined, but implicitly derived here due to the

semantic of *closed policies* employed in MBM as mentioned previously, in order to facilitate the notation in the validation framework below.

**Definition 3.2** The set of negative authorisation policies for the SR level is defined as

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

Subsequently, the *ServicePermissions* are refined into *AT-Permission* objects (actor-target permissions, ATP for short), which represent authorisation policies in a DAS (Sect. 2.2). 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  $(sv_r, res)$ . In fact, the path discovery must also consider some other constraints such as service dependencies and the compliance of security requirements—these are not explained here though, as they are discussed in detail later on in Sect. 4.

In order to formalise ATPpermissions, we begin with an auxiliary definition as follows.

**Definition 3.3** 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

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

The  $P$  set 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 a local DAS path, such that  $p_1 = \langle v_1, \dots, v_k \rangle$ ;
- (ii)  $p_2 \in P$  is a DAS path that *spans*  $x - 1$  subsystems, such that  $p_2 = \langle v_{k+1}, \dots, v_m \rangle$ ;
- (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  $E$ ).

Now, the *ATP* set 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  $\langle v_1, \dots, v_n \rangle \in P$ , where  $v_1 = a$  and  $v_n = t$ .

The set  $RATP \subseteq ATP \times SP$  formalises the refinement relations from service permissions to ATPs, i.e. it maps the association of policies of the SR level ( $SP$ ) with the corresponding policies derived for the DAS level (*ATP*).

The following refinement phase comprises the automated generation of policies that consider the equipment 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.

**Definition 3.4** 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

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

The  $EP$  set 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  $ep_1 = \langle v_1, \dots, v_k \rangle$ ;
- (ii)  $ep_2 \in EP$  is an expanded path that *spans*  $x - 1$  subsystems, such that  $ep_2 = \langle v_{k+1}, \dots, v_m \rangle$ ;
- (iii)  $c_1$  and  $c_2$  are connectors such that  $(c_1, c_2) \in E$  (i.e. there is an edge in DAS that connects  $c_1$  and  $c_2$ , see Definition 2.2),  $(v_k, c_1) \in NcC$ , and  $(v_{k+1}, c_2) \in NcC$ .

The *AEP* set of *Allowed Expanded Paths* (Sect. 3.2) 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.

The set  $RAEP \subseteq AEP \times ATP$  formalises the refinements from ATPs to AEPs, i.e. it represents the association of policies of the DAS level (*ATP*) with the corresponding policies derived for the ES level (*AEP*).

In the last phase of the refinement process, a special back-end function for each supported mechanism type 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 traverse the mechanism.

## 4 Validation approach

The consistency amongst abstraction levels of a policy hierarchy is a crucial issue. If the policy sets at the different levels are in perfect harmony, only then 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 [24], 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.

The reader should note that *completeness* concerns *positive* policies, whilst *consistency* deals with *negative* policies. As such, these two properties are complementary and thereby provide the necessary and sufficient criteria to ensure the propagation of the meanings conveyed by both policies and system model in the MBM approach (see Sect. 3.1). 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 work is to establish validation criteria for the automated refinement described in Sect. 3.2. As such, we do not aim at validating a particular refinement algorithm, but rather to come up with a general result that establishes necessary and sufficient conditions so that the application of the policy refinement to a model instance always complies with the aforementioned general validation criteria. Since the refinement correctness between the RO and the SR levels was already extensively studied by Lück [15], the topmost abstraction level (RO) is left out of the analysis here. This validation 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, i.e. a model yielded after the execution of a given refinement algorithm. This complete model is thus composed of both a group of system objects and a policy set for each of the SR, DAS, and ES levels. 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 amongst 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 produced configuration parameters.

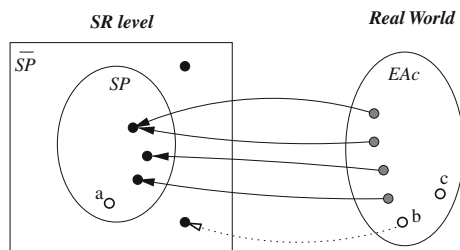
### 4.1 Validation overview

In the following sections we first establish the consistency conditions for a valid refinement from the SR to the DAS level in Sect. 4.2. These conditions are subdivided into two groups: refinement consistency conditions and structural consistency conditions. The first subgroup is presented in Sect. 4.2.1 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 (Sect. 4.2.2), in turn, establish the compatibility of the DAS structure (i.e. the managed system representation) with the *ATP* policy set.

Subsequently, Sect. 4.3 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 Sect. 4.3.1 and validate the ES policy set (*AEP*) in comparison to *ATP*. As at the previous level, the second subgroup (Sect. 4.4) 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 4.4.1 presents the composition consistency conditions, which validate the interconnection between ESs.

Furthermore, we prove in general, by means of the theorems and lemma in Appendix A, that the local conditions can be generalised relying on the assertion of the composition conditions. These theorems and the lemma do not have to be checked for each model instance as the other condition sets do.

In order to prove that the conditions are able to validate a model, we examine thereafter the application to a real



**Fig. 6** Relation between the SR level and the real-world

environment of the configuration generated using the policy refinement process. Section 4.5 presents assumptions about the model capability of representing the real world environment, the so called *model representativeness axioms*. Thus, two theorems are proved to follow from these axioms and from the consistency conditions:

*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.*

These two theorems establish a relation between the *input* for the refinement process (the policies of the SR level) and its ultimate *output* (the possible accesses in the real world that result from the implementation of the configuration generated in the refinement).<sup>1</sup> This relation is represented by the Venn diagram of Fig. 6. The input is depicted on the left-hand side by the  $SP$  set of service permissions (i.e. the positive authorisation policies), and its complement set,  $\overline{SP}$ , with the negative authorisation policies. 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 on the relation of the 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, valid elements of each set are represented by black or grey circles (for the input and output, respectively), whilst 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, an  $SP$  element such as  $a$ , which does not have a related element in  $EAc$ , will not exist if VT1 holds (considering that all  $SP$  elements have at least one corresponding access in the real world; see Axiom 2 in Sect. 4.5). Conversely, the existence of elements such as  $b$  and  $c$  in  $EAc$  would violate VT2. The  $b$  element represents a possible real-world access that has an abstract representation in the SR level which does not pertain to the  $SP$  policy set—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.

## 4.2 SR/DAS congruence

### 4.2.1 Refinement consistency conditions

The refinement validation from the SR to the DAS level must assert the consistency between the two levels in respect to both authorisation policies and security requirements. For this purpose, the first group of conditions below concerns the *RATP* refinement relation (Definition 2.3) between the set of authorisation policies in SR ( $SP$ ) and the analogous set in DAS ( $ATP$ ). To improve legibility, we will henceforth interchangeably use 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 *ATP permission*, and the symbol  $atp$  shall all indicate an element of the *ATP* DAS policy set.

Before presenting the first condition, the predicate  $atcomp$  is in sequence introduced in order to capture the compatibility between actors and targets of the model. A compatible actor-target pair 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 4.1** The predicate  $atcomp : A \times T \rightarrow \{0, 1\}$  denotes whether an actor-target pair 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 4.2** The predicate  $dep : Sv \times Sv \times R \rightarrow \{0, 1\}$ , indicates if a service depends on another to access a resource.

<sup>1</sup> Notice that, though the output of the refinement algorithms is the generated configuration parameters, we are considering here the whole refinement process and its ultimate output, i.e. the accesses enabled/disabled in the real system using the generated configuration parameters. The link between yielded configuration and system behaviour is established by means of the axioms in Sect. 4.5.

It is recursively defined as follows:

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

The first refinement consistency condition below aims to establish 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 DAS which are related to service permissions will be completely enabled by corresponding elements of the ATP policy set.

