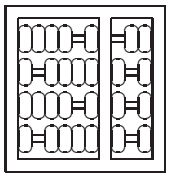

INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Scalable Model-based Policy Refinement and Validation for Network Security Systems

João Porto de Albuquerque

Technical Report - IC-06-005 - Relatório Técnico

March - 2006 - Março

Contents

Scalable Model-based Policy Refinement and Validation for Network Security Systems

João Porto de Albuquerque*
Institute of Computing – State University of Campinas
Caixa Postal 6176 13083-970 Campinas/SP Brazil
jporto@ic.unicamp.br

Abstract

This report builds upon previous work on Model-based Management, and particularly on the Diagram of Abstract Subsystems (DAS) approach, further elaborating on the correctness and performance of the automated policy refinement process. The modelling technique and the automated policy refinement process are firstly presented to illustrate the practical use of the DAS approach. Subsequently, the graphical model is formalised using an algebraic notation, which is thus utilised to define validation conditions for a layered model, i.e. conditions to which a resulting model must comply if the lower-level policy sets have been correctly generated. The scalability of the refinement algorithms and the additional effort needed to validate model instances are also analysed and discussed.

1 Introduction

Current enterprises heavily rely upon network infra-structures 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 [3, 27], research still remains to be done to offering proper abstraction, integration and tool support for the management of security mechanism configuration. 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 for the security of those environments, since a single maladjustment between two mechanisms can leave the whole system vulnerable to attacks.

The Model-based Management (MBM) [13, 12, 14] employs an object-oriented layered model that aims at providing a smooth transition from an abstract view of the system to

*This work was done while the author was in the University of Dortmund (Germany) funded by a scholarship of the German Academic Interchange Service (DAAD).

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 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, culminating with the derivation of configuration parameters for all the security mechanisms of the system.

This approach was already successfully applied to the management of many different mechanism types, such as packet filters [12], Virtual Private Networks [14], Kerberos systems [20], and the integrated management of a number of network security mechanisms [8]. Lück [15] presents a comprehensive overview of the previous works and offers a formalisation of the refinement process.

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 [18, 19, 17]. A subject that was not sufficiently explored up to the moment is the correctness of the automated policy refinement using DAS, which was briefly thematised in [19]. 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-level levels can be expected to be in conformance with the abstract policies specified. Moreover, since the DAS level was introduced to improve the scalability of MBM, one would suppose that the refinement process also scales better than the approach without DAS. The present report is thus dedicated to examining these issues.

To pursuit this goal, the modelling technique and the automated policy refinement process are firstly presented and illustrated by simple model examples, in order to give the reader an understanding of the practical use of the DAS approach. Thereafter, a validation approach starts with the formalisation of the graphical model using an algebraic notation, which is thus utilised to define consistency conditions for a layered model to be valid. These consistency conditions concern both policies and system objects, and are expressed in terms of relations among the various model objects and classes. In order to reason about the effect of the application of the generated configuration to the real environment, a series of axioms are defined to capture the assumptions that are implicit in the modelling. Subsequently, theorems are presented to prove that the defined conditions are sufficient and necessary to guarantee the validity of the refinement, and the soundness of the validation approach is discussed. The scalability of the refinement algorithms and the additional effort needed to validate model instances are also analysed.

1.1 Report Organisation

The report is organised as follows: Sect. 2 presents the main elements of the modelling technique employed; and analyses a simple application example. Sect. 3 describes the supporting tool and the automated policy refinement process, from a modeller's point of view. Subsequently, Sect. 4 elaborates on the formalisation and validation of the generated

policy hierarchy. In Sect. 5, the refinement and the validation results are analysed in terms of the scalability improvements achieved. Finally, Sect. 6 summarises the contributions of this report.

2 Modelling Technique

This section describes the modelling technique of this work by presenting the classes of the meta-model used and their relationship; i.e. by defining the underlying structure behind each model instance. Figure 1 depicts the three-layered meta-model. Each of the levels is a refinement of the superior one in the sense of a "policy hierarchy" [16, 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.

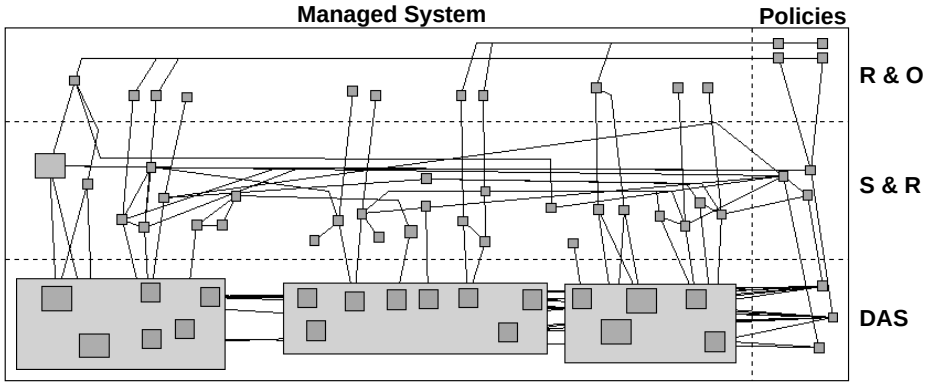

Figure 1: Model Overview

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 [12, 14] and extended. The RO level is based on concepts from Role-Based Access Control (RBAC) [6, 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 described in Sect. 2.1.

The third model layer (DAS) is the main concern of this paper. It aims at offering a modular description of the system's *overall structure* thereby improving the scalability of the modelling technique. DAS is further discussed in Sect. 2.2.

2.1 Layers RO and SR

The topmost level of our model describes the system relying on the concept of roles; i.e. system permissions are established based on functional roles in the enterprise, and then

appropriately assigned to users [6]. The main classes in this level are depicted on the top of Fig. 2 and represent: *Roles* in which people who are working in the modelled environment act; *Objects* that should be subject to access control; and *AccessModes*; i.e. the ways of accessing objects. The class *AccessPermission* expresses an authorisation policy, allowing the performer of a *Role* to access a particular *Object* in the way defined by *AccessMode*.

These classes 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 [7]). 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; its set of classes is shown at the bottom of Fig. 2. Objects of these classes represent: (a) people working in the modelled environment (*User*); (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 suffix "type", 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, but they rather stand for the basic types of these subjects that may occur in run-time (see also [15]).

2.2 Diagram of Abstract Subsystems

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*.

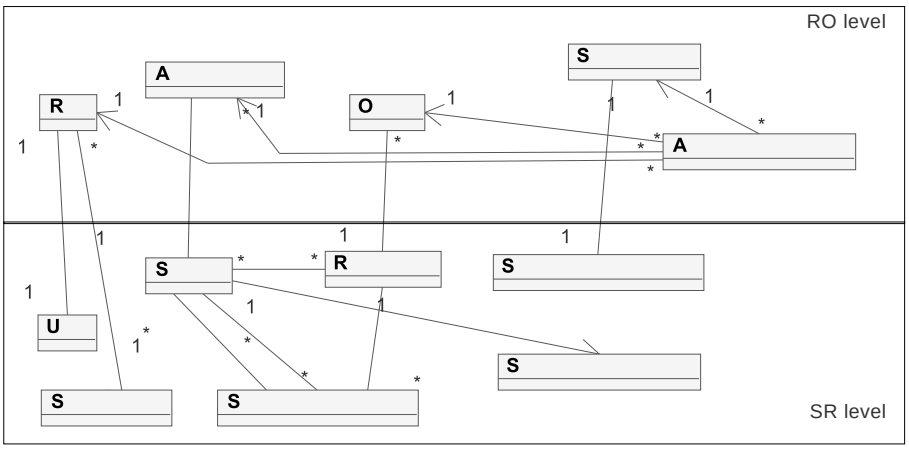

Figure 2: Classes of the RO and SR levels

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 [18]).

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 additional type of security policies: *abstract security goals*[1]. The goals are defined at the RO level by *SecurityGoal* objects (Fig. 2). They extend the RBAC model by abstractly representing the security properties that are required to access an object or to perform a given access mode. The modeller thus defines a *SecurityGoal* by attaching it a label (e.g. "Top Confidential" or "Mission Critic, 4x7 availability needed"), and connecting *Objects* and *AccessModes* to the goal. In this manner, security goals complement the authorisation policies expressed by *AccessPermissions*

[1]The concepts of security goals, requirements and assumptions elaborated in this section have evolved from the *security vectors* proposed in [20].

– which represent *what* must be allowed – with qualitative information about *how* accesses must be performed (following thus the definition of policy goals in [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 [9]. Thus, a *SecurityRequirement* details the security properties that must be fulfilled to achieve the corresponding *SecurityGoal*. These properties are expressed by a vector of *security levels* (natural numbers ranging from 1 to 4) with respect to four categories: *confidentiality*, *integrity*, *availability*, and *accountability*. These categories comprise a representative subset of the most common functional security requirements as stated in standards like [4, 10] and are not intended to be exhaustive. Each 4-tuple of security level values constitutes a *security class* (for instance, $(1, 1, 1, 1)$ is the lowest possible class). Hence, a *SecurityRequirement* specifies how a *SecurityGoal* should be accomplished by the system in terms of the lowest security class that must be enforced for the related objects and access modes.

On the other hand, a model includes an additional type of objects in order to express the security properties that an entity is assumed to *assure*: the *SecurityAssumptions* (see Fig. 2). Similarly to the *SecurityRequirements*, *SecurityAssumptions* are also represented by a security class. At the SR level, *SecurityAssumptions* are associated to each *Service* in order to represent the security class that the service provides; i.e. the security levels one can assume in a communication that involves that service.

As for the DAS level, *SecurityAssumptions* are assigned to each AS in order to express the security class that the elements inside an AS are assumed to share. This makes possible, for instance, to model that one should expect higher confidentiality level of an internal network than of a public network. Additionally, a modeller may assign a *SecurityAssumption* to individual *Actors*, *Targets*, or *Mediators*. The particular object is then assumed to have different security properties than that of the other elements in the subsystem. Thus, as long as a given DAS object does not have a directly associated *SecurityAssumption*, it is the assumed to warrant the security class of the corresponding AS.

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:

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

These basic classes are used to define both the components of the system itself and the security mechanisms employed to control the activity of the former. For the latter, special process classes are also defined to handle each specific mechanism type supported.

Notice that the ES representation has a static character; i.e. it is a picture of the system components and connections, but it does not include a behavioural description of their interactions. Therefore, the modelling of processes is somewhat particular. A process object in the ES level actually stands for a *prototype* of processes that might occur in the real environment. One should thus see a process object not as a picture of one given 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.

In order to make these concepts clear, the next section presents the modelling of a concrete network environment. This environment will be also used as a basis for illustrating the examples throughout the subsequent sections of this paper.

2.5 Application Scenario

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

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

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

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

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

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

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

The three-layered model for this environment is shown in Fig. 3. We describe the main elements of each model layer and their mutual relations in the next section[2].

2.5.1 RO Level

This level contains the following basic objects: the *Roles* "Employee" and "Anonymous Internet User", and the *Objects* "Internal e-mail", "Website", "Internet e-mail" and "Internet

[2]For a detailed elaboration on the modelling process we refer to [18, 17].

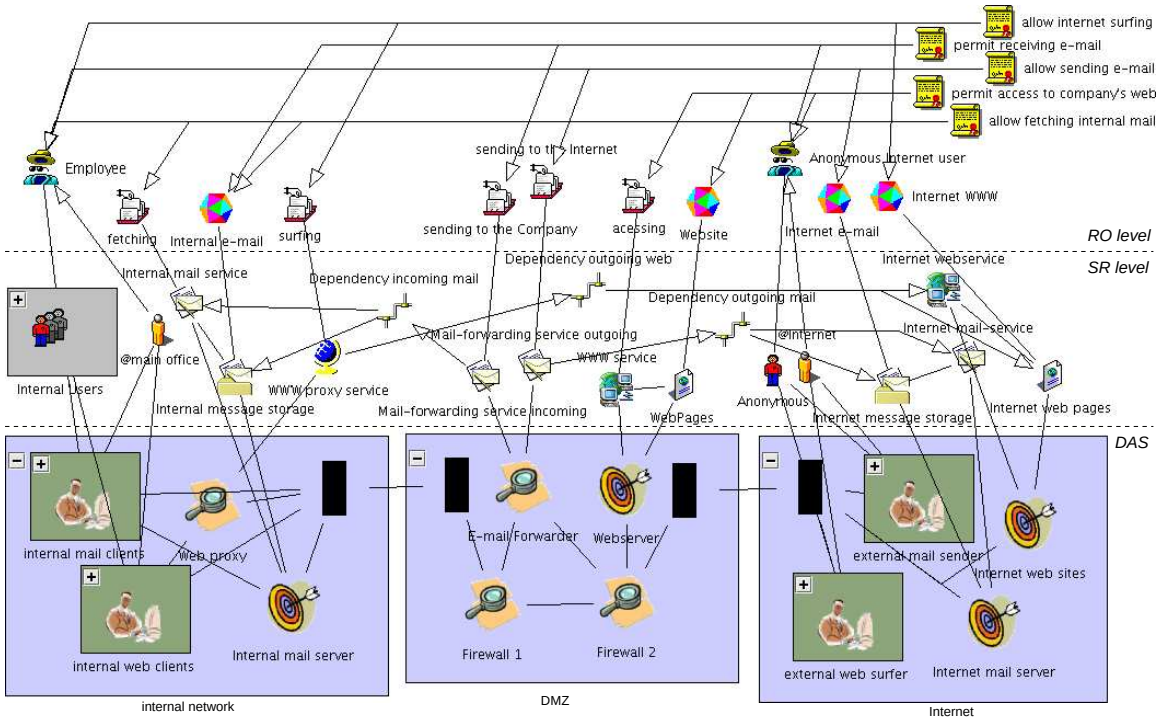

Figure 3: Model Example

WWW". These objects are associated with *AccesModes* by means of five *AccessPermissions* (at the top, on the right of Fig. 3), each of the latter corresponding to one of the abstract policy statements. Thus, for instance, the *AccessPermission* "allow Internet surfing" models the policy statement **P1**, associating the role "Employee" to "surfing" and "Internet WWW". The other policy statements are analogously modelled by the remaining *AccessPermissions*.