**Condition 4.2.1** Let  $sp = (u, st, sv_1, r)$  be a service permission in  $SP$ . For each actor,  $a \in A$  that refines the *user-subject type* pair  $(u, st)$ , and each target  $t \in T$  that refines a *service-resource* pair 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  $RATP$ :

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

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  $r$  resource. If the  $sv_1$  service has direct access to  $r$ , then  $sv_1$  and  $sv_2$  will be the same (see Definition 4.2).

The second condition below conversely asserts that to each DAS permission in the  $ATP$  set, 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, since each  $atp$  is required to be mapped to an authorising service permission.

**Condition 4.2.2** Let  $atp = \langle a, \dots, t \rangle$  be an  $ATP$  permission 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  $SP$  set must contain a service permission  $sp = (u, st, sv_2, r)$  that is related to  $atp$  through  $RATP$  and the  $sv_1$  service 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 Definition 4.2):

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

Subsequently, the third condition below shall validate the *correspondence* of pairs  $(atp, sp)$  contained in the  $RATP$  relation. 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. 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 logging enabled may improve the traceability of all communication flows that pass through it (in fact, encryption services that are related to Virtual Private Network mechanisms are not active on 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).

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

**Definition 4.3** Let  $atp = \langle v_1, \dots, v_n \rangle$  be an element of  $ATP$  and  $Sva \subset Sv$  the set of services associated to the elements of  $atp$ , such that  $Sva = \{sv \mid (v_j, sv) \in RM \vee (v_j, sv, r) \in RT \text{ for some } 1 \leq j \leq n\}$ . 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 Definition 2.5. It independently selects the greater security level for each dimension of the security classes, i.e. for each category of security requirements considered (see also Sect. 2.3). 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 an  $ATP$  permission. There are three security assumptions that must be considered: a) the overall security assumption of the  $ATP$  permission; 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 security classes from (a), (b), and (c)—just like a chord is the sonorous effect of several simultaneously produced tones.

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

**Definition 4.4** Let *v* be a DAS node, such that *atp* ∈ *ATP* contains *v*. The function  $esa : V \times ATP \rightarrow SC$  is thus defined as

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

To finish the consideration of security assumptions on 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 four categories of requirements, the assumption of a whole path cannot be picked from a single object. Rather, the weakest security level for each category must be independently determined amongst 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 4.5** 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  $atp = \langle v_1, \dots, v_n \rangle$  be an element of *ATP*; thus:

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

This definition relies upon a generalisation of the binary operator  $\prod$  declared in Definition 2.5. It selects the lowest values independently for each dimension of the security classes—thus yielding the desired result for the whole path.

Figure 7 illustrates the previous concepts with a path example and its associated security assumptions. The *p* path comprises four nodes (*v1–v4*) which are distributed over three different subsystems (*sub1*, *sub2* and *sub3*). The grey tones reflect the security assumption of each element, such that the darker the background colour of a given object, the higher (stronger) 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 clarity. For instance, the *v1* node is assumed to ensure

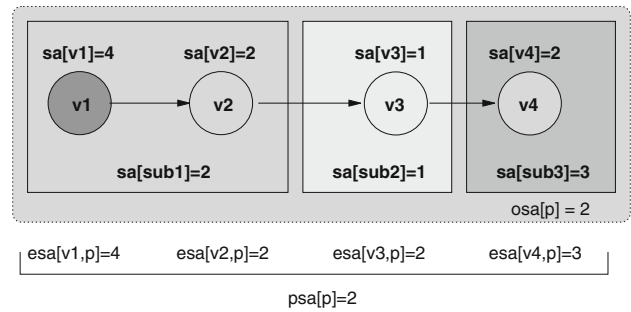


Fig. 7 Example of active security assumptions in a path

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, 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.  $psa[p]=2$ .

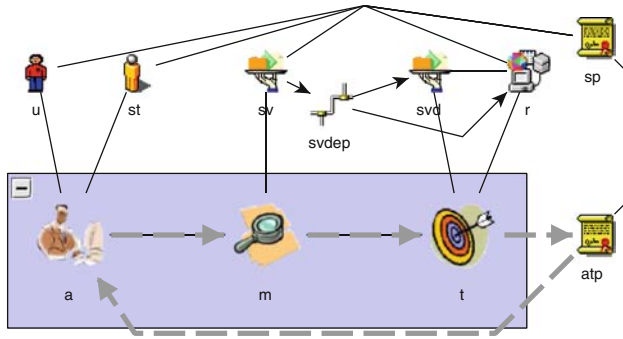
Finally, relying upon the aforementioned considerations, we can define the third refinement consistency condition as follows:

**Condition 4.2.3** Let  $sp = \langle u, st, sv, r \rangle$  be a service permission in *SP* and  $atp \in ATP$  an *ATPermission*, such that  $sp = \langle u, st, sv, r \rangle$ ,  $atp = \langle v_1, \dots, v_n, \rangle$ , 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 *user-subject* type pair  $\langle u, st \rangle$ ;
- (ii) the target  $t \in T$  must refine a *service-resource* pair like  $\langle sv_t, r \rangle$ ;
- (iii) actor *a* and target *t* must be compatible (Definition 4.1);
- (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 resource *r*, either  $sv_d$  is the  $sv_t$  service related to the target or a mediator that refines  $sv_d$  must be found along *atp*.

Formally stated:

$$\forall atp \in ATP, sp \in SP : atp = \langle v_1, \dots, v_n, \rangle \wedge sp = \langle u, st, sv, r \rangle \wedge (sp, atp) \in RATP \Rightarrow (v_1, u, st) \in RA \wedge (v_n, sv_t, r) \in RT \wedge atcomp[a, t] \wedge sr[sp] \leq psa[atp] \wedge \forall sv_d : dep[sv, sv_d, r] \Rightarrow$$



**Fig. 8** Example of correspondence between a service permission and an ATPPermission

$$sv_d = sv_i \vee \exists v_j : (v_j, sv_d) \in RM \text{ for } 1 < j < n$$

A generic example of an  $sp$  service permission and its corresponding  $atp$  is shown in Fig. 8. Each of the required items in Condition 4.2.3 can be seen to be satisfied by  $atp$  except for the security class compliance, which is not depicted by objects, but is represented in their properties.

#### 4.2.2 Structural consistency conditions

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

**Condition 4.2.4** 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  $(u, st)$ :

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

**Condition 4.2.5** 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  $(sv, r)$ :

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

**Condition 4.2.6 (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 contains 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 4.2.7** 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  $\langle v_1, c_1, c_2, v_2 \rangle$ :

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

### 4.3 DAS/ES congruence

#### 4.3.1 Refinement consistency conditions

**Definition 4.6** Consider a DAS path  $ap \in P$  in an expanded path  $ep \in EP$  (see Definitions 3.3 and 3.4). Let  $ap = \langle v_1, \dots, v_n \rangle$ , and let  $e_1, \dots, e_m$  be the ordered sequence of processes and connectors of  $ep$ , i.e. the sequence of 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:

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

**Condition 4.3.1** Let  $atp = \langle a, \dots, t \rangle$  be an element of  $ATP$ . Suppose there are two processes  $pc_a, pc_t \in Pc$ , such that  $pc_a$  refines the  $a$  actor and  $pc_t$  refines the  $t$  target. Thus, an allowed expanded path  $aep = \langle pc_a, \dots, pc_t \rangle$  must exist in  $AEP$ , such that  $aep$  is a refinement and a valid expansion of  $atp$ :

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

**Condition 4.3.2** 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  $atp$ :

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

#### 4.4 Local structural consistency conditions

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

**Definition 4.7** An expanded path in the model (Definition 3.4) 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 if 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 ones can act as gateways). All those details are irrelevant for the purposes of this validation and are thus abstractly treated by means of the above predicate.

**Definition 4.8** A local ES path (Definition 3.4) 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

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

which leads us to introduce a first set of conditions to deal with the DAS/ES refinement relations:

**Condition 4.4.1** Let  $v$  be an actor or target in DAS. Thus, a process  $pc \in Pc$  must exist that refines  $v$ :

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

**Condition 4.4.2** Let  $m$  be a mediator in  $M$ . Thus, a *unique* process  $pc \in Pc$  must exist that refines  $v$ :

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

**Condition 4.4.3** Let  $pc$  be a process in  $Pc$ . Thus, an element in DAS must exist that is refined by  $pc$ :

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

**Condition 4.4.4** Let  $uc$  be a 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  $(u, st)$ :

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

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

**Condition 4.4.5** Let  $v_1, v_2 \in V$  be elements of DAS, and let  $pc_1, pc_2$  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 (Definition 4.8):

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

**Condition 4.4.6** Let  $pc_1, pc_2 \in Pc$  be two processes that are connected by a locally connected ES path  $ep$  (Definition 4.8). 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  $(v_1, v_2) \in E$ :

$$\forall ep = \langle pc_1, \dots, pc_2 \rangle : lconn[ep] \\ \Rightarrow \exists v_1, v_2 \in V : (pc_1, v_1), (pc_2, v_2) \in RPc \wedge (v_1, v_2) \in E$$

#### 4.4.1 Composition consistency conditions

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

**Definition 4.9** An expanded path (Definition 3.4) 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:

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

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

**Condition 4.4.7** 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 the 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  $pc_2$ :

$$\forall v_1, v_2 \in V, c_1, c_2 \in C, pc_1, pc_2 \in Pc : \\ \langle v_1, c_1, c_2, v_2 \rangle \in P \wedge (pc_1, v_1), (pc_2, v_2) \in RPc \\ \Rightarrow \exists ep = \langle pc_1, \dots, pc_2 \rangle : cconn[ep]$$

**Condition 4.4.8** Let  $pc_1$  and  $pc_2$  be two processes that are connected by a connector-connected path  $ep$ . 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  $\langle v_1, c_1, c_2, v_2 \rangle$ :

$$\forall ep = \langle pc_1, \dots, pc_2 \rangle : cconn[ep] \\ \Rightarrow \exists v_1, v_2 \in V, c_1, c_2 \in C : \\ (pc_1, v_1), (pc_2, v_2) \in RPc \wedge \langle v_1, c_1, c_2, v_2 \rangle \in P$$

#### 4.5 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 to represent. 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. Stated 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.

#### 4.5.1 Accesses and their abstract representations

As explained in Sect. 4, 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. To this concern, the relevant sets are formalised by the following definition.

**Definition 4.10** 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 through all security mechanisms along the network path between the initiator and responding processes.

The explanation above already anticipates the relation between an 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  $Ac$  set can be represented as a relation on the respective ES-level sets:

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

It is worth remembering here that a process in the ES level represents a *prototype* for actual processes that might be started in the real environment (Sect. 2.4). As such, the relation established by Axiom 1 between processes and accesses in the real environment imposes that the latter will also stand for the *prototype* of actual accesses, or as stated previously, 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.

An observation is due here, since the axiom states that all potential accesses are a subset of the set of 4-tuples linking user credentials, processes and system objects. From an administrator's point of view, this is at first glance hard to accept, for mapping all possible accesses on a given network scenario is a daunting task due to the large number of alternatives, be them simple or more complex ones, and that knowledge would not be available to the administrator. It must be clear, then, that the set of 4-tuples to be considered will normally turn out to be quite smaller than initially thought and the statement will make sense. The reason is that the administrator, when modelling the system, does not have to cover all possible accesses the network technology allows, but rather only the accesses the security policy to be implemented should allow. Security is achieved because forbidden accesses (the complement of the set of policy-allowed accesses) are automatically blocked due to the adoption of closed policies (see fourth paragraph of Sect. 3.1).

Hence, the axiom establishes that each access to be considered can be described in terms of system objects in this way: the  $pc_i$  process, on behalf of the  $uc$  user credential, communicates with the  $pc_r$  process that manipulates the  $so$  system object. But we can just as well describe the access by means of the corresponding abstract entities like this: the  $u$  user logged in the system as the  $st$  subject type is using the  $sv$  service in order to access the  $r$  resource. And these two quite different expressions refer to one and the same phenomenon in the real world—an access—though 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 Definition 4.11 as follows.

**Definition 4.11** The correspondence between an access in the real environment and the SR level is denoted by the function  $\text{abstract-rep} : Ac \rightarrow U \times St \times Sv \times R$ . The 4-tuple  $(u, st, sv, r)$  returned by the function is the abstract representation of a given access in the set  $Ac$ .<sup>2</sup>

The reader may be asking him/herself at this point: why is the access mapped to a representation at the SR level? Why not any other model level? This choice is not arbitrary;

<sup>2</sup> Functions in this section are assumed to exist in the environment and stand out with a different typeface.

instead, it is due to the fact that the SR level is the top-most level considered in the validation (see the beginning of Sect. 4). As such, the SR representation offers 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 of such 4-tuples, there will usually be 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. 4.2.1) are by purpose left out of the analysis in this section and in the validation theorems of Sect. 6.

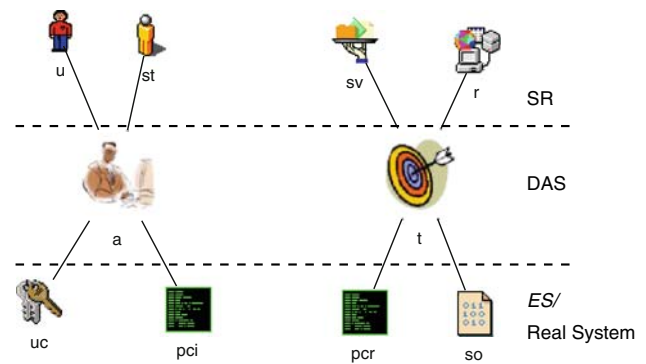
This is done in order to simplify the assumptions and proofs, such that one can more clearly perceive their intent and meaning. Nevertheless, these results and assumptions could simply be changed to cover dependencies by the addition of extra conditions to the statements. Therefore, leaving dependencies out does not consist on a limitation for the validation presented here.

From this background, we can reinterpret the *SP* set 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*. In a formal notation:

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

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  $(uc, pci, pcr, so)$  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 Fig. 9. Hence, an *a* actor is connected to both the user credential and initiator process pair



**Fig. 9** Correspondence between accesses in the real environment and the SR level

$(uc, pci)$ , and to the user and subject type pair  $(u, st)$ —thus establishing the correspondence between them. Analogously, a *t* target is connected both to the responding process and system object pair  $(pcr, so)$  and to the service and resource pair  $(sv, r)$ .

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.

**Axiom 3** The model structure correctly represents the correspondence between accesses in the real environment and their abstract representations. Let  $(uc, pci, pcr, so) \in Ac$  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 user credential and initiator process pair  $(uc, pci)$ , and to the user and subject type pair  $(u, st)$ ;
- (ii) a target that is connected both to the responding process and system object pair  $(pcr, so)$  and to the service and resource pair  $(sv, r)$ ;
- (iii) actor *a* and target *t* must be compatible (see Definition 4.1);
- (iv) a host that is connected both to the  $pcr$  responding process and to the *so* system object.

Formally stated:

$$\begin{aligned} (u, st, sv, r) &= \text{abstract-rep} [(uc, pci, pcr, so)] \\ &\Leftrightarrow \exists a \in A, t \in T : (a, u, st) \in RA \wedge (t, sv, r) \in RT \\ &\wedge atcomp[a, t] \\ &\wedge (uc, a, u) \in RUC \wedge (pci, a), (pcr, t) \in RPC \\ &\wedge (so, t) \in RSO \wedge samehost[pcr, so] \end{aligned}$$

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

**Definition 4.12** 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:

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

#### 4.5.2 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 4.13** The following predicates map the system behaviour in the real environment:

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

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

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

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

Concerning the possibility of a user credential starting 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, formalised in the next axiom.