2.5.2 SR Level

As for the modelling of users, the *User* "Anonymous" and the *SubjectType* "@Internet" are defined in association with the role "Anonymous Internet User". For the role "Employee", several *User* objects are grouped in the *TypedFolder*[3] "Internal Users", and the corresponding *SubjectType* is "@main office".

Regarding the modelling of services and resources, in turn, the *AccessMode* "sending to the company" (RO level) is refined by the *Service* "Mail-forwarding service incoming e-mail". This service must rely on the "Internal mail service" in order to provide access to the resource "Internal message store" (which is associated with the RO *Object* "Internal e-mail"). This fact is then represented by the connection of the two services to "Dependency incoming e-mail". Similar situations occur with the services "Mail-forwarding service outgo-

[3]Typed folders are, as the name suggests, entities that simply group a number of elements of the same type, in order to offer a more compact representation. Further details can be found in [8].

ing" and "WWW Proxy service", which depend correspondingly on "Internet mail-service" and "Internet web service". On the other hand, a quite simple situation occurs with the *AccessMode* "accessing" and the *Object* "Website" (RO level); in these cases the service "WWW Service" and the resource "Web pages", which are correspondingly associated to those objects, are directly connected to each other.

2.5.3 DAS

Three ASs are defined to represent the structural blocks that compose our scenario: "internal network", "dmz" and "Internet" (bottom of Fig. 3). In the AS "internal network", the *Actors* "internal mail clients" and "internal web clients" are created to map the processes of this subsystem with active behaviour (they are the agents of the policies **P1**, **P2** and **P3**). They are also both connected to the objects in the SR level that map the same behaviour: the *TypedFolder* "Internal Users" and the *SubjectType* "@main office". Similarly, the *Actors* "external mail sender" and "external web surfer" are defined in the AS "Internet" and related to the corresponding objects in the SR level (they address the policies **P5** and **P4**, respectively).

Due to their intermediary or supporting functions, the *Mediators* "Web proxy" ("internal network"), "E-mail Forwarder" ("dmz") are created to map the corresponding services in the SR level. The mediators representing the two firewalls in "dmz", in contrast, are not related to any service in the SR level, since they stand for more technical elements that are not present at the service-oriented abstraction layer of the model.

In the AS "internal network", the *Target* "Internal mail server" is then used to represent the passive character of its related objects "Mail Server" and "Internal Mail Files", also associating these to the service "Internal mail service" and the resource "Internal message store" in the SR level. This pattern also applies to the other targets of the model: "Webserver" (AS "dmz"), "Internet mail server" and "Internet web server" (AS "Internet"). As for the Connectors, they are inserted in order to represent communication possibilities between ASs and thus correspond to the physical interfaces of the actual system.

2.5.4 Expanded Subsystems (ES)

Figure 4 shows the ESs corresponding to the ASs "internal network" and "dmz" (right). The relation from actors, mediators and targets in the abstract representation AS (at the top) to objects in the detailed view ES (bottom) is graphically indicated by edges. For instance, the *Actors* "internal mail clients" and "internal web clients" in the "internal network" (on the left-hand side) are related to objects to represent the corresponding processes that run in two different workstations, as well as to the user credentials and login names of the users that may take advantage of these processes.

In the AS "DMZ" (on the right-hand side), the *Mediator* "E-mail Forwarder" and the *Target* "Webserver" are correspondingly mapped to processes and resources that implement them. These objects in the ES that are directly assigned to AS entities are, in turn, related to a series of other objects in order to provide the model with detailed information about the communication – such as protocol stacks and the network interfaces (see bottom of

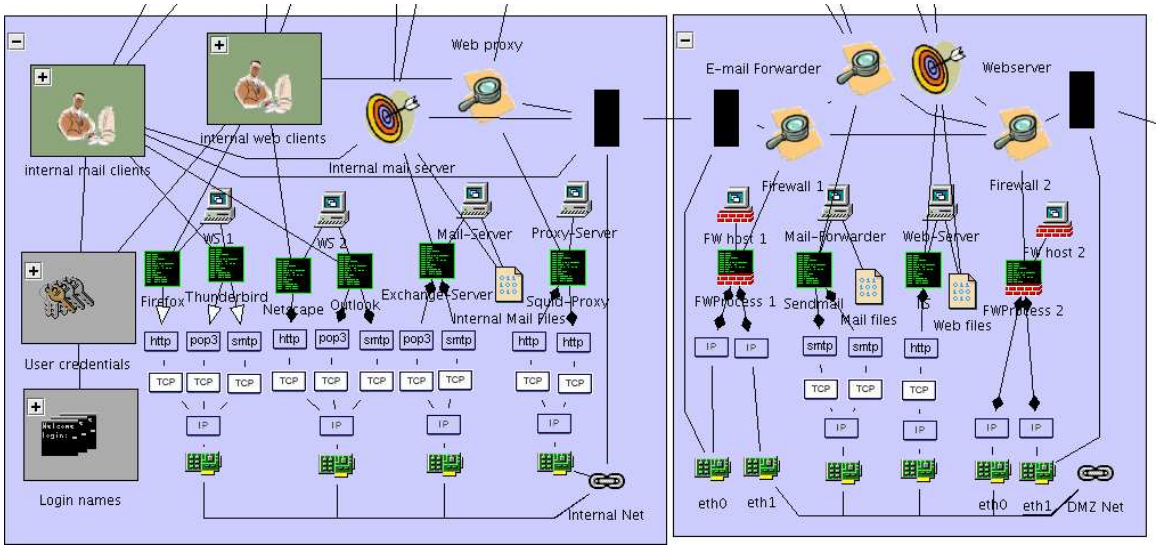

Figure 4: Example of Expanded Subsystems

Fig. 4). In this manner, the correspondence between the abstract view of the system (AS) and its actual mechanisms is established.

3 Tool Support and Automated Refinement

In the Model-Based Management approach, a software tool firstly assists the modelling by means of a diagram editor, and thereafter it automatically derives lower-level policies based on the model entities defined. The next sections elaborate on the practical use of MBM in the particular context of this work by presenting in turn the tool support offered to the modeller and the automated process of policy hierarchy building.

3.1 Supporting Tool

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

as long as there is a meta-model that defines the structure for models in that particular context.

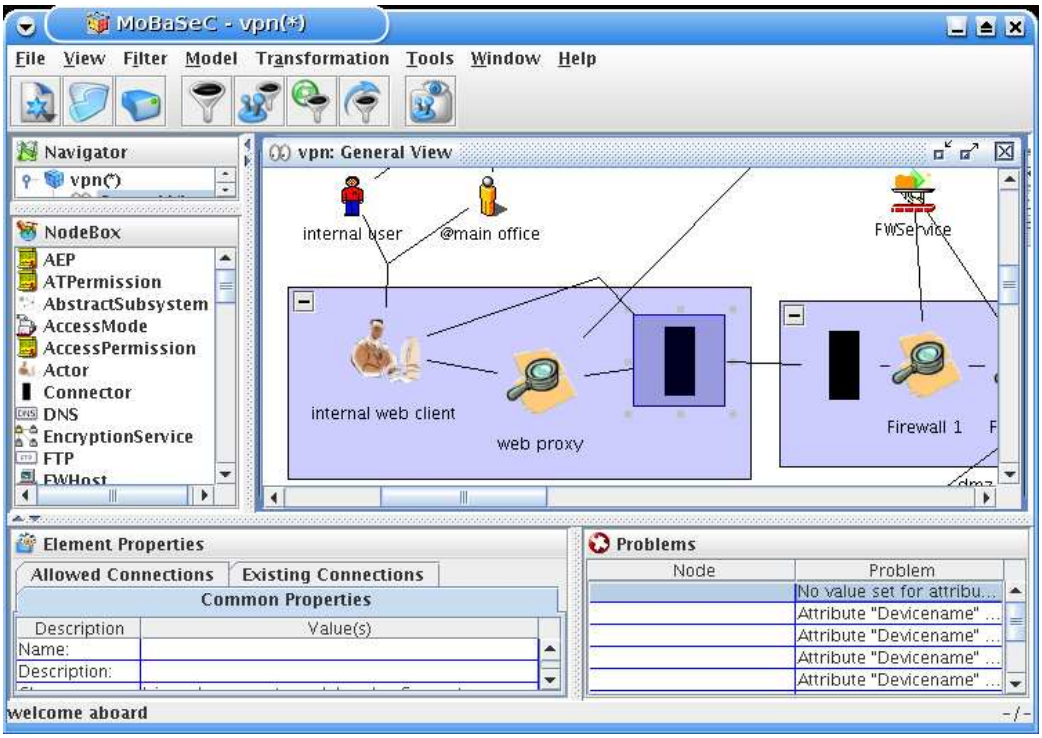

Figure 5: Graphical interface of the supporting tool

MoBaSec is implemented in Java and basically consists of an object-oriented graph editor whose interface is shown in Fig. 5 (for a comprehensive explanation of the tool see [15]). To define a model instance, the user simply selects one of the classes available in the meta-model that are shown in the panel "Nodebox" (at the centre in the left hand side), and create a new object of this class in a window that contains a view of the model instance (in the right hand side at the centre). Connections between two objects are thus established by dragging-and-dropping edges to connect the objects. Furthermore, objects have properties that can be set in a special panel (at the bottom in the left hand side). The property values provide both general information about the objects (e.g. names and descriptions) and, for some classes, special information that will be used in the configuration files generation, such as IP-addresses and port numbers.

MoBaSeC also relies upon the *Model-View-Controller* architectural pattern [2], according to which the model should be separated from its graphical representation. Consequently, the user might define different model views that, for instance, can filter some of the objects, so he/she is able to better visualise given parts of the model. In order to enhance the handling of large models, the tool also incorporates the *focus and context* techniques *semantic zooming* and *fisheye views*. They enable the designer to define in detail a certain model part without losing sight of the system as a whole—due to size restrictions we will

not go into further details about these techniques here; the interested reader may find them in [17].

Furthermore, the tool checks the consistency rules defined in the meta-model in order to ensure that the model instance is valid. The verification of basic rules is performed on-the-fly as the user inputs the model objects and properties—e.g. whether a connection between two objects is allowed. Once the model is complete, a menu item can be used to trigger a more comprehensive model checking, in which a number of general consistency rules are applied, and the problems found are listed in a separated panel (in the right hand side at the bottom of Fig. 5). In this manner, the user receives immediate feedback and can correct the problems before the policy refinement takes place.

3.2 Automated Policy Hierarchy Building

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 *policy refinement* [16, 1] or *policy transformation* [26, 25]. We adopt in this work the first of them.

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 our modelling is structured in different abstraction levels, the analysis of the system's objects, relationships and policies at a certain abstraction level enables the generation of lower level policies, based also on the system's model at the lower level and on the relations between entities of the two layers. As such, the model entities of a certain level and their relationships supply the contextual information needed to automatically interpret and refine the policies of the same level.

An overview of the hierarchical structure of the policies supported in our modelling is presented in Fig. 6. 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 established automatically 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 Fig. 6, 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 special 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

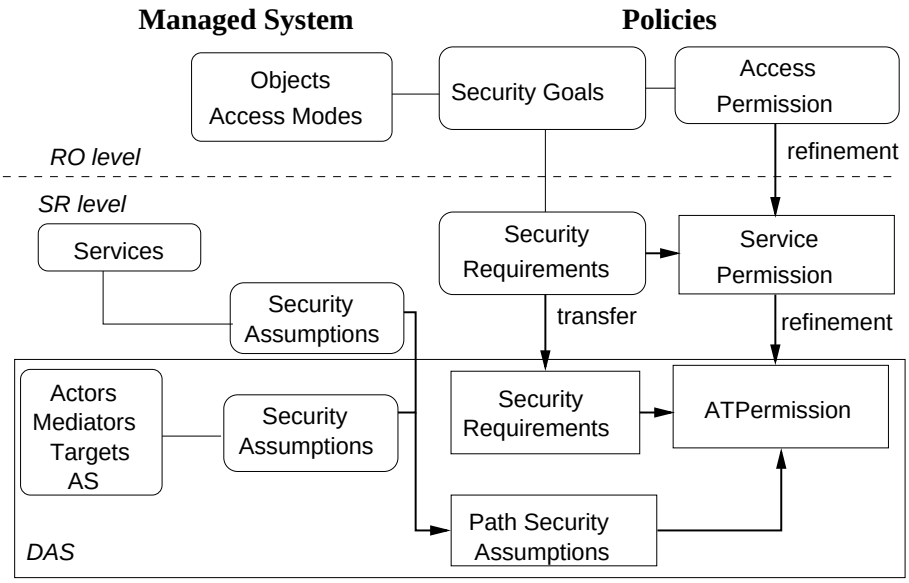

Figure 6: Policies and Security Requirements in the System

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 lower-level layers *must* implement. As such, the positive and negative *authorisation policies* for users — i.e. the *user privileges*, or things that users *may* do or not — are also to be interpreted as *obligation policies* for the system — i.e. the *system duties* or things that the system *must* support (allow) or not (forbid). All of these different connotations of policies in the highest level must be propagated to the inferior levels by the policy refinement process.

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

3.2.1 Refinement RO/SR

The automated refinement of authorisation policies starts from the analysis of the *AccessPermissions* in the RO level and their related objects in order to generate a set of corresponding permissions in the SR level. Thus, each triple of *Role*, *AccessMode* and *Object* (r, am, o) related to an *AccessPermission* produces a set of 4-tuples (u, st, sv, r), each of which expresses an authorisation for a *SubjectType* on behalf of a *User* to use a *Service* in order to access a *Resource*. These tuples are represented by *ServicePermission* objects (see Fig. 6).

A refinement example is shown in Fig. 7. The *AccessPermission* "allow internet surfing" is refined into the *ServicePermission* "sp1" that associates the objects "internal user", "@main office", "WWW proxyservice", and "internet webpages". Each of these elements refine the RO-level objects that are related to the permission "allow internet surfing",

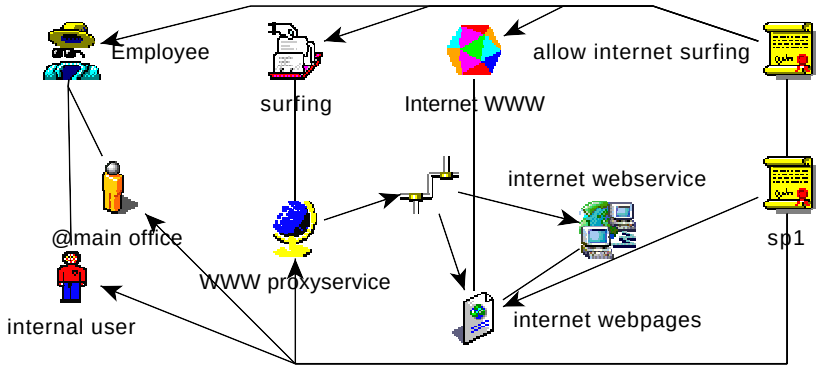

Figure 7: Example of Refinement $RO \rightarrow SR$

namely: "Employee", "surfing", and "Internet WWW", respectively.

In addition, a security requirement is also calculated for each generated *ServicePermission*. This is accomplished by comparing the security classes assigned to the *SecurityGoals* of *Object* and *AccessMode* that participate in the *AccessPermission* being refined (as Fig. 6 illustrates). The *SecurityRequirement* results from choosing the maximum level for each vector dimension between the values of the two security classes. For instance, if an *Object* has a security goal assigned to the class $(3, 3, 2, 1)$, and the *AccessMode* has a security goal assigned to $(3, 1, 4, 2)$, the resulting *SecurityRequirement* has the class $(3, 3, 4, 2)$.

3.2.2 Refinement SR/DAS

Subsequently, the *ServicePermissions* are refined into *ATPermission* objects (actor-target permissions, ATP for short), which represent authorisation policies in a DAS (Sect. 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_t, res). If the service sv has direct access to the resource r, then sv_t—i.e. the service related to the *Target*—and sv will be the same. Otherwise, sv must rely upon a dependency chain that leads to sv_t, in order to access r. In this case, to each service that takes part the dependency chain, a corresponding mediator must be found along the path between *Actor* and *Target*.

A visualisable example for this refinement step is given in Fig. 8 (it is in fact a continuation of the refinement in the previous section). For the *ServicePermission* "sp1' " (top, at the right hand side) the refinement algorithm determines the ATP "atp1" that starts in the *Actor* "internal web clients" (this is indicated by the gray, dotted line from the ATP to the *Actor*) that is connect to the "internal user" and "@main office". The path traverses the *Mediators* "web proxy", "Firewall 1" and "Firewall 2" (as well as some connectors along the way), and ends in the *Target* "internet web sites" (indicated by the dotted line from the latter to the ATP) that is connected to the resource "internet webpages". Notice that the mediator "web proxy" that refines the "WWW proxy service" must be in the path, since there is a dependency chain for this service to reach the resource "internet web pages"; otherwise, "atp1" could use the shortest path that goes from "internal web clients" through

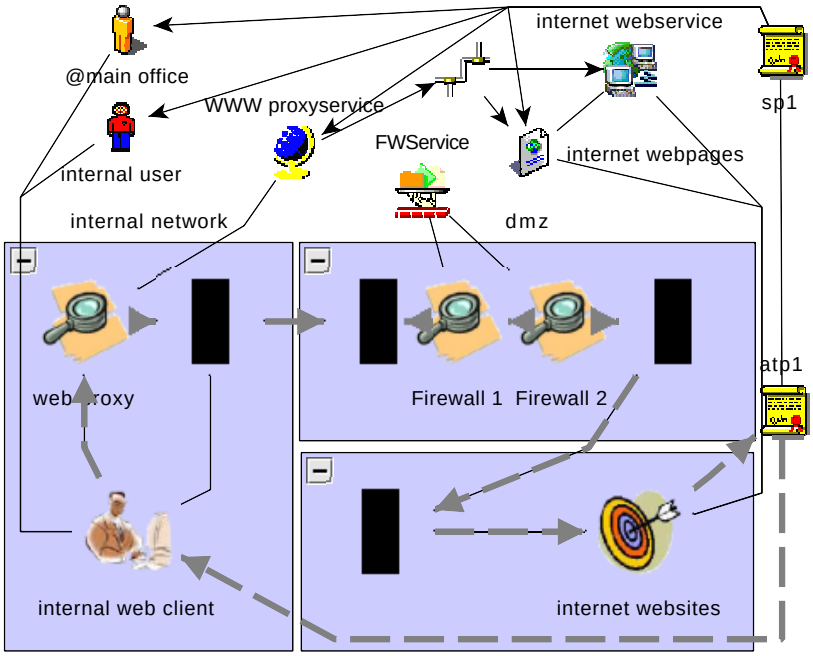

Figure 8: Example of Refinement $SR \rightarrow DAS$

the connectors directly to "Firewall 1".

During the path discovery, the refinement algorithm also checks the security assumption of the path against its security requirement, which in turn comes from the requirement that was determined for the *ServicePermission* in the preceding step (as illustrated in Fig. 6). The *path security assumption* reflects security levels that can be provided by all the elements along the path, and it is calculated with basis on the assumptions of the individual objects and the services that take part in the communication path, as illustrated in Fig. 6 (this topic is further elaborated later on in Sect. 4.3.1). Only paths that comply with the requirements become ATPs; thus, if the tool cannot find such a valid path to refine a *ServicePermission*, an error message is presented to the user, so she/he can modify the system model to make it congruous to the policies.

3.2.3 Refinement DAS/ES

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

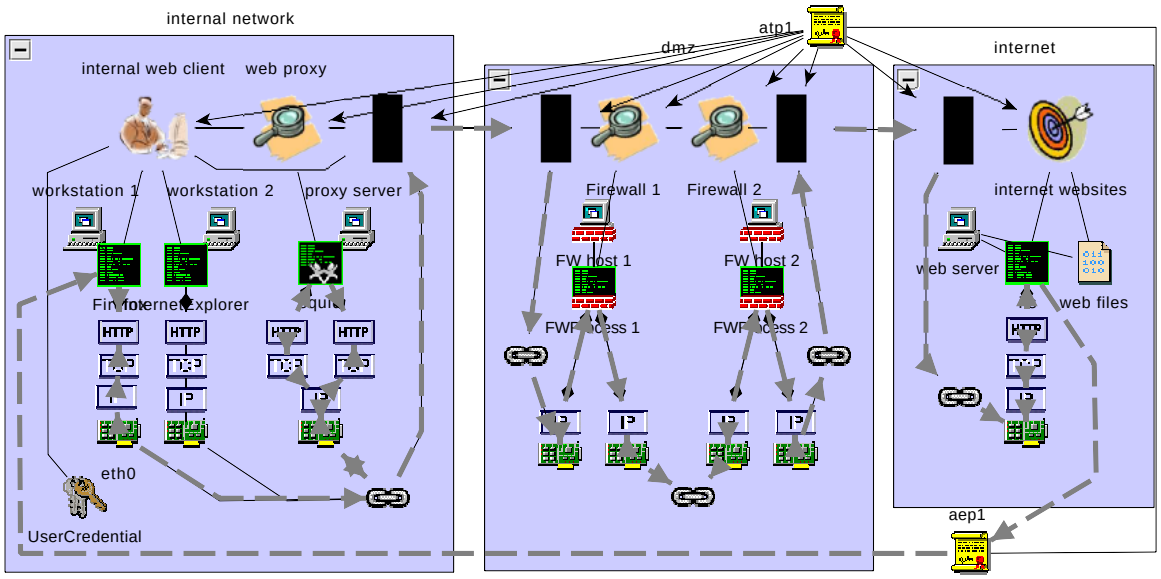

Figure 9: Example of Refinement $DAS \rightarrow ES$

The ATP of the example in the previous section and one of its corresponding AEPs are shown in Fig. 9. The path allowed by "aep1" is marked in the figure by gray, thicker and dashed lines. It starts in the subsystem "internal network" with the process connected to "workstation1"—which is related to the actor "internal web client"—and goes through protocol stacks and network links up to the process in the "proxy server"—which is related to the mediator "web proxy". From this point, the path continues through protocols and links, flows to the subsystem "dmz" traversing connectors and it then passes through two firewalls, reaching the subsystem "internet" and its end point is thus the process related to the "web server"—which is in turn associated with the target "internet websites". For the same ATP, an additional AEP similar to "aep1" is also generated, beginning in the process related to the "workstation2"; it is omitted in Fig. 9.

It is worthwhile noticing that the compliance with security requirements must not be checked in this phase, since it was already verified for the abstract path (ATP) . Indeed, an AEP contains only processes that are related to the abstract entities in the corresponding ATP, whose security properties were already checked. The premise is that only processes are active elements that must be controlled, and passive objects, like protocols and links, must not be considered—a classical assumption in access control models [23] (off course, problems like covert channels are not addressed here).

3.2.4 Configuration Parameter Generation

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

the configuration for a given mechanism must allow only the accesses corresponding to the AEPs that traverses the mechanism.

For instance, considering "aep1" of the example in the previous section (Fig. 9), a back-end function for a specific web proxy software analyses the path and converts it into configuration parameters for the proxy mechanism in the host "proxy server" in order to allow the access. Another specific back-end function will then analyse "aep1" and generate configuration files for the two firewalls in the "dmz" subsystem; i.e. it will produce packet-filter rules that correspond to the characteristics of the path (addresses involved, protocol, ports, connection orientation etc.).

4 Formal Refinement Validation

The consistency among abstraction levels of a policy hierarchy is a crucial issue. Only if the policy sets at the different levels are in perfect harmony, one can trust the system behaviour resulting from the application of the lowest-level policies to be in conformance with the specified abstract goals. Especially, 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.

Note that *completeness* concerns *positive* policies, whilst *consistency* deals with *negative* policies. As such, these two properties are complementary and provide thereby the criteria necessary and sufficient to ensure the propagation of the meanings conveyed by policies and system model in the MBM approach (see Sect. 3.2.1). I shall thus assume that the fulfilment of these two criteria attests the *correctness* or *validity* of the policy refinement.