**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  $a$ :

$$can-use[uc, pc] \Leftrightarrow \exists a \in A, u \in U : (uc, a, u) \in RUC \wedge (pc, a) \in RPC$$

Notice that the above axiom requires the system to be able to correctly identify and authenticate 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 [20]), this work assumes that identification and authentication of users take place in a secure and correct manner, and the main concern is with what happens 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, though connected to the same target, being placed on different machines, an assumption formalised as follows:

**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:

$$can-handle[pc, so] \Leftrightarrow samehost[pc, so] \wedge \exists t \in T : (pc, t) \in RPC \wedge (so, t) \in RSo$$

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 correctly enforce 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. This assumption requires the correct implementation of the configuration according to the lowest-level policies in the model, as in the next 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  $pc_1, \dots, pc_n$  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:

- the  $uc$  credential can be used to access the  $pc_1$  process;
- process  $pc_n$  can handle the  $so$  system object;
- there is an allowed expanded path  $aep \in AEP$  in the model, such that  $aep = \langle pc_1, \dots, pc_n \rangle$ , i.e.  $aep$  contains exactly the same process sequence  $pc_1, \dots, pc_n$  of  $eac$ .

Formally stated:

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

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:

**Definition 4.14** The function  $\text{pc-esa} : Pc \times Eac \rightarrow SC$  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 Definition 8.2 in Sect. A.3. 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, according to the following axiom:

**Axiom 7** Let  $eac \in EAc$  be an enabled access in the real environment, such that  $eac = (uc, pc_1, pc_n, so)$  and  $pc_1, \dots, pc_n$  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  $pc_1, \dots, pc_n$  (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  $aep$  (see Definition 8.2 in Sect. A.3). Formally stated:

$\forall eac \in EAc, aep \in AEP :$

$$eac = (uc, pc_1, pc_n, so), aep = \langle pc_1, \dots, pc_n \rangle \\ \Rightarrow \text{pc-esa}[pc_j, eac] = \text{esa}'[pc_j, aep] \quad \text{for } 1 \leq j \leq n$$

Finally, we must consider the security class one can expect of an access as a whole. Analogously to the  $psa$  and  $psa'$  functions (Definitions 4.5 and 8.3), 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 4.15** The function  $\text{acc-psa} : Eac \rightarrow SC$  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  $pc_1, \dots, pc_n$  is the sequence of processes in the communication flow of  $eac$ , thus:

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

## 5 Practical application and use example

The approach has shown its practical relevance in a series of case studies [7,8,10]. A supporting tool was implemented and has been employed for the integrated configuration management of packet filters and VPN gateways of realistic network environments, with different number of network elements and growing security policy complexity. Back-end functions were implemented for the generation of configuration files of the corresponding mechanisms of the OpenBSD operating system (*pf* and *isakmpd*), successfully covering the basic functionalities of these mechanisms.

A comprehensive covering of these application cases lies outside the scope of the present paper as it is done elsewhere [7,8,10]. To give a flavour of the results achieved, though, Fig. 10 shows a simple model example produced with the associated software tool that illustrates the practical use of our approach in a common situation. The high-level policies modelled allow the employees of a branch office to access the company's web server located at the main office's network, which is accessible via the Internet. However, the security requirements associated to the service and resources of this example require security levels that are not fulfilled when the communication is performed directly through the Internet, since the security assumption of the AS "Internet" (1,1,1,1) is lower than required one (2,2,2,2). Thus, the tool automatically finds out and selects the path that connects the main office with the branch office via a VPN tunnel established by the two VPN gateways. In Fig. 10 the policy objects (Sect. 3.2) *ServicePermission*, *ATPermission*, and the allowed expanded path *ADP* are automatically generated by the tool.

In the example of Fig. 10, the DAS path of *ATPermission*—which goes from *intern Client* to *Webserver* passing through *VPN Gateway 1*, three connectors, and *VPN Gateway 2*—is able to fulfil the requirements due to the tunnel established by *EncryptionService* between *VPN Gateway 1* and *VPN Gateway 2*. Clearly, should the requirements be fulfilled without the VPN tunnel—for example, if the requirements were set to (1,1,1,1)—the refinement algorithm would again choose the shortest path available, this time effectively bypassing the two VPN gateways and not generating any configuration for processes "isakmpd 1" and "isakmpd 2".

In order to generate the configuration files from the derived lower-level path (ADP), 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.
- The *IP address* used for the communication with the other gateways must be inputted in the corresponding

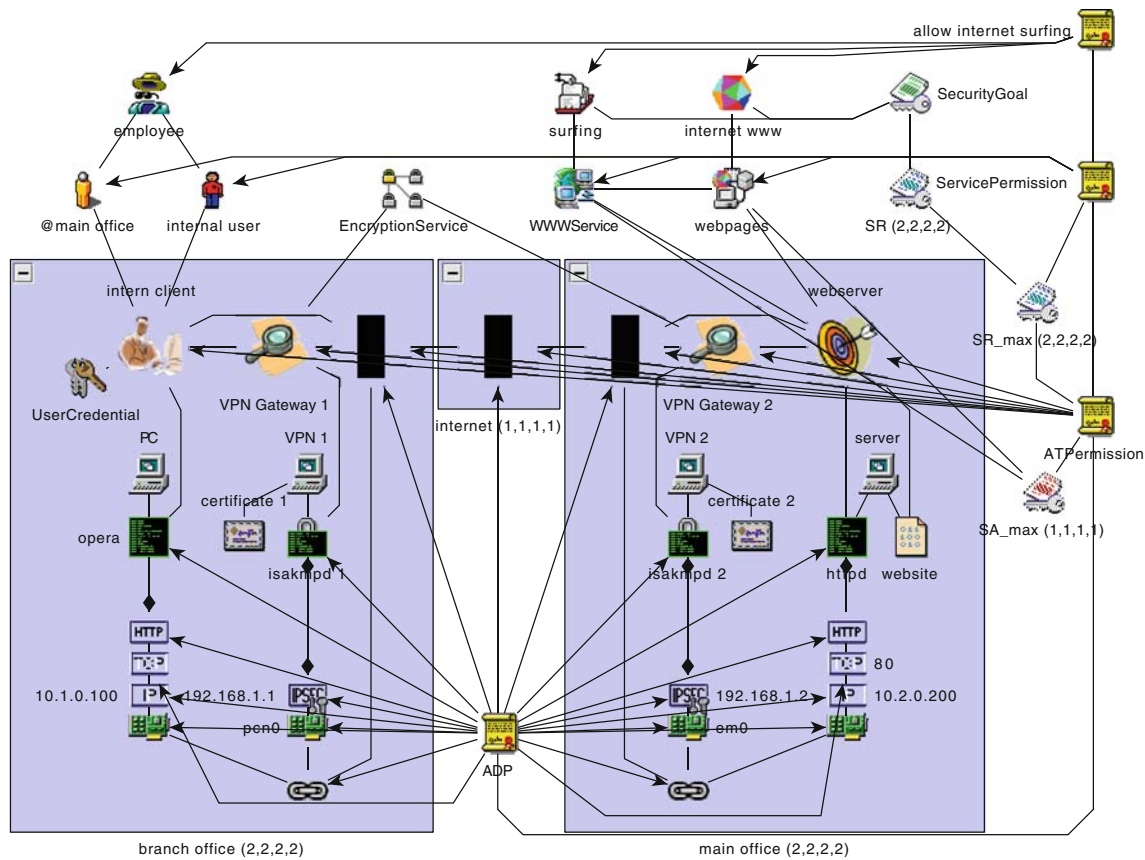


Fig. 10 Simple VPN model after the automated refinement

property of the *IPSec* protocol object that is associated to the VPN process.

- The *network address* and *network mask* of the network to which the VPN gateway is connected must be provided in the corresponding object properties.