Therefore, the main goal of the present chapter is to validate the automated refinement described in Section 3; i.e. to prove that the application of the policy refinement to a model instance always complies with the aforementioned validation criteria. Since the refinement correctness between the levels RO and SR was already extensively studied by Lück [15], the topmost abstraction level (RO) is let out of the scope of the analysis here. The validation presented as follows is thus based on an analysis of the policy refinement that starts from the SR level. On the one hand, the analysis considers a complete model after the policy refinement, which is composed of both a group of system objects and a policy set for each of the levels SR, DAS, and ES. On the other hand, another important issue to be analysed is the effect to the real world caused by the implementation of the lowest-level policy set.

The validation approach of this work consists of establishing a series of consistency conditions that the model must fulfil in order to be valid. These consistency conditions

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

Firstly, the next section presents a formalisation of the relevant model entities and relationships. This formalisation will serve as a basis for all subsequent sections. Thereafter, Section 4.2 gives a detailed overview over the validation approach as whole, in order to guide the reader throughout the remaining sections of this chapter.

4.1 Formalism of the Model

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

The possible connections between two or more classes in the meta-model is then represented by a relation on those sets that formalise the classes. Each particular instance in a such relation thus represents one or more edges in the model. As a convention, whenever an element x of a set in a given level is associated to another element y of a set in the immediate superior layer, we say that x is a refinement from y. Consider, for instance, the relation of refinement that exists between a *User* and a *SubjectType*, in the SR level, and an *Actor* in DAS. The sets that formalise these classes are correspondingly U, St and A. Each actor refines a pair user-subject type (see Sect. 2.5.3), so that a relation $RA \subseteq A \times U \times St$ is defined, in order to formalise this fact. For example, suppose that an actor $a \in A$ is associated to the user $u \in U$ and to the subject type $st \in St$ in the model (as in the left hand of Fig. 13). Thus, this association is formalised by the tuple $(a, u, st) \in RA$, and we say that a is a refinement from u, and also that a is a refinement from st, or yet that a is a refinement from (u, st). Notice also that, as a convention, the refinement relations are named beginning with the letter R followed by the refining class; i.e. the class (in the lower abstraction level) that refines the other class or classes.

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

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

Definition 4.1.1. The SR level has the following components:

- U, St, Sv, R, disjoint sets respectively encompassing objects of the classes *User*, *SubjectType*, *Services*, and *Resources*;
- $SvDep \subseteq Sv \times Sv \times R$, a partially ordered relationship to express that a service depends on another to access a resource;
- $SP \subseteq U \times St \times Sv \times R$, a set of *ServicePermission* objects (see Sect. 3).

Along with the set of *positive* authorisation policies SP (Sect. 3.2.2), we also define a set of *negative* authorisation policies $\overline{SP}$, which is complementary to the former one. This second set is due to the semantic of *closed policies* that *ServicePermissions* have in our modelling (see Sect. 3.2.1).

Definition 4.1.2. The set of negative authorisation policies for the SR level is defined as

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

Definition 4.1.3. The DAS level comprises the following elements:

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

Definition 4.1.4. A *local DAS path* is a path in the DAS graph that is completely contained into a single subsystem; i.e. it *spans* one subsystem. The set of local DAS paths is defined as

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

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

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

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

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

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

Definition 4.1.5. The security requirements and assumptions (see Sect. 2.3) are 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);
- $sr : SP \rightarrow SC$, a function that gives the security class required by a *ServicePermission*;
- $sa : Sv \cup Su \cup V \rightarrow SC$, a function that returns the security assumptions of services, subsystems, and elements in DAS.

Definition 4.1.6. Suppose sc_1 and sc_2 are security classes in SC, such that $sc_1 = (l_1, l_2, l_3, l_4)$ and $sc_2 = (m_1, m_2, m_3, m_4)$. Thus the following operations are defined:

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

Definition 4.1.7. The associations between elements of SR and DAS are defined 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 a *Target*;
- $RATP \subseteq ATP \times SP$, refinements from a service permission to an ATPs.

Definition 4.1.8. The ES level has the following elements:

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

Definition 4.1.9. A *local ES path* is a path in the ES graph that is completely contained into a single subsystem; i.e. it *spans* one subsystem. The set of local ES paths is defined as:

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

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

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

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

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

The set AEP of *Allowed Expanded Paths* (Sect. 3.2.4) is a subset of EP that contains the policies in the ES level.

Definition 4.1.10. The associations between elements of the ESs and DAS are defined by the following relations:

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

4.2 Validation Approach Overview

Figure 10 depicts the overview of the condition sets we define in the following sections. Each of these sets is represented by a bold line with arrows pointing to the related model entities; the numbers beside indicate the section in which the set is defined. Thus, we first establish the consistency conditions for a valid refinement from the SR level to the DAS in Section 4.3. These conditions are subdivided into two groups: refinement consistency conditions and structural consistency conditions. The first subgroup is presented in Section 4.3.1 and contains conditions that validate the DAS policy set (ATP) in comparison to the two types of policies at the SR level: service permissions (SP) and security requirements (SR).

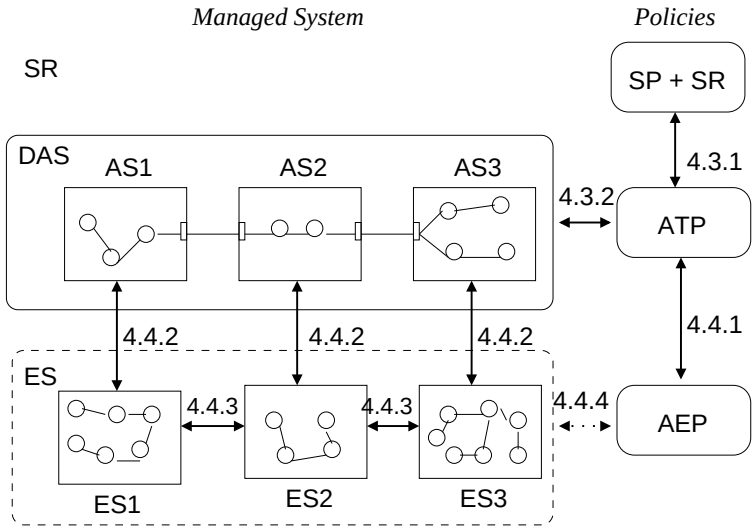

Figure 10: Overview of Validation Condition Sets

The structural conditions (Section 4.3.2), in turn, establish the compatibility of the DAS structure (i.e. the managed system representation) with the ATP policy set.

Subsequently, Section 4.4 analyses the relation between the DAS and the Expanded Subsystems level (ES). Three subgroups of conditions are defined here: refinement consistency conditions, local structural consistency conditions, and composition consistency conditions. The conditions of the first group are presented in Section 4.4.1 and validate the ES policy set (AEP) in comparison to ATP. As at the previous level, the second subgroup (Section 4.4.2) also comprises structural conditions, but this has an important difference: due to the segmented structure of the ES level, these conditions have a *local* scope. They aim at checking whether the abstract view of each subsystem (AS) corresponds to the subsystem elements in the expanded view (ES), and are thus restricted to consider the elements within the boundaries of each subsystem in turn. Section 4.4.3 presents the composition consistency conditions, which validate the interconnection between ESs. Furthermore, we prove in general (by means of the theorems in Section 4.4.4) that the local conditions can be generalised relying on the assertion of the composition conditions. Since these theorems do not have to be checked for each model instance as the other condition sets do, they are represented in Fig. 10 by a dotted line that points both to the AEP set and to the whole ES level.

In order to prove that the conditions are able to validate a model, we examine thereafter the application to a real environment of the configuration generated using the policy refinement process. Section 4.5 presents assumptions about the model capability of representing the real world environment, so called *model representativeness axioms*. Thus, with basis on these axioms and on the consistency conditions, we prove two theorems in Section 4.6:

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

correspond to the policy;

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

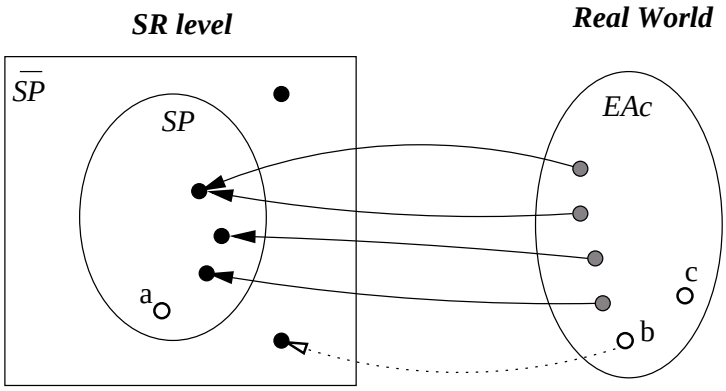

Figure 11: Relation between the SR level and the Real-world

These two theorems establish a relation between the *input* for the refinement process (the policies of the SR level) and its *output* (the possible accesses in the real world). This relation can be represented by the Venn diagram of Figure 11. The input is depicted on the left-hand side by the SP set of service permissions (i.e. the positive authorisation policies), and its complementary set $\overline{SP}$ with the negative authorisation policies (see Sect. 4.1). On the right-hand side, the output is represented by the set of enabled accesses (EAc); i.e. all potential accesses in the real world that are enabled by the whole security system. The arrows connect each of these accesses with its corresponding abstract representation in the sets SP and $\overline{SP}$. In fact, security requirements are also part of the input, but they can be seen in this scheme as a restriction of the relation of abstract representation, such that an access in the real world is only represented by an element of the SR level if the security assumptions of the former comply with the security requirements of the latter.

In Fig. 11, valid elements of each set are represented by black or gray circles (for the input and output, respectively), while white circles stand for invalid elements; i.e. elements that will not be present if the refinement is valid. Indeed, VT1 assures that each element of SP will have all of its corresponding accesses in the real world enabled (i.e. they will pertain to EAc). Therefore, a SP element such as a, which does not have a related element in EAc, will not exist if VT1 holds (considering all SP elements have at least one corresponding access in the real world, see Axiom 2 in Sect. 4.5). Conversely, the existence of elements such as b and c in EAc would violate VT2. The element b represents a possible real-world access that has an abstract representation in the SR level which do not pertain to the policy set SP – thus contradicting VT2. As for c, it stands for an access that does not have an abstract representation at all, and that is just as well forbidden by VT2.

To conclude the validation, Sect. 4.7 makes considerations about the validation soundness and Sect. 5 then analyses the verification complexity of the conditions defined.

4.3 SR/DAS Congruence

4.3.1 Refinement Consistency Conditions

The refinement validation from SR level to DAS must assert the consistency between the two levels in respect to both authorisation policies and security requirements. For this purpose, this first group of conditions concerns the refinement relation $RATP$ (Def. 4.1.10) between the set of authorisation policies at SR (SP) and the homologous set in DAS (ATP). To improve legibility, I will interchangeably use henceforth the terms *service permission*, *SR permission*, and the symbol *sp* to refer to elements in SP; i.e. to policies in the SR level. Analogously, the terms *DAS permission* and *ATPermission*, and the symbol *atp* shall all indicate an element of the DAS policy set ATP.

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

Definition 4.3.1. The predicate $atcomp : A \times T \rightarrow \{0, 1\}$ denotes whether a pair actor-target is compatible; i.e. whether the actor and the target are effectively able to communicate.

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

Definition 4.3.2. The predicate $dep : Sv \times Sv \times R \rightarrow \{0, 1\}$, indicates if a service depends on another to access a resource. It is recursively defined as follows.

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

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

Condition 4.3.1.1. Let $sp = (u, st, sv_1, r)$ be a service permission in SP. For each actor $a \in A$ that refines the pair *user-subject type* (u, st), and each target $t \in T$ that refines a pair *service-resource* like (sv_1, r), given that a and t are compatible, the set ATP must contain a DAS permission *atp* that connects a to t and is related to *sp* through $RATP$. Formally stated:

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

Notice that sv_1 and sv_2 may be different in case of a service dependency, i.e. if sv_1 must rely upon other services to get access to the resource r (see Sect. 3.2.2). If the service sv_1 has direct access to r, then sv_1 and sv_2 will be the same (see Def. 4.3.2).

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

Condition 4.3.1.2. Let $atp = \langle a, \ldots, t \rangle$ be an *ATPermission* in ATP. Suppose there is a user $u \in U$, a subject type $st \in St$, two services $sv_1, sv_2 \in Sv$, and a resource $r \in R$, such that a refines (u, st), and t refines (sv_1, r). Thus, the set SP must contain a service permission $sp = (u, st, sv_2, r)$ that is related to *atp* through $RATP$ and the service sv_1 must have access to r by a dependency chain that passes through sv_2 (sv_1 and sv_2 may also be the same, see Def. 4.3.2). Formally stated:

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

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

The fulfilment of security requirements demands closer examination. In this respect, the first point to be considered is that security assumptions of services that are related to the DAS elements along a given path are assumed to be active throughout the whole path. As such, the security assumptions associated to services are used to model situations in which a certain service, with particularly desirable security properties, is employed to improve the security level of the whole communication path. For instance, a packet filter with activated logging may improve the traceability of all communication flows that pass through it[4].

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

Definition 4.3.3. Let $atp = \langle v_1, \ldots, v_n \rangle$ be an element of ATP and $Sva \subset Sv$ the set of services associated to the elements of *atp*, such that $Sva = \{sv \mid (v_j, sv) \in RM \ \vee \ (v_j, sv, r) \in RT$ for some $1 \leq j \leq n\}$. The function $osa : ATP \rightarrow SC$ is thus defined as:

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

[4]In fact, encryption services that are related to Virtual Private Network mechanisms are not active in the whole path, but rather in the subpath between two VPN gateways. However, since the formalisation 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 combination of the security classes is performed in this definition by a generalisation of the binary operator $\sqcup$ declared in Def. 4.1.6. It selects the greater security level independently for each dimension of the security classes; i.e. for each category of security requirement considered (see also Sect. 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 a *ATPermission*. There are three security assumptions that must be considered: a) the overall security assumption of the *ATPermission*; b) the security assumption associated to the subsystem to which the element pertains (i.e. the security properties expected from the environment wherein the individual is located); and c) the security class assigned to the node itself (represented by its corresponding assumption). These three sources of information about security properties correspond to different protection layers that build upon each other. Although separately modelled, they are all active at the same time in a given component of the real system. Therefore, the security class effectively active in a certain DAS object is the result of the combination of the the security classes from (a), (b), and (c) – just like a chord is the sonorous effect of various tones simultaneously produced.

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

Definition 4.3.4. Let v be a DAS node, such that $atp \in ATP$ contains v. The function $esa : V \times ATP \rightarrow SC$ is thus defined as:

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

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

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

Definition 4.3.5. The *path security assumption* of an *ATPermission* represents the security class that all elements along the path can provide. It is given by the function $psa : ATP \rightarrow SC$. Let $atp = \langle v_1, \ldots, v_n \rangle$ be an element of ATP, thus:

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

This definition relies upon a generalisation of the binary operator $\sqcap$ declared in Def. 4.1.6. It selects the lowest values independently for each dimension of the security classes – thus yielding the desired result for the whole path.

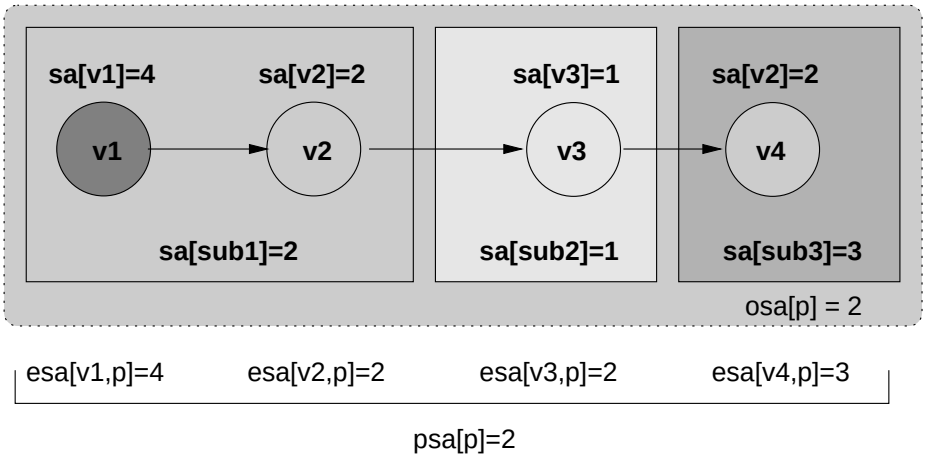

Figure 12: Example of active security assumptions in a path

Figure 12 illustrates the previous concepts with a path example and its associated security assumptions. The path p comprises four nodes (v1–v4) which are distributed over three different subsystems (sub1, sub2 and sub3). The gray tones reflect the security assumption of each element, such that the darker is the background colour of a given object the higher (stronger) is the security level that it can provide (notice that the security classes consist of a vector of four levels, but in this example only one level is considered for the sake of clearness). For instance, the node v1 is assumed to ensure level 4 (sa[v1]=4) even though the subsystem to which it belongs has an assumption of level 2 only (sa[sub1]=2). The effective security level that v1 is assumed to provide in the path is then a combination of the previous two levels with the *overall path assumption* of p – which is of level 2; i.e. osa[p]=2 – resulting in an *effective security assumption* of level 4 (esa[v1,p]=4). On the other hand, the nodes v2 and v3 are the weakest in the path since their effective security assumption amount only to level 2. Consequently, the *path security assumption* of p is also of level 2; i.e. psa[p]=2.

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

Condition 4.3.1.3. Let $sp = (u, st, sv, r)$ be a service permission in SP and $atp \in ATP$ an *ATPermission*, such that $sp = (u, st, sv, r)$, $atp = \langle v_1, \ldots, v_n, \rangle$, and the two are related by $RATP$; i.e. $(atp, sp) \in RATP$. Hence the following propositions must hold:

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

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

(iii) actor a and target t must be compatible (Def. 4.3.1);

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

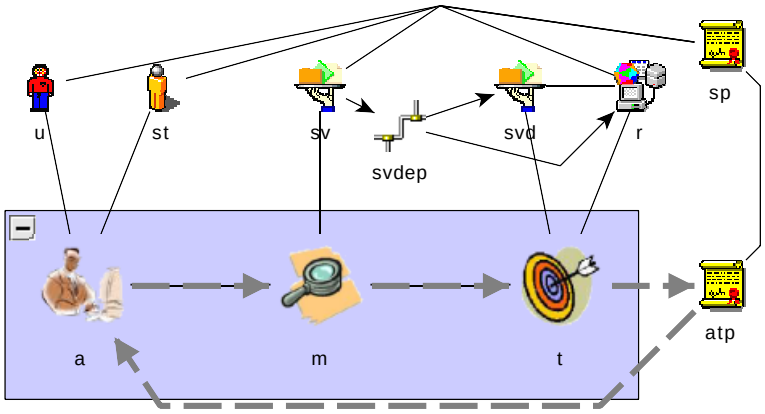

Figure 13: Example of correspondence between a service permission and an ATPermission

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

Formally stated:

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

A generic example of a service permission sp and its corresponding atp is shown in Figure 13. Each of the required items in Cond. 4.3.1.3 can be seen to be satisfied by atp – except the security class compliance which is not depicted by objects, but is represented in their properties.

4.3.2 Structural Consistency Conditions

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

Condition 4.3.2.1. Let $a \in A$ be an actor in the model. Thus, there must exist a user $u \in U$ and a subject type $st \in St$, such that a is associated to the pair (u, st). Formally stated:

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

Condition 4.3.2.2. Let $t \in T$ be a target in the model. Thus, there must exist a service $sv \in Sv$ and a resource $r \in R$, such that t is associated to the pair (sv, r). Formally stated:

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

Condition 4.3.2.3 (DAS Edge Minimality)**.** Let $v_1, v_2 \in V$ be two nodes in DAS, such that there is an edge connecting them. Thus, there must be some permission $atp \in ATP$ that contain the edge (v_1, v_2).

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

Condition 4.3.2.4. Let v_1, v_2 be two vertices in DAS which are not connectors, and $c_1, c_2 \in C$ be two connectors. Suppose there is a DAS path that connects v_1 to v_2 passing through c_1 and c_2. Thus, there must exist a permission $atp \in ATP$ that includes the path $\langle v_1, c_1, c_2, v_2 \rangle$. Formally stated:

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

4.4 DAS/ES Congruence

4.4.1 Refinement Consistency Conditions

Definition 4.4.1. Consider a DAS path $ap \in P$ in an expanded path $ep \in EP$ (see Defs. 4.1.4 and 4.1.9). Let $ap = \langle v_1, \ldots, v_n \rangle$, and let $e_1, \ldots, e_m$ be the ordered sequence of processes and connectors of ep; i.e. the sequence of the elements in ep in the order they occur, such that $e_i \in Pc \cup C$, for $1 < i < m$. The predicate $expand : P \times EP \rightarrow \{0, 1\}$ if an expanded path is a valid expansion of a DAS path, and is defined as follows.

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

Condition 4.4.1.1. Let $atp = \langle a, \ldots, t \rangle$ be an element of ATP. Suppose there are two processes $pc_a, pc_t \in Pc$, such that pc_a refines the actor a and pc_t refines the target t. Thus, an allowed expanded path $aep = \langle pc_a, \ldots, pc_t \rangle$ must exist in AEP, such that aep is a refinement and a valid expansion of atp. Formally stated:

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

Condition 4.4.1.2. Let aep be an allowed expanded path in AEP. Thus, an atp must exist in ATP such that aep refines and is a valid expansion of atp. Formally stated:

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

4.4.2 Local Structural Consistency Conditions

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

Definition 4.4.2. An expanded path in the model (Def. 4.1.9) is said *compatible* if it represents a valid path; i.e. if it connects processes through valid protocol stacks, and connectors. The predicate $compatible : EP \rightarrow \{0, 1\}$ indicates whether an expanded path is compatible.

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

Definition 4.4.3. A local ES path (Def. 4.1.9) is said *locally connected* if it connects two processes and is compatible. This fact is denoted by the predicate $lconn : LEP \rightarrow \{0, 1\}$, formally defined as follows.

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

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

Condition 4.4.2.1. Let v be an actor or target in DAS. Thus, a process $pc \in Pc$ must exist that refines v. Formally stated:

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

Condition 4.4.2.2. Let m be a mediator in M. Thus, a *unique* process $pc \in Pc$ must exist that refines v. Formally stated:

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

Condition 4.4.2.3. Let pc be a process in Pc. Thus, an element in DAS must exist that is refined by pc. Formally stated:

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

Condition 4.4.2.4. Let uc be an user credential in Uc, a be an actor in A and u be a user in U, such that uc refines (a, u). Thus, a subject type $st \in St$ must exist, such that a refines (u, st). Formally stated:

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

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

Condition 4.4.2.5. Let $v_1, v_2 \in V$ be elements of DAS, and let $pc_1, pc2$ be processes in the corresponding ES, such that there is a DAS edge $(v_1, v_2) \in E$, pc_1 refines v_1 and pc_2 refines v_2. Thus, a local ES path must exist that connects pc_1 to pc_2 and is locally connected (Def. 4.4.3). Formally stated:

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

Condition 4.4.2.6. Let $pc_1, pc_2 \in Pc$ be two processes that are connected by a locally-connected ES path p (Def. 4.4.3). Thus, two DAS nodes $v_1, v_2 \in V$ must exist such that pc_1 refines v_1, pc_2 refines v_2, and there is a DAS edge $(v_1, v_2) \in E$. Formally stated:

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

4.4.3 Composition Consistency Conditions

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

Definition 4.4.4. An expanded path (Def. 4.1.9) is considered *connector connected* if it is compatible and connects a processor of a subsystem to another process in an adjacent subsystem (through a pair of connectors) without traversing other processes along the way. Formally stated:

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

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

Condition 4.4.3.1. Let $v_1, v_2 \in V$ be two DAS elements that pertain to adjacent subsystems, such that they are linked by a pair of connectors, i.e. the DAS contains he path $\langle v_1, c_1, c_2, v_2 \rangle$. Suppose that $pc_1, pc_2 \in Pc$ are processes such that pc_1 refines v_1 and pc_2 refines v_2. Thus, a *connector-connected* path must exist that goes from pc_1 to pc_2. Formally stated:

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

Condition 4.4.3.2. Let pc_1 and pc_2 be two processes that are connected by a connector-connected path p. Thus, two DAS nodes $v_1, v_2 \in V$ must exist such that pc_1 refines v_1, pc_2 refines v_2, and the model contains the DAS path $\langle v_1, c_1, c_2, v_2 \rangle$. Formally stated:

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

4.4.4 Generalisation Theorems and Lemma

In order to prove that the local and composition conditions are sufficient to validate the abstract refinement from the DAS to the ES level, we must consider now these layers as a whole, thus achieving a generalised result from the previous considerations (as illustrated in Fig. 10).

Firstly, we define a generalised predicate with basis on Defs. 4.4.3 and 4.4.4.

Definition 4.4.5. A path is considered *connected* if it is either *locally connected* (Def. 4.4.3) or *connector connected* (Def. 4.4.4). This fact is denoted by the predicate $conn : EP \rightarrow \{0, 1\}$, formally defined as: $conn[p] \Leftrightarrow\ lconn[p]\ \vee\ cconn[p]$.

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

Theorem 4.4.4.1 (GT1)**.** *Let dp be an expanded path in the model, such that it is contained in some allowed expanded path (i.e. $dp \subseteq aep \in AEP$) and assume $pc_1, \ldots, pc_m$ is the sequence of processes in dp. In this case, the model contains a connected path between each pair of successive processes in the sequence $pc_1, \ldots, pc_m$. Formally stated:*

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

Theorem 4.4.4.2 (GT2)**.** *Let $pc_a, pc_b \in Pc$ be two processes such that there is a connected path p in the model from pc_a to pc_b. In this case, the model contains an allowed expanded path aep that encloses p; i.e. aep allows the connection from pc_a to pc_b through exactly the same processes along the path p. Formally stated:*

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

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

4.4.4.1 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. 4.3.1).