As such, the implemented tool generates configuration files for the two VPN gateways based on the analysis of the model and on the provided information. Additionally, the tool also enforces the validation conditions defined in Sect. 4.

From the standpoint of the practical application, these conditions 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 an 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. This first group consists of the *pre-conditions* for the algorithm of the respective refinement step (SR/DAS or DAS/ES). The second condition group comprises conditions that must be present in the model after the execution of the refinement algorithm at a given step. Therefore, they are the *post-conditions* for the respective refinement algorithm and

are thus achieved by the algorithm. The third condition group is in turn composed of additional conditions that must be verified after the refinement algorithm is performed (for details in the implementation of the conditions into the software tool refer Porto de Albuquerque [6]).

The enforcement of these conditions as a whole is 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 given network systems are often not able to enforce the claimed high-level policies unless additional protection mechanisms are introduced. In such cases, some of the validation conditions defined here cannot be satisfied, and the tool indicates the model elements which violate the policies. This way the modeller receives valuable information to design the necessary extensions to the system in order to make it congruous to the policies.

We can illustrate this still with the example of Fig. 10: if there were no VPN gateways in any of the networks, the tool would inform the user that no path could be found to satisfy the security requirements. The user would thus find out that there is a need to add a security mechanism that provides encryption services, thereby supporting the security level required for implementing the high-level security policies.

## 6 Discussion of the validation results and soundness

Based on the axioms of Sect. 4.5, the validation theorems proposed in Sect. 4.1 are now formalised.

**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:*

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

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

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

The two theorems can be shown to follow from the conditions and axioms previously defined—which is accomplished in Appendix B. By analysing the two validation theorems, one can conclude that they ensure the compliance of the policy sets with the validation criteria defined in the beginning of Sect. 4. 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 in the model. These criteria are cited in the literature as the most important ones to assure the refinement *correctness* in a multi-layered policy hierarchy (refer for instance, Abrams and Bailey [1] and Sloman and Lupu [24]). 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. 3.1)—be propagated and maintained from one abstraction level to its immediate inferior neighbour.

Therefore, the formal proofs mean that: provided that a model (including system and policy representation after a process refinement) fulfils the defined condition sets, and that 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 [22], which allows only those accesses enabled by the access control policies.

The computational effort required by the verification of the defined conditions has been analysed in [6]. To that

respect, the DAS level can afford to split up the system analysis, i.e. it allows the validation to be performed in a modular fashion, which is more appropriate when dealing with large models. Indeed, the DAS/ES structural conditions have a local scope and thus limit to consideration only the elements inside the boundaries of one subsystem at a time, 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.

It should be noted 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. [3], analysed in Sect. 7), but rather implicitly assumed. The model representativeness axioms of Sect. 4.5 thus serve to make 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 by adding contextual information, thus having a subordinate character. This means that, in isolation, they do not have a meaning of their own, rather they only acquire significance together with an access permission by modifying (restricting) the modelled authorisation. They act just like adverbs in the natural language, which are used to imprint a different connotation to verbs and adjectives. 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 [20], as the latter have a much stronger, independent character (although there is of course some resemblance between the two concepts).

Furthermore, the requirement support is left 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 [4], by adjusting the vector dimension and the level range in order to map families and components in each of those classes. For the purpose of the present work

though, the simplified approach that was adopted is sufficient.

## 7 Related work

Guttman [13] presents an approach that defines a security policy for packet filters in a logic-based language, as a global policy about which packets can get where. It has the advantage of offering an additional method for solving the localisation problem—i.e. to determine the filtering decisions of individual routers—and an automated verification method for the localisation problem. Burns et al. [3] 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 network state against the specified policy. These two 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.

The graphical tool Firmato [2] offers a more intuitive approach, since it supports the interactive policy design by means of diagrams, and automatically derives the corresponding configurations for mechanisms. However, this and the other approaches aforementioned are mostly concerned with a low-level view of the management problem. These works 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. On the other hand, there are already some approaches that turn the problem onto the right direction by starting from convenient abstract models that are able to address the management problem from a business point of view.

The Power prototype [19] aims at supporting the building of policy hierarchies by means of a tool-assisted policy refinement. The tool is then able to refine the high-level policy according to pre-defined templates and finally producing the configuration information for particular devices. Cuppens et al. [5] present a more formal approach for specifying network security policies, which is based on the Or-BAC model, an extension of the Role-Based Access Control Model (RBAC). 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. However, both approaches lack analysis capabilities, so the problem of 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.

## 8 Conclusion

This paper introduces a formal approach to the validation of policy hierarchies for the model-based management (MBM) of the configuration of network security systems. This validation builds upon a formalism for the modelling framework that appeared in previous works [7,8], in order to establish general conditions that a given system and policy model must uphold to assure the *correctness* of the policy refinement, i.e. general validation conditions for ensuring that the generated lower-level policies uphold the abstract policies defined by the user. The conditions were based on the general criteria of *completeness* and *consistency*, so that the normative meaning conveyed by policies and system model in MBM (analysed in Sect. 3.1) 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. We also implemented a software tool that performs the automated policy refinement and enforces the validation conditions.

To be able to reason about the system behaviour that results from the application of the generated configuration, *model representativeness axioms* were defined in this paper 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, we proved two *validation theorems* that 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 MBM

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 security policies, i.e. they collectively behave as an integrated system that delivers the intended security services. On the other hand, we believe that the validation framework proposed here is general enough to be of value in other contexts in which policy-based management and policy hierarchies are used, particularly with automated policy refinement algorithms that must enforce consistency and completeness between different policy layers.

## Appendix A: 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 now must consider these layers as a whole, thus achieving a generalised result from the previous considerations.

First, we define a generalised predicate based on Definitions 4.8 and 4.9.

**Definition 8.1** A path is considered *connected* if it is either *locally connected* (Definition 4.8) or *connector connected* (Definition 4.9). 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 generalise the previous results, by means of the theorems presented in the next sections.

### A.1 Generalisation Theorem 1

**Generalisation Theorem 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  $pc_1, \dots, pc_m$  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  $pc_1, \dots, pc_m$ . Formally stated:*

$\forall dp \subseteq aep \in AEP :$

$pc_1, \dots, pc_m$  is the sequence of processes in  $dp \Rightarrow$   
 $\exists p = \langle pc_i, \dots, pc_{i+1} \rangle : conn[p] \text{ for } 1 \leq i < m$

*Proof* 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 3.4). Thus, all processes along  $dp_1$  pertain to the same AS, and the basis

hypothesis can be stated as

$dp_1 \subseteq aep_1 \in AEP :$

$pc_1, \dots, pc_m$  is the process sequence in  $dp_1$   
 $\wedge sub[pc_1] = sub[pc_2] = \dots = sub[pc_m]$  (1)

Hence, applying this hypothesis to Condition 4.3.2, it follows that there is a corresponding abstract path  $ap_1$  in DAS, such that

$ap_1 \subseteq atp_1 \in ATP,$

$ap_1 = \langle v_1, \dots, v_m \rangle : expand[dp_1, ap_1]$

The definition of the predicate  $expand$  (Definition 4.6) implies that

$(e_i, v_i) \in R Pc \vee (e_i = v_i)$  for  $1 \leq i \leq n$  (2)

where  $e_1, \dots, e_n$  is the sequence of processes and connectors of  $dp_1$ . But the hypothesis in (1) states that all elements of  $dp_1$  pertain to the same AS, so that the structure of  $AEP$  elements (described in Definition 3.4) implies that there are no connectors in  $dp_1$  (i.e.  $n = m$ ). Hence, the right-hand term of the disjunction in (2) is false for all elements—since only connectors can be contained in both an element of  $ATP$  and an element of  $AEP$ . Equation (2) then becomes

$(pc_i, v_i) \in R Pc$  for  $1 \leq i \leq m$  (3)