Definition 4.4.6. Let e be a process or connector that is contained in the path $aep \in AEP$, and let $atp \in ATP$ be the *ATPermission* related to aep; i.e. $(aep, atp) \in RAEP$. The function $esa : Pc \cup C \times AEP \rightarrow SC$ denotes the *effective security assumption* that e can assure in the context of aep and is defined as follows.

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

Notice that Cond. 4.4.2.3 implies that a related DAS element v must exist to each ES process, thus assuring a value for the function esa' in the second case of the definition above.

The security class that can be assured in the context of an AEP is denoted by the function psa' (similarly to the function psa of Def. 4.3.5).

Definition 4.4.7. The *path security assumption* of an AEP represents the security class that all elements along the path can provide. It is given by the function $psa' : AEP \rightarrow SC$. Let aep be an element of AEP, such that $e_1, \ldots, e_m$ is the sequence of elements of aep that pertain to $Pc \cup C$ (i.e. only processes and connectors along the path), thus:

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

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

Lemma 4.4.4.3. *Let $aep \in AEP$ be an allowed expanded path and $atp \in ATP$ an* ATPermission*. If aep is a refinement of atp then both can assure the same security levels; i.e. the path security assumption of aep is equal of that of atp. Formally stated:*

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

Proof. Let us consider without loss of generality an allowed expanded path $aep \in AEP$ and an *ATPermission* $atp \in ATP$, such that $(aep, atp) \in RAEP$. Let $atp = \langle v_1, \ldots, v_n \rangle$ and let $e_1, \ldots, e_m$ be the sequence of processes and connectors of *aep*. According to Def. 4.4.7 thus follows that:

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

Since *aep* is related to *atp*, from Cond. 4.4.1.2 results that $expand[aep, atp]$ must be valid. Then relying upon the definition of *expand* (Def. 4.4.1), we conclude that for each e_j in the equation above, if it is a process then there is an associated v_j in *atp* – i.e. $(e_j, v_j) \in RPc$ – and then the definition of esa' implies $esa'[e_j, aep] = esa[v_j, atp]$. On the other hand, if e_j is a connector, then $e_j = v_j$ and $esa'[e_j, aep] = esa[v_j, atp]$ also holds. Therefore, Eq. (1) can 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 $psa[atp]$ (Def. 4.3.5). □

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

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

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. In this respect, the relevant sets are formalised by the following definition.

Definition 4.5.1. The real environment comprises the following sets:

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

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

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

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

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

It is worthwhile to remember here that a process in the ES level represents a *prototype* for actual processes that might be started in the real environment (Sect. 2.4). As such, the relation established by Axiom 1 between processes and accesses in the real environment imposes that the later will also stand for the *prototype* of actual accesses, or as we put it above, for potential accesses in the system that share some common properties. These common properties are thus each of the 4-tuple dimensions mentioned in Axiom 1.

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

Definition 4.5.2. The correspondence between an access in the real environment and the SR level is denoted by the function $\mathsf{abstract\text{-}rep} : Ac \rightarrow U \times St \times Sv \times R$. The 4-tuple (u, st, sv, r) returned by the function is the abstract representation of a given access in the set Ac.

The reader may be asking his/herself at this point: why is the access mapped to a representation at the SR level? Why not another model level? This choice is not arbitrary; instead, it is due to the fact that the SR level is the topmost level considered in the validation (see the beginning Section 4). As such, the SR representation offer the reference for the expected system behaviour in the context of the present validation.

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

An additional point that worths mentioning here is that the service dependencies of the SR level (see Sect. 4.3.1) are by purpose let out of analysis in this section and in the validation theorems of the next section. This is done in order to simplify the assumptions and proofs, such that one can more clearly perceive their intent and meaning. Nevertheless, these results and assumptions could be simply changed to cover dependencies by the adding of extra conditions to the statements. Therefore, letting dependencies out does not consist of a limitation for the validation presented here.

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

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

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

In addition to that, we must now analyse the the model structure that reflects the relation of abstract representation defined above. In order to yield valid results, the model structure must correctly represent the correspondence between each access in the real environment and its abstract representation at the SR level. For each 4-tuple (uc, pc_i, pc_r, so) that stands for an access in the real environment, the model structure must thus connect the objects in each dimension of that tuple with the objects of the corresponding abstract representation in the SR level; i.e. with the objects of the corresponding 4-tuple (u, st, sv, r). The connection is established through actor and targets as illustrated in Figure 14. Hence, an actor a is connected to both the pair user credential and initiator process (uc, pc_i), and

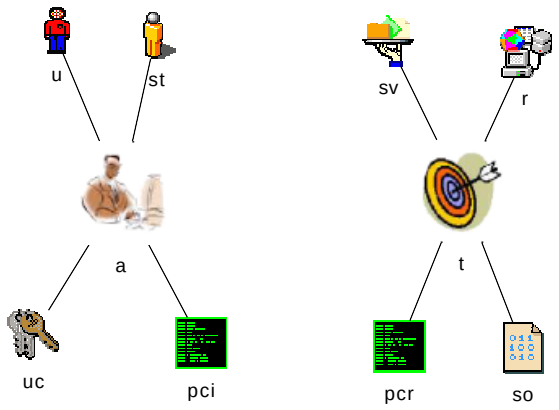

Figure 14: Correspondence between accesses in the real environment and the SR level

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

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

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

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

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

(iii) actor a and target t must be compatible (see Def. 4.3.1);

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

Formally stated:

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

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

Definition 4.5.3. The predicate $samehost : Pc \times So \rightarrow \{0,1\}$ denotes if a process and a system object are connected to the same host. It is defined as:

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

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

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

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

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

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

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

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

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

Note that this axiom requires that the system be able to identify and authenticate correctly the user credential, and that the privileges for starting processes be enforced in conformance with the modelled actors. As in most access control models (as pointed out by Sandhu [23]), this work assumes that identification and authentication of users take place in a secure and correct manner, and the main concern is with what happen afterwards.

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

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

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

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

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

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

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

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

Formally stated:

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

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

Definition 4.5.5. The function $\mathsf{pc\text{-}esa} : Pc \times Eac \rightarrow SC$ gives the effective security assumption of a process in the context of an enabled access in the real environment.

Subsequently, since Axiom 6 implies that for each enabled access (say eac) there is a corresponding allowed expanded path in the model (say aep), each process along the communication path of eac is assumed to enforce the same security class as that assured by the corresponding process object in the context of aep – i.e. the effective security assumption of Def. 4.4.6 in Sect. 4.4.4. This assumption follows from a correct attribution of security classes to objects in the model. The modeller must be consistent by the classification of

model objects, so that the effective security assumption of processes correspond to the security properties of the entities in the real environment. The following axiom formalises this assumption.

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

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

Finally, we must consider the security class one can expect of an access as a whole. Analogous to the functions psa and psa' (Defs. 4.3.5 and 4.4.7), 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 4.5.6. The function $\mathsf{acc\text{-}psa} : Eac \rightarrow SC$ gives the *path security assumption* of an enabled access in the real environment; i.e. the security class that all elements within the scope of an allowed communication flow in the real environment can provide. Let eac be an element of EAc, such that $eac = (uc, pc_1, pc_n, so)$ and $pc_1, \ldots, pc_n$ is the sequence of processes in the communication flow of eac, thus:

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

4.6 Validation Theorems

Based on the axioms of the previous section, the validation theorems proposed in Section 4.2 are formalised and proved in the next sections.

4.6.1 Proof of VT1

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

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

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

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

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

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

$$\exists\, a_1 \in A, t_1 \in T : (a_1, u_1, st_1) \in RA \wedge (t_1, sv_1, r_1) \in RT \wedge atcomp[a_1, t_1] \quad (2)$$

$$\wedge (uc_1, a_1, u_1) \in RUc \wedge (so_1, t_1) \in RSo \wedge samehost[pc, so] \quad (3)$$

$$\wedge\, (pci_1, a_1), (pcr_1, t_1) \in RPc \quad (4)$$

By applying Cond. 4.3.1.1 to Eq. (2) it thus follows that there is an *ATPermission* between the actor a_1 and target t_1:

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

Therefore, Cond. 4.4.1.1 implies that from the expression above follows that:

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

Hence, considering Eq. (4) it results in that the universal quantifier in the previous expression also applies to the processes pci_1 and pcr_1, so that:

$$\exists\, aep_1 \in AEP,\ aep_1 = \langle pci_1, \ldots, pcr_1 \rangle : expand[aep_1, atp_1] \quad (5)$$

Axiom 4 implies that from $(uc_1, a_1, u_1) \in RUc$ in Eq. (3), and $(pci_1, a_1) \in RPc$ in Eq. (4) it thus follows:

$$\mathsf{can\text{-}use}(uc_1, pci_1) \quad (6)$$

Analogously, Axiom 5 implies that from $(so_1, t_1) \in RSo \wedge samehost[pc, so]$ in Eq. (3) and $(pcr_1, t_1) \in RPc$ in Eq. (4) we can conclude:

$$\mathsf{can\text{-}handle}(pcr_1, so_1) \quad (7)$$

By applying Axiom 6 to Eqs. (5), (6), and (7) it thus comes that $ac_1 \in EAc$; i.e. the access is enabled by the system in the real environment. □

4.6.2 Proof of VT2

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

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

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

$$\mathsf{can\text{-}use}[uc_1, pci_1] \tag{8}$$

$$\mathsf{can\text{-}handle}[pcr_1, so_1] \tag{9}$$

$$\exists\ aep_1 \in AEP : aep_1 = \langle pci_1, \ldots, pcr_1 \rangle \tag{10}$$

By applying Cond. 4.4.1.2 to Eq. (10) it follows that a corresponding *ATPermission* must exist in the model:

$$\exists!\ atp_1 \in ATP,\ atp_1 = \langle a_1, \ldots, t_1 \rangle :\ expand[aep_1, atp_1] \tag{11}$$

On the other hand, from Eq. (8) and Axiom 4 it comes that:

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

The definition of the predicate *expand* (Def. 4.4.1) and Eq. (11) imply that $(pci_1, a_1) \in RPc$. But Cond. 4.4.2.3 asserts that there is only one *Actor* associated to each process, so that the expression above can only be true for $a = a_1$ and $u = u_1$. Thus it can be reformulated as follows.

$$(uc_1, a_1, u_1) \in RUc \wedge (pci_1, a_1) \in RPc \tag{12}$$

By applying Cond. 4.4.2.4 to actor a_1 and user u_1 it follows that there must be a related subject type in the model:

$$\exists\ st \in St : (a_1, u_1, st) \in RA \tag{13}$$

Considering now Eq. (9), Axiom 5 implies that:

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

Analogously to the actor argument, Def. 4.4.1 and Eq. (11) imply that $(pcr_1, t_1) \in RPc$. Since Cond. 4.4.2.3 asserts that there is only one *Target* associated to each process, the expression above can only be true for $t = t_1$, so that it can be rewritten as follows.

$$(pcr_1, t_1) \in RPc\ \wedge (so_1, t_1) \in RSo\ \wedge\ samehost[pcr_1, so_1] \tag{14}$$

Cond. 4.3.2.2 then asserts that for the *Target* t_1 there are related service and resource objects at the SR level:

$$\exists\ sv \in Sv, r \in R : (t_1, sv, r) \in RT \tag{15}$$

Now, by selecting the subject type $st_1 \in St$ to satisfy Eq. (13), and the service $sv_1 \in Sv$ and target $t_1 \in T$ from Eq. (15) we conclude that:

$$(a_1, u_1, st_1) \in RA \wedge (t_1, sv_1, r_1) \in RT \tag{16}$$

By applying Cond. 4.3.1.2 to Eqs. (16) and (11) it comes that there is a related service permission in the model:

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

Therefore, Cond. 4.3.1.3 implies that the pair actor-target of atp_1 is compatible (i.e. $atcomp[a_1, t_1]$). This fact together with Eqs. (16), (13), (12), and (14) fulfil all the conditions of Axiom 3. We then conclude that sp_1 is the abstract representation of the enabled access ac_1:

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

Only the compliance with the security requirements remains yet to be proved; i.e. the second part of the theorem: $sr[sp_1] \leq \mathsf{acc\text{-}psa}[ac_1]$. In this respect, Def. 4.5.6 implies that:

$$\mathsf{acc\text{-}psa}[ac_1] = \bigsqcap_{1 \leq j \leq n} \mathsf{pc\text{-}esa}[pc_j, ac_1] \tag{17}$$

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

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

By substituting this expression in Eq. (17) it results in:

$$\mathsf{acc\text{-}psa}[ac_1] = \bigsqcap_{1 \leq j \leq n} esa'[pc_j, aep_1] \tag{18}$$

But according to Def. 4.4.7:

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

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

$$psa'[aep_1] = \bigsqcap_{1 \leq j \leq n} esa'[pc_j, aep_1] \quad \sqcap \quad \bigsqcap_{1 \leq k \leq o} esa'[c_k, aep_1] \tag{19}$$

Where $c_1, \ldots, c_o$ is the sequence of connectors in the expanded path aep_1. Thus, since the operator $\sqcap$ always select the minimum values for each security class dimension (see Def. 4.1.6 in Sect. 4.1), by comparing Eqs. (18) and (19) we conclude that $psa'[aep_1]$ is either equal $\mathsf{acc\text{-}psa}[ac_1]$, if all connectors have greater values than the minimum levels among the processes; or otherwise $psa'[aep_1]$ will be less than $\mathsf{acc\text{-}psa}[ac_1]$, that is:

$$psa'[aep_1] \leq \mathsf{acc\text{-}psa}[ac_1] \tag{20}$$

On the other hand, Lemma 4.4.4.3 implies that $psa'[aep_1] = psa[atp_1]$. Since Cond. 4.3.1.3 imposes that $sr[sp_1] \leq psa[atp_1]$, from Eq. (20) it results in that $sr[sp_1] \leq \mathsf{acc\text{-}psa}[ac_1]$. $\square$

4.7 Validation Soundness

Analysing the two validation theorems demonstrated in the previous section, one can conclude that they ensure the compliance of the policy sets with the validation criteria defined in the beginning of Sect. 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 model. These criteria are cited in the literature as the most important to assure the refinement *correctness* in a multi-layered policy hierarchy (see, for instance, [1] and [24]). Indeed, in the particular context of this work, the criteria ensure that the different meanings conveyed by policies and system model — i.e. both positive and negative prescriptions of system duties and of user privileges (see Sect. 3.2) — be propagated and maintained from one level to its immediate inferior neighbour.