Considering now a pair of subsequent DAS elements in  $ap_1$ , say  $v_j$  and  $v_{j+1}$ , Condition 4.4.5 asserts that

$\forall pc_a, pc_b \in Pc : (pc_a, v_j) \in R Pc$

$\wedge (pc_b, v_{j+1}) \in R Pc \Rightarrow$

$\exists p = \langle pc_a, \dots, pc_b \rangle : lconn[p]$  (4)

Clearly, the universal quantifier in (4) implies that also for  $pc_j$  and  $pc_{j+1}$  that are respectively connected to  $v_j$  and  $v_{j+1}$  by (3):

$\exists p = \langle pc_j, \dots, pc_{j+1} \rangle : lconn[p]$  (5)

Since  $pc_j$  and  $pc_{j+1}$  were chosen without loss of generality, (5) is valid for each pair of subsequent processes in  $dp_1$ . Definition 8.1 asserts that a local-connected path is also connected, so that we prove, as intended, that for the basis case

$\exists p = \langle pc_i, \dots, pc_{i+1} \rangle : conn[p]$  for  $1 \leq i < m$

□

For the general case, let us assume

$dp_2 \subseteq aep \in AEP :$

$pc_1, \dots, pc_o$  is the process sequence in  $dp_2, s$   
 $dp_2$  spans  $k$  ASs (6)

The induction hypothesis then reads:

*Theorem 1 is valid for all  $dp \subseteq atp \in ATP :$*

*$dp$  spans up to  $k - 1$  ASs (7)*

Proceeding in analogy with the basis case, from (6), Condition 4.3.2, and Definition 4.6 follows that there is a corresponding  $ap_2$  in DAS, such that

$$ap_2 \subseteq atp_2 \in ATP, ap_2 = \langle v_1, \dots, v_p \rangle : (e_i, v_i) \in RPC \vee (e_i = v_i) \text{ for } 1 \leq i \leq p \quad (8)$$

where  $e_1, \dots, e_p$  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  $e_1 = pc_1, \dots, e_{r-1} = pc_{r-1}$ , 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 3.4,  $dsp_1$  is thus completely enclosed in one subsystem, i.e.  $sub[pc_1] = sub[pc_2] = \dots = sub[pc_{k-1}]$ . Furthermore, the same Definition 3.4 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  $dsp_1$ , which gives

$$\exists p = \langle pc_j, \dots, pc_{j+1} \rangle : conn[p] \text{ for } 1 \leq j < r - 1 \quad (9)$$

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 (8) there is a corresponding DAS subpath in  $ap_2$ ,

$$asp_1 = \langle v_{r-1}, v_r, v_{r+1}, v_{r+2} \rangle : (v_{r-1}, v_r) \in E \wedge (v_r, v_{r+1}) \in E \wedge (v_{r+1}, v_{r+2}) \in E \quad (10)$$

and

$$(e_{r-1}, v_{r-1}) \in RPC \wedge e_r = v_r \in C \wedge e_{r+1} = v_{r+1} \in C \wedge (e_{r+1}, v_{r+1}) \in RPC \quad (11)$$

Thus, by applying (10) to Condition 4.4.7 comes

$$\forall pc_x, pc_y \in Pc : (pc_x, v_x) \in RPC \wedge (pc_y, v_y) \in RPC \Rightarrow \exists p = \langle pc_x, \dots, pc_y \rangle : cconn[p] \quad (12)$$

Clearly, the universal quantifier in (12) and (11) imply that

$$\exists p = \langle e_{r-1}, \dots, e_{r+2} \rangle : cconn[p] \quad (13)$$

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 8.1, we can then extend (9) to

$$\exists p = \langle pc_j, \dots, pc_{j+1} \rangle : conn[p] \text{ for } 1 \leq j < r \quad (14)$$

Now, let  $dsp_3$  be the subpath of  $dp_2$  that goes from  $e_{r+2} = pc_r$  to the end ( $e_p$ ). Since (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  $\langle e_1, \dots, e_r \rangle$ ),  $dsp_3$  thus spans  $k - 1$  subsystems. Hence, the induction hypothesis (7) can be applied to  $dsp_3$ , achieving

$$\exists p = \langle pc_l, \dots, pc_{l+1} \rangle : conn[p] \text{ for } r \leq l < n \quad (15)$$

Therefore, from (14) and (15) follows

$$\exists p = \langle pc_j, \dots, pc_{j+1} \rangle : conn[p] \text{ for } 1 \leq j < n$$

□

## A.2 Generalisation Theorem 2

**Generalisation Theorem 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:*

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

*Proof* Relying upon Definition 8.1, the theorems' assertion can be expanded into two subgoals to be proved:

$$\forall pc_x, pc_y \in Pc : \exists p_1 = \langle pc_x, \dots, pc_y \rangle \wedge lconn(p_1) \Rightarrow \exists aep_1 \in AEP : pc_1, \dots, pc_m \text{ is the sequence of processes in } aep_1 \wedge pc_x = pc_i \wedge pc_y = pc_{i+1} \text{ for } 1 \leq i < m \quad (16)$$

and

$$\forall pc_z, pc_w \in Pc : \exists p_2 = \langle pc_z, \dots, pc_w \rangle \wedge cconn(p_2) \Rightarrow \exists aep_2 \in AEP : pc_1, \dots, pc_o \text{ is the sequence of processes in } aep_2 \wedge pc_z = pc_j \wedge pc_w = pc_{j+1} \text{ for } 1 \leq j < o \quad (17)$$

Beginning with (16) and from Condition 4.4.6, it follows that

$$\exists v_x, v_y \in V : (pc_x, v_x) \in RPC \wedge (pc_y, v_y) \in RPC \wedge (v_x, v_y) \in E \quad (18)$$

Condition 4.2.6 and (18) thus imply

$$\exists atp_1 \in ATP : \langle v_x, v_y \rangle \subseteq atp_1 \quad (19)$$

Therefore, if we consider  $atp_1 = \langle v_1, \dots, v_n \rangle$  then  $v_x = v_i$  and  $v_y = v_{i+1}$  for some  $1 \leq i < n$ . Now, from Condition 4.3.1 follows

$$\forall w_1 \in Pc, w_m \in Pc : (w_1, v_1) \in RPC \wedge (w_m, v_n) \in RPC \Rightarrow \exists aep \in AEP, aep = \langle w_1, \dots, w_n \rangle : expand[aep, atp_1] \quad (20)$$

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 2.2) 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 (20) implies that also for  $pc_x$  and  $pc_y$ , correspondingly related to  $v_x = v_i$  and  $v_y = v_{i+1}$  by (18):

$$\exists aep_1 \in AEP, aep_1 = \langle pc_x, \dots, pc_y \rangle$$

□

In the second case,  $i > 1$  and  $i + 1 < n$ , and Definition 2.2 implies  $v_i \in M$  and  $v_{i+1} \in M$  (both are *Mediators*). Hence, Condition 4.4.2 implies that there is only one process related to each mediator, so that from (18), (20), and Definition 4.6 (definition of function *expand*) follows that

$$\exists aep_1 \in AEP, aep = \langle w_1, \dots, pc_x, \dots, pc_y, \dots, w_n \rangle \quad (21)$$

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 (17), we must apply Condition 4.4.8, achieving

$$\begin{aligned} \exists v_z, v_w \in V, c_1, c_2 \in C : (pc_z, v_z) \in R Pc \\ \wedge (pc_w, v_w) \in R Pc \wedge (v_z, c_1) \in E \wedge (c_1, c_2) \\ \in E \wedge (c_2, v_w) \in E \end{aligned} \quad (22)$$

and from (22) and Condition 4.2.7 comes

$$\exists atp_2 \in ATP : \langle v_z, c_1, c_2, v_w \rangle \subseteq atp_2 \quad (23)$$

Therefore, if we consider  $atp_2 = \langle v_1, \dots, v_n \rangle$  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 Condition 4.3.1 thus follows

$$\begin{aligned} \forall w_1 \in Pc, w_o \in Pc : (w_1, v_1) \in R Pc \\ \wedge (w_o, v_n) \in R Pc \Rightarrow \\ \exists aep \in AEP, aep = \langle w_1, \dots, w_o \rangle : expand[aep, atp_2] \end{aligned} \quad (24)$$

Hence, four analogous subcases to the ones in the first part of this proof must be considered 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)  $k > 1 \wedge k + 3 = n$ . As previously, we can prove for all four subcases that

$$\exists aep_2 \in AEP, aep_2 = \langle w_1, \dots, pc_z, \dots, pc_w, \dots, w_o \rangle$$

□

### A.3 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. 4.2.1).

**Definition 8.2** 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:

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

Notice that Condition 4.4.3 implies that a related DAS element  $v$  must exist for each ES process, thus assuring a value for the function  $esa'$  in the second case of the aforementioned definition.