In this manner, the formal proofs mean that: provided a model (including system and policy representation after a process refinement) fulfils the defined condition sets, and this model accurately represents a real-world environment (i.e. the axioms hold), one can conclude that the generated *output* — i.e. the possible system behaviour that results from using the generated configuration — is exactly in conformance with the *input* — i.e. the system view and the policies at the most abstract level considered (SR). As such, the whole system can be seen to act as a reference monitor in the access control terminology [22], which allows only those accesses enabled by the access control policies.

Notice 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, but rather implicitly assumed. The model representativeness axioms of Sect. 4.5 thus serve to making explicit the assumptions that are implicit in the modelling, thereby allowing us to draw conclusions about the system functioning behind the model. In this manner, the axioms reflect assumptions concerning both the *correct modelling* of the real world entities, and the *correct implementation* of the normative model elements.

As a matter of fact, the experience has also shown that, in practice, the modelled network systems are frequently not capable of enforcing the given high-level policies. In this case, the validation conditions defined can not be satisfied, and the tool indicates the model elements which violate the policies. In this manner, the modeller receives thereby valuable information to do the necessary modifications on the system in order to make it congruous to the policies.

As for the security goals and requirements, they should be actually seen as a prescription of *modes* in which accesses may take place. They complement the access control policy adding contextual information, having thus a subordinate character, i.e. in isolation, they do not have a meaning for their own, but they only acquire significance together with an access permission by modifying (restricting) the modelled authorisation. As such, they act just like adverbs in the natural language, which are used to imprint a different connotation to verbs and adjectives[5]. Consequently, the security requirements and assumptions in this

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

work are not to be directly compared with clearances and labels of traditional mandatory policies in lattice-based access control models [23], as the latter have a much stronger, independent character (although there is off course some resemblance between the two worlds).

Furthermore, the requirement support is let flexible enough, so that it is possible to address the needs of different scenarios, depending on how fine-grained and comprehensively one would like to verify the requirement assurance by the system. For instance, a modeller may expand the categories above to encompass all functional classes prescribed by the Common Criteria [4], adjusting the vector dimension and the level range in order to map families and components in each one of that classes. For the purpose of the present work tough, the simplified approach adopted is sufficient.

5 Refinement and Validation Analysis

From the standpoint of the practical application, the consistency conditions defined in Sect. 4 can be sorted into three groups. The first condition group is checked on the fly as the modeller defines the objects and associations in the diagram. Each condition in this group reflects a simple relation between objects in the model and it is checked either: a) whenever a object is created; b) whenever a new association between two objects is defined — i.e. a new edge is drawn in the diagram; c) whenever the modeller launches the menu option "Check Model" in the tool. As such, this first group consists of the *pre-conditions* for the algorithm of the respective refinement step (SR/DAS or DAS/ES). Furthermore, since these conditions are tested together with the model input, the computational cost for their verification is insignificant (in comparison with the time that the modeller needs to draw the model).

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

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

In the following sections, each of the conditions defined in Sect. 4 is assigned to one of the three aforementioned groups, and the verification complexity of the conditions of the third group is analysed in detail, as well as the running time for the refinement algorithms. This is performed in two parts, in such a way that each part corresponds to a refinement step through the abstraction layers. In this manner, the next section presents the analysis of the refinement SR/DAS. Subsequently, the conditions and algorithms concerning the refinement DAS/ES are then examined in Sect. 5.2. Finally, the scalability of the whole checking process is evaluated in Sect. 5.3.

5.1 Analysis of the SR/DAS phase

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

- *Pre-conditions*: structural conditions 4.3.2.1, and 4.3.2.2
- *Post-conditions*: refinement conditions 4.3.1.1, 4.3.1.2, 4.3.1.3.

Thereafter, only Condition 4.3.2.3 must be extra verified after the execution of the refinement algorithm of this phase. The checking consists, for each edge of a DAS, of ensuring that this edge is used in at least one of the ATPs in the set generated policy set. As such, if we make the refinement algorithm mark, during its execution, the DAS edges it uses(with no extra time penalty), the verification effort for this condition will be simply to check for each edge whether it is marked — thus linear to the amount of DAS edges, i.e. $O(|E|)$.

As for the refinement algorithm SR/DAS itself, it consists of a variation of the *modified Dijkstra algorithm* for single-source shortest path discovery [5, Ch. 25], including particularities of the DAS representation such as the fact that a path may traverse mediators but not actors and targets (the refinement algorithms are listed in Appendix A). Since a path discovery must be performed for each compatible pair *Actor-Target*, we can overestimate the running time for the algorithm by considering that a search is executed for all pairs of actors and targets in the model. Thus, the algorithm is $O(|A| \cdot |T| \cdot E \lg V)$, as the cost for each path discovery with the modified Dijkstra algorithm is $O(E \lg V)$ [5, p. 530].

5.2 Analysis of the AS/ES phase

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

- *Pre-conditions*: local structural conditions 4.4.2.1, 4.4.2.2, 4.4.2.3 and 4.4.2.4;
- *Post-conditions*: refinement conditions 4.4.1.1 and 4.4.1.2.

In order to implement the conditions that must be additionally tested — namely, conditions 4.4.2.5, 4.4.2.6, 4.4.3.1, and 4.4.3.2 — let us consider an auxiliary data structure called *process connection matrix* $PCM = (x_{ij})$ that consists of a $n \times n$ matrix in which n is the number of processes and each element x_{ij} represents the fact the process pc_i is connected to the process pc_j in the ES model; i.e. $conn[pc_i, pc_j]$. From the definition of the predicate *conn* (Def. 4.4.5), one can conclude that the matrix will include processes that are both locally connected and those connected through connectors.

A PCM matrix will be then constructed for each expanded subsystem separately. The first step is to take an ES subgraph for a particular subsystem s and generate an auxiliary graph $ES' = (W', F')$ that encloses all elements of s and additionally also the processes in the adjacent subsystems that are linked through connectors to elements of s. As such, the PCM matrix is in fact a subset of the *transitive closure* of ES' that only contains the processes. It can be thus calculated using the Floyd-Warshall algorithm in time $O(|W'|^3)$ [5, Ch. 26].

Using these auxiliary structures, the verification of the validation conditions is quite straightforward. We must just take each pair of processes in PCM and, if they are connected there must be a corresponding edge in DAS between the related objects (Conds. 4.4.2.6 and 4.4.3.2); otherwise, the edge must not exist — thus satisfying Conds. 4.4.2.5, 4.4.3.1, as all involved elements in DAS must be connect to at least one process from Conds. 4.4.2.1 and 4.4.2.2. Therefore, the cost for this checking is $O(n^2)$, since n^2 lookups for edges in DAS will be performed (which we can consider to be executed in constant time).

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

$$\sum \left|W_i'\right|^3, \tag{21}$$

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

Let us analyse now the refinement algorithm of this phase. It consists of, for each edge of the ATPs like (v_1, v_2), finding a local path between the ES processes that refine v_1 and v_2. But we can use the same Floyd-Warshall algorithm of the validation above to calculate at the same time a matrix with the local shortest-paths between each pair of processes in a subsystem, with the same asymptotic behaviour as before (in fact, just with a very tight additional constant time, see [5, Ch. 26]). Hence, the path determination is performed with constant time in the refinement phase (just looking at the matrix generated by the validation), so the DAS/ES refinement running time is $O(E)$.

5.3 Scalability Improvements

Comparing the two refinement phases previously analysed, we observe that the SR/DAS phase has a global character with respect to subsystems — i.e. paths that span the whole DAS must be found, and the extra condition is related to all edges in the graph — whilst the DAS/ES step is performed locally — i.e. only local paths and local connections are considered, except from the immediate vicinity of the connectors. This asymmetry is due to the fact that, for the DAS/ES step, the theorems and the lemma of Sect. 4.4.4 ensure that the local and composition conditions can be generalised to the whole ES level. Moreover, since the results are proved formally, it is not needed to check them for each particular model instance, but only the local and compositions conditions. This asymmetric character supplies the basis for the scalability gains achieved by this approach, as explained in the following sections.

5.3.1 Scalability on the Refinement

The first type of scalability gain achieved by our approach can be shown by comparing the refinement algorithms. In this respect, the phase SR/DAS is quite costly: $O(|A|\cdot|T|\cdot E \lg V)$,

since paths must be discovered in the whole DAS graph (see Sect. 5.1). As we discussed in Sect. 5.2, the refinement effort for the DAS/ES step is only $O(E)$, as long as results from the validation process are exploited (the validation costs will be analysed later on in the next section). Therefore, the total running time for the refinement is dominated by the costly path discoveries of the SR/DAS phase. However, this path discoveries must only consider the abstract and less numerous DAS elements instead of dealing with all the elements contained in the expanded subsystems.

To make the advantage of the DAS approach clear, let us compare this result with the procedure without using the DAS level, as that employed by previous work on model-based management of security systems like [15, 8]. In this case, the the set of service permissions (at the SR level) has to be refined into the PH level, which corresponds to a graph comprising all ESs in our methodology. A path discovery procedure similar to that described in Sect. 5.1 is executed, in order to find out the allowed paths between processes, but it must now consider all elements of the detailed view, so that the cost for each path discovery is $O(W \lg F)$ using the modified Dijkstra algorithm. Analogously to Sect. 5.1, we can thus estimate the total running time as $O(Pc_a \cdot Pc_t \cdot W \lg F)$, where Pc_a and Pc_t are the number of processes in the system that act as actors and targets, respectively[6].

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

$$\frac{|A| \cdot |T| \cdot E \lg V}{Pc_a \cdot Pc_t \cdot F \lg W} \tag{22}$$

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

Furthermore, we can rely on the experimental results gathered in [18] in order to estimate the gain in terms of growth behaviour. In that work, the DAS modelling was applied to two different scenarios: (E1) a simple hypothetical network environment (very similar to that exposed in Sect. 2.5); and (E2) a realistic application case, which is an extension of E1 to address an enterprise network. While the number of objects in the ES level (still called PH in that work) has increased by almost 470% from E1 to E2, the number of DAS elements has grown only 69%. Due to the aforementioned dependency on the environment, these results cannot be quantitative generalised. In qualitative terms though, we conclude that similar scalability gains in the refinement process can be expected from the introduction of the DAS in the modelling of large network environments, since they are similar to the analysed scenarios.

[6] Actually, in [8] the path discovery is performed on an auxiliary graph in which each protocol stack is grouped in a single node. This reduces a little the number of elements that must be considered in the search. Nevertheless, since this reduction affects only protocol objects and do not influence the overall model growth behaviour, I will not take it into account here.

5.3.2 Scalability on the Validation

With respect to the validation process, a situation which is quite the opposite of that with the refinement: in this case, the total cost is dominated by the local validation DAS/ES (Sect. 5.2) which is much more expensive than the simple SR/DAS checking (Sect. 5.1). As in the previous section though, this fact also contributes positively for the scalability of the approach. Indeed, the local consistency conditions are restricted to consider only the elements inside the boundaries of one subsystem in turn — with the exception of the elements of the neighbour subsystems which are directly linked to connectors — so the analysis process can be split up and can be performed independently on each subsystem, improving thereby the scalability of the validation.