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 Definition 4.5).

**Definition 8.3** 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  $e_1, \dots, e_m$  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):

**Generalisation Lemma 1** 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 to that of  $atp$ :

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

*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  $atp = \langle v_1, \dots, v_n \rangle$  and let  $e_1, \dots, e_m$  be the sequence of processes and connectors of  $aep$ . According to Definition 8.3, it follows that

$$psa'[aep] = \prod_{1 \leq j \leq m} esa'[e_j, aep] \quad (25)$$

Since  $aep$  is related to  $atp$ , from Condition 4.3.2 results that  $expand[aep, atp]$  must be valid. Then relying upon the definition of *expand* (Definition 4.6), we conclude that for each

$e_j$  in the equation aforementioned, if it is a process then there is an associated  $v_j$  in  $atp$ —i.e.  $(e_j, v_j) \in R P c$ —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 hold. Equation (25) can now be rewritten as

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

and this is exactly what results from the definition of  $psa[atp]$  (Definition 4.5).  $\square$

## Appendix B: Proofs of the validation theorems

### B.1 Proof of VT1

We want to prove that, for each policy in the SR level, the system enables all accesses in the real environment that correspond to the policy:

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

To prove this general assertion, let us consider, without loss of generality, a service permission  $sp_1$  and its related set of accesses in the real environment  $A_{sp_1}$ , such that

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

Axiom 2 implies that  $Ac^{sp_1}$  is not empty, so let us consider, again without loss of generality, an element  $ac_1$  of  $Ac^{sp_1}$ , such that  $ac_1 \in Ac^{sp_1}$ , 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 was generally chosen, the result can be generalised, and VT1 will thereby be proved.

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

$$\exists a_1 \in A, t_1 \in T : \\ (a_1, u_1, st_1) \in RA \wedge (t_1, sv_1, r_1) \in RT \\ \wedge atcomp[a_1, t_1] \tag{26}$$

$$\wedge (uc_1, a_1, u_1) \in R U c \wedge (so_1, t_1) \in R S o \\ \wedge samehost[pc, so] \tag{27}$$

$$\wedge (pci_1, a_1), (pcr_1, t_1) \in R P c \tag{28}$$

By applying Condition 4.2.1 to (26) it thus follows that there is an  $ATPermission$  between the actor  $a_1$  and target  $t_1$ ,

$\exists atp_1 \in ATP, atp_1 = \langle a_1, \dots, t_1 \rangle : (sp_1, atp_1) \in RATP$  therefore Condition 4.3.1 implies that from the expression above follows

$$\forall w_1 \in P c, w_m \in P c : (w_1, a_1) \in R P c \\ \wedge (w_m, t_1) \in R P c \Rightarrow \\ \exists aep \in AEP, aep = \langle w_1, \dots, w_n \rangle : expand[aep, atp_1]$$

Hence, considering (28) it results that the universal quantifier in the previous expression also applies to the processes  $pci_1$  and  $pcr_1$ , so that

$$\exists aep_1 \in AEP, aep_1 = \langle pci_1, \dots, pcr_1 \rangle \\ : expand[aep_1, atp_1] \tag{29}$$

Axiom 4 implies that from  $(uc_1, a_1, u_1) \in R U c$  in (27) and  $(pci_1, a_1) \in R P c$  in (28) follows

$$\mathbf{can-use}(uc_1, pci_1) \tag{30}$$

Analogously, Axiom 5 implies that from  $(so_1, t_1) \in R S o \wedge samehost[pc, so]$  in (27) and  $(pcr_1, t_1) \in R P c$  in (28) we can conclude:

$$\mathbf{can-handle}(pcr_1, so_1) \tag{31}$$

By applying Axiom 6 to (29), (30), and (31) it thus comes that  $ac_1 \in EAc$ , i.e. the access is enabled by the system in the real environment.  $\square$

### B.2 Proof of VT2

We want to prove that for each enabled access in the real environment (element of  $EAc$ ), there is a corresponding policy at the SR level, i.e. a corresponding element in  $SP$ . 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

$$\mathbf{can-use}[uc_1, pci_1] \tag{32}$$

$$\mathbf{can-handle}[pcr_1, so_1] \tag{33}$$

$$\exists aep_1 \in AEP : aep_1 = \langle pci_1, \dots, pcr_1 \rangle \tag{34}$$

By applying Condition 4.3.2 to (34) it follows that a corresponding  $ATPermission$  must exist in the model:

$$\exists ! atp_1 \in ATP, atp_1 = \langle a_1, \dots, t_1 \rangle \\ : expand[aep_1, atp_1] \tag{35}$$

On the other hand, from (32) and Axiom 4 it comes that

$$\exists a \in A, u \in U : (uc_1, a, u) \in R U c \wedge (pci_1, a) \in R P c$$

The definition of the predicate  $expand$  (Definition 4.6) and (35) imply that  $(pci_1, a_1) \in R P c$ . But Condition 4.4.3 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

$$(uc_1, a_1, u_1) \in RUC \wedge (pci_1, a_1) \in RPC \quad (36)$$

By applying Condition 4.4.4 to actor  $a_1$  and user  $u_1$ , it follows that there must be a related subject type in the model:

$$\exists st \in St : (a_1, u_1, st) \in RA \quad (37)$$

Considering now (33), Axiom 5 implies that

$$\begin{aligned} \exists t \in T : (pcr_1, t) \in RPC \wedge (so_1, t) \in RSo \\ \wedge samehost[pcr_1, so_1] \end{aligned}$$

and, analogously to the actor argument, Definition 4.6 and (35) imply that  $(pcr_1, t_1) \in RPC$ . Since Condition 4.4.3 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

$$\begin{aligned} (pcr_1, t_1) \in RPC \wedge (so_1, t_1) \in RSo \\ \wedge samehost[pcr_1, so_1] \end{aligned} \quad (38)$$

Condition 4.2.5 then asserts that for the *Target*  $t_1$ , there are related service and resource objects at the SR level:

$$\exists sv \in Sv, r \in R : (t_1, sv, r) \in RT \quad (39)$$

Now, by selecting the subject type  $st_1 \in St$  to satisfy (37), and the service  $sv_1 \in Sv$  and target  $t_1 \in T$  from (39) we conclude that

$$(a_1, u_1, st_1) \in RA \wedge (t_1, sv_1, r_1) \in RT \quad (40)$$

and by applying Condition 4.2.2 to (40) and (35) it comes that there is a related service permission in the model:

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

Therefore, Condition 4.2.3 implies that the actor-target pair of  $atp_1$  is compatible (i.e.  $atcomp[a_1, t_1]$ ). This fact, together with (40), (37), (36), and (38), fulfils all the conditions of Axiom 3. We then conclude that  $sp_1$  is the abstract representation of the  $ac_1$  enabled access:

$$sp_1 = \mathbf{abstract-rep}[ac_1]$$

□

Only the compliance with the security requirements remains yet to be proved, i.e. the second part of the theorem:  $sr[sp_1] \leq \mathbf{acc-psa}[ac_1]$ . For that, Definition 4.15 implies that

$$\mathbf{acc-psa}[ac_1] = \prod_{1 \leq j \leq n} \mathbf{pc-esa}[pc_j, ac_1] \quad (41)$$

where  $pc_1, \dots, pc_n$  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,