In contrast, in the case in which the DAS layer does not exist, the analysis must consider all the elements of the detailed view, so that the verification effort using the same Floyd-Warshall algorithm (i.e. analogously to Sect. 5.2) would be bounded by $|W|^3$. To compare this with the total validation cost for the AS/ES validation given by Eq. 21, we must observe that $|W|$, i.e. the number of ES elements, is actually the sum of the elements in each ES $|W'_i|$ (with a small overlap of some processes which is not considered here). Thus the relation between the validation costs with and without the DAS is bounded by:

$$\frac{|W'_1|^3 + |W'_2|^3 + \cdots + |W'_s n|^3}{\left(|W'_1| + |W'_2| + \cdots + |W'_s|\right)^3}, \tag{23}$$

where s is the number of subsystems in the model. Although the asymptotic behaviour of both terms is cubic, one can perceive from Eq. 23 that the more subsystems a model has, the faster is the validation performed in relation to the approach without the DAS intermediary layer. Furthermore, with DAS the analysis of subsystems can be performed in parallel, achieving even more significant performance gains.

6 Summary

The main contribution of this report consists of offering a formal approach to the validation of policy hierarchies that use DAS for the configuration management of network security systems. Two criteria were selected to ensure the correctness of the refinement: the *completeness* and *consistency* among the different policy sets in the hierarchy. With basis 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 Diagram of Abstract Subsystems (DAS), and from the DAS to the Expanded Subsystems. As regards to this last phase, the structural conditions must only concern elements inside the same subsystem, for their validity in the whole level is assured by means of generalisation theorems and a lemma.

Subsequently, in order to be able to reason about the system behaviour that results from the application of the generated configuration, *model representativeness axioms* were defined

that formalise the implicit assumptions behind the model. These axioms state both the hypothesis one must assume about the mapping of real world entities by the model objects (*correct modelling*) and those concerning the effective application of normative aspects in the model to the real system (*correct implementation*). Relying upon axioms and conditions, two *validation theorems* were thus proved that ensure the compliance between the resulting system behaviour and the system view and the policies at the most abstract level considered (SR). The two theorems correspond to the elected validation criteria of *completeness* and *consistency*, so that the meaning of the formal proofs is: once a model complies with the conditions defined and the axioms are true, the refinement process is *correct*.

Furthermore, the computational effort required by the verification of the defined conditions and by the refinement algorithms have been analysed. The performance of refinement algorithms depends on idiosyncrasies of the modelled environment, so that it was not possible to achieve general quantitative results in this respect. Nevertheless, the analysis shows that relying upon data from the environments modelled so far — and assuming that these data are significant representatives for other similar environments — the introduction of the DAS layer can be expected to make the refinement process to perform faster and to scale better than the approach without using DAS. As for the validation of policy hierarchies, the DAS affords the splitting up of the system analysis, i.e. it enables the validation to be performed in a modular fashion, which is more appropriate when dealing with large models.

Acknowledgements

I am most indebt to Prof. Heiko Krumm. This work would certainly not reach the actual state without their insightful commentaries, our lively debates and their patient reviews of many previous versions of the report. Furthermore, this work would not be possible without the long-term support and advisory of Prof. Paulo Lício de Geus. I would like to express here my sincere gratitude to both.

I would also like to thank René jeruschkat for the implementation of the meta-model and refinement algorithms in the context of his diploma thesis [11].

References

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

[2] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, 1996.

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

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

[5] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[6] D. Ferraiolo and R. Kuhn. Role-based access control. In *Proceeedings of 15th NIST-NCSC National Security Computer Conference*, Baltimore, MD, 1992.

[7] D. F. Ferraiolo, J. F. Barkley, and D. R. Kuhn. A role-based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information and System Security*, 2(1):34–64, February 1999.

[8] Gereon Geist. Model-based management of security services: Integrated enforcement of policies in company networks. Master's thesis, University of Dortmund, Germany, 2003. in German.

[9] Institute of Electrical and Electronics Engineers, New York. *IEEE Standard Glossary of Sof tware Engineering Terminology*, 1990.

[10] ITU-T Telecommunications Standardization Sector of International Telecommunication Union. *X.810 – Security Frameworks for Open Systems: Overview*, November 1995.

[11] René Jeruschkat. Modellbasierte policy-verfeinerung für sicherheitsdienste großer vernetzter it-systeme. Master's thesis, University of Dortmund, March 2006. in German, english title: Model-based Policy Refinement for Security Services in Large-scale Networks.

[12] I. Lück, C. Schäfer, and H. Krumm. Model-based tool-assistance for packet-filter design. In E. Lupu M. Sloman, J. Lobo, editor, *Proc. IEEE Workshop Policy 2001: Policies for Distributed Systems and Networks*, number 1995 in Lecture Notes in Computer Science, pages 120–136, Heidelberg, 2001. Springer Verlag.

[13] I. Lück, M. Schönbach, A. Mester, and H. Krumm. Derivation of backup service management applications from service and system models. In B. Stiller R. Stadler, editor, *Active Technologies for Network and Service Management, Proc. DSOM'99*, number 1700 in Lecture Notes in Computer Science, pages 243–255, Heidelberg, 1999. Springer Verlag.

[14] I. Lück, S. Vögel, and H. Krumm. Model-based configuration of VPNs. In R. Stadtler and M. Ulema, editors, *Proc. 8th IEEE/IFIP Network Operations and Management Symposium NOMS 2002*, pages 589–602, Florence, Italy, 2002. IEEE.

[15] Ingo Lück. *Model-based Security Service Configuration*. PhD thesis, University of Dortmund, Germany, To appear in April 2006.

[16] J. D. Moffett and M. S. Sloman. Policy hierarchies for distributed system management. *IEEE JSAC Special Issue on Network Management*, 11(9), 11 1993.

[17] João Porto de Albuquerque, Holger Isenberg, Heiko Krumm, and Paulo Lício de Geus. Improving the configuration management of large network security systems. In Jürgen Schönwälder and Joan Serrat, editors, *Ambient Networks: 16th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2005, Barcelona, Spain, October 24-26, 2005, Proceedings*, volume 3775 of *Lecture Notes in Computer Science*, pages 36–47, Berlin Heidelberg, Germany, October 2005. Springer-Verlag.

[18] João Porto de Albuquerque, Heiko Krumm, and Paulo Lício de Geus. On scalability and modularisation in the modelling of security systems. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *Computer Security - ESORICS 2005, 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005, Proceedings*, volume 3679 of *Lecture Notes in Computer Science*, pages 287–304, Berlin Heidelberg, Germany, September 2005. Springer-Verlag.

[19] João Porto de Albuquerque, Heiko Krumm, and Paulo Lício de Geus. Policy modeling and refinement for network security systems. In *6th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2005), 6-8 June 2005, Stockholm, Sweden*, pages 24–33, Washington, DC, USA, 2005. IEEE Computer Society.

[20] G. Rothmaier. Model-based security management: Abstract requirements, trusts areas, and configuration of security services. Master's thesis, University of Dortmund, Germany, 2001. in German.

[21] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[22] R. S. Sandhu and P. Samarati. Access control: Principles and practice. *IEEE Communications*, 32(9), September 1994.

[23] Ravi S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11):9–19, 1993.

[24] M. Sloman and E. C. Lupu. Security and management policy specification. *IEEE Network, Special Issue on Policy-Based Networking*, 16(2):10–19, March/April 2002.

[25] A. Westerinen, J. Schnizlein, J. Strassner, M. Scherling, B. Quinn, S. Herzog, A. Huynh, M. Carlson, J. Perry, and S. Waldbusser. *Terminology for Policy-Based Management*. Internet Engineering Task Force, November 2001. RFC 3198.

[26] R. Wies. Using a classification of management policies for policy specification and policy transformation. In Adarshpal S. Sethi, Yves Raynaud, and Fabienne Fure-Vincent, editors, *Integrated Network Management IV*, volume 4, pages 44–56, Santa Barbara, CA, 1995. Chapman & Hall.

[27] E. D. Zwicky, S. Cooper, and D. B. Chapman. *Building Internet Firewalls*. O'Reilly and Associates, Sebastopol, CA, 2nd edition, 2000.

A Refinement Algorithms

Algorithm 1 defines the refinement process from SR to DAS; i.e. the generation of a set of $ATPs$ for each service permission in SP. Hence, for each *Actor* that refines the *User* and *SubjectType* associated with the service permission *sp* being refined (line 2), the algorithm executes the auxiliary function $findpaths()$ (line 6) providing as parameter the set of *Targets* related to *sp*. This set comprises *Targets* in two different cases: a) if the *Service sv* (associated to *sp*) has direct access to the *Ressource r*, the related *Target* will refine these two objects through the relation RT (line 4); b) if there is a service dependency, then the related *Target* will refine *r* and another *Service sd*, on which *sv* depends to access *r* through the relation $SvDep$ (line 5). Notice that, for the sake of notation conciseness, the service dependency is henceforth restricted to one level.

The function $findpaths(P, a, T)$, in turn, is a straightforward adaptation of the classical *modified Dijkstra algorithm* for discovery of single-source shortest paths [5, Ch. 25]. It returns the set P of all paths between the initial node a and any node in the destination set T and is thus not shown in details here. In the sequence, the algorithm verifies each path found by invoking the function $testpath()$; if the path is considered valid, its security requirement is set as the one of the related service permission (line 9) and it is then added to the ATP set (line 10).

Algorithm 1 Refinement $SR \rightarrow DAS$

```
 1: for all sp(u, st, sv, r) ∈ SP do
 2:   for all a : (a, u, st) ∈ RA do
 3:     P ← ∅ {set of path candidates}
 4:     T ← {t : (t, sv, r) ∈ RT} ∪
 5:           {t : (t, sd, r) ∈ RT ∧ (sv, sd, r) ∈ SvDep} {set of targets related to sp}
 6:     findpaths(P, a, T)
 7:     for all p ∈ P do
 8:       if testpath(p, sp) then
 9:         sr[p] ← sr[sp]
10:         ATP ← ATP ∪ p
11:       end if
12:     end for
13:   end for
14: end for
```

Algorithm 2 presents the $testpath(p, sp)$ function, which takes as input the path p to be tested and the corresponding service permission *sp*. The path shall be tested into two different aspects: i) satisfaction of service dependencies; and ii) compatibility between *Actor* and *Target*.

In the *for*-loop of lines 2–6, the service dependencies related to the permission are checked. The loop then tests if, for each service dependency, a corresponding mediator is present in the path (line 3). In the negative case, the path is considered invalid.

The second test applied to the path verifies the compatibility of the related *Actor* (say

Algorithm 2 Function $testpath(p, sp : (u, st, sv, r)) : boolean$

```
1: sa[p] ← sa[sv]; testpath ← true
2: for all sd : (sv, sd, r) ∈ SvDep do {tests service dependencies}
3:    if ∄ m : (m, sd) ∈ RM and m ∈ p then
4:       testpath ← false
5:    end if
6: end for
7: if not path_compatible(p) then
8:    testpath ← false
9: end if
```

a) and *Target* (say t); i.e. it tests whether the *protocol* and *connection orientation* defined for a in the AS expanded view matches the corresponding definitions for t. In the case there is a mediator along the path, the compatibility is tested first from the *Actor* to the *Mediator*, and then from this to the *Target*; for the *Mediator* can here act as a gateway. For instance, in Fig. 4 of Sect. 2.4, the process "Netscape" (of host "WS2") is assigned to the protocol "http", and the diamond near the latter indicates that the former will initiate the communication (i.e. has outgoing orientation). The corresponding *Actor* "internal web clients" is hence compatible with the *Mediator* "Webproxy", since the process "Squid-Proxy" assigned to the latter is as well connected to a "http" protocol, but has an incoming orientation. The algorithm for this verification is trivial and is thus omitted here.

The interested reader may find a more detailed explanation of the refinement algorithms and of the configuration generation in [11].

B Proof of the Generalisation Theorems

B.1 Proof of GT1

The proof of this theorem is based on an induction in 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 4.1.9). Thus all processes along dp_1 pertain to the same AS, and the basis hypothesis can be stated as

$$dp_1 \subseteq aep_1 \in AEP : \ pc_1, \ldots, pc_m \text{ is the process sequence in } dp_1 \\ \wedge \ sub[pc_1] = sub[pc_2] = \cdots = sub[pc_m] \quad (24)$$

Hence, applying this hypothesis to Cond. 4.4.1.2 follows that there is a corresponding abstract path ap_1 in DAS, such that

$$ap_1 \subseteq atp_1 \in ATP, \ ap_1 = \langle v_1, \ldots, v_m \rangle : expand[dp_1, ap_1]$$

The definition of the predicate *expand* (Definition 4.4.1) implies that

$$(e_i, v_i) \in RPc \ \vee \ (e_i = v_i) \