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

Substituting this expression in (41) results in

$$\mathbf{acc-psa}[ac_1] = \prod_{1 \leq j \leq n} \mathbf{esa}'[pc_j, aep_1] \quad (42)$$

but according to Definition 8.3,

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

where  $e_1, \dots, e_m$  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 comes that:

$$\begin{aligned} \mathbf{psa}'[aep_1] = \prod_{1 \leq j \leq n} \mathbf{esa}'[pc_j, aep_1] \\ \sqcap \prod_{1 \leq k \leq o} \mathbf{esa}'[c_k, aep_1] \end{aligned} \quad (43)$$

where  $c_1, \dots, c_o$  is the sequence of connectors in the expanded path  $aep_1$ . Thus, since the operator  $\sqcap$  always selects the minimum values for each security class dimension (see Definition 2.5 in Sect. 2.3), by comparing (42) and (43), we conclude that  $\mathbf{psa}'[aep_1]$  is either equal to  $\mathbf{acc-psa}[ac_1]$ , if all connectors have greater values than the minimum levels amongst the processes; or otherwise is less than  $\mathbf{acc-psa}[ac_1]$ , that is:

$$\mathbf{psa}'[aep_1] \leq \mathbf{acc-psa}[ac_1] \quad (44)$$

On the other hand, Lemma 1 implies that  $\mathbf{psa}'[aep_1] = \mathbf{psa}[atp_1]$ . Since Condition 4.2.3 imposes that  $sr[sp_1] \leq \mathbf{psa}[atp_1]$ , from (44) it results that  $sr[sp_1] \leq \mathbf{acc-psa}[ac_1]$ . □

## References

1. Abrams, M., Bailey, D.: Abstraction and refinement of layered security policy. In: Abrams, M., Jajodia, S., Podell, H. (eds.) *Information Security: An Integrated Collection of Essays*, pp. 126–136. IEEE Computer Society Press, Los Alamitos (1994)
2. Barta, Y., Mayer, A.J., Nissim, K., Wool, A.: Firmato: a novel firewall management toolkit. *ACM Trans. Comput. Syst.* **22**(4), 381–420 (2004)
3. Burns, J., Cheng, A., Gurung, P., Rajagopalan, S., Rao, P., Rosenbluth, D., Surendran, A., D.M., Jr.: Automatic management of network security policy. In: *DARPA Information Survivability Conference and Exposition (DISCEX II'01)*, vol. 2 (2001)
4. Common Criteria Project: *Common Criteria for Information Technology Security Evaluation (CC 2.2)*, Part 2: Security functional requirements (2004)
5. Cuppens, F., Cuppens-Boulahia, N., Sans, T., Miège, A.: A formal approach to specify and deploy a network security policy. In: *Formal Aspects in Security and Trust (FAST 2004)* (2004)

6. de Albuquerque, J.P.: Model-based Configuration Management of Security Systems in Complex Network Environments. PhD thesis, Institute of Computing, University of Campinas (2006)
7. de Albuquerque, J.P., Isenberg, H., Krumm, H., de Geus, P.L.: Improving the configuration management of large network security systems. In: Schönwälder, J., Serrat, J. (eds.) *Ambient Networks: 16th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2005, Barcelona, 24–26 Oct 2005*, Proceedings, vol. 3775 of *Lecture Notes in Computer Science*, pp. 36–47. Springer, Berlin (2005)
8. de Albuquerque, J.P., Krumm, H., de Geus, P.L.: On scalability and modularisation in the modelling of security systems. In: di Vimercati, S.D.C., Syverson, P.F., Gollmann, D. (eds.) *Computer Security—ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, 12–14 Sept 2005*, Proceedings, vol. 3679 of *Lecture Notes in Computer Science*, pp. 287–304. Springer, Berlin (2005)
9. de Albuquerque, J.P., Krumm, H., de Geus, P.L.: Policy modelling 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, pp. 24–33. IEEE Computer Society, Washington (2005)
10. de Albuquerque, J.P., Krumm, H., de Geus, P.L.: Model-based management of security services in complex network environments. In: *IEEE/IFIP Network Operations and Management Symposium: Pervasive Management for Ubiquitous Networks and Services, NOMS 2008, 7–11 Apr 2008, Salvador*, pp. 1031–1036 (2008)
11. Ferraiolo, D., Kuhn, R. Role-based access control. In: *Proceedings of 15th NIST-NCSC National Security Computer Conference*, Baltimore (1992)
12. Ferraiolo, D.F., Barkley, J.F., Kuhn, D.R.: A role-based access control model and reference implementation within a corporate intranet. *ACM Trans. Inf. Syst. Secur.* **2**(1), 34–64 (1999)
13. Guttman, J.D.: Filtering postures: local enforcement for global policies. In: *SP '97: Proceedings of the 1997 IEEE Symposium on Security and Privacy*, p. 120. IEEE Computer Society, Washington (1997)
14. Institute of Electrical and Electronics Engineers, New York: *IEEE Standard Glossary of Software Engineering Terminology* (1990)
15. Lück, I.: *Model-based Security Service Configuration*. PhD thesis, University of Dortmund, Germany (2006)
16. Lück, I., Schäfer, C., Krumm, H.: Model-based tool-assistance for packet-filter design. In: Sloman, E.L.M., Lobo, J. (eds.) *Proceedings of IEEE Workshop Policy 2001: Policies for Distributed Systems and Networks*, number 1995 in *Lecture Notes in Computer Science*, pp. 120–136. Springer, Heidelberg (2001)
17. Lück, I., Vögel, S., Krumm, H.: Model-based configuration of VPNs. In: Stadler, R., Ulema, M. (eds.) *Proceedings of 8th IEEE/IFIP Network Operations and Management Symposium NOMS 2002*, pp. 589–602. IEEE, Florence (2002)
18. Moffett, J.D., Sloman, M.S.: Policy hierarchies for distributed system management. *IEEE JSAC Special Issue Netw. Manage.* **11**(9), 11 (1993)
19. Mont, M., Baldwin, A., Goh, C.: POWER prototype: Towards integrated policy-based management. In: Hong, J., Weihmayer, R. (eds.) *Proceedings of IEEE/IFIP Network Operations and Management Symposium (NOMS2000)*, pp. 789–802, Hawaii (2000)
20. Sandhu, R.S.: Lattice-based access control models. *IEEE Comput.* **26**(11), 9–19 (1993)
21. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-based access control models. *IEEE Comput.* **29**(2), 38–47 (1996)
22. Sandhu, R.S., Samarati, P.: Access control: principles and practice. *IEEE Commun.* **32**(9) (1994)
23. Sloman, M.: Policy driven management for distributed systems. *J. Netw. Syst. Manage.* **2**(4), 333–360 (1994)
24. Sloman, M., Lupu, E.C.: Security and management policy specification. *IEEE Netw. Special Issue Policy-Based Netw.* **16**(2), 10–19 (2002)
25. Westerinen, A., Schnizlein, J., Strassner, J., Scherling, M., Quinn, B., Herzog, S., Huynh, A., Carlson, M., Perry, J., Waldbusser, S.: *Terminology for Policy-Based Management*. Internet Engineering Task Force, (2001) RFC 3198
26. Wies, R.: Using a classification of management policies for policy specification and policy transformation. In: Sethi, A.S., Raynaud, Y., Fure-Vincent, F. (eds.) *Integrated Network Management IV*, vol. 4, pp. 44–56. Chapman & Hall, Santa Barbara (1995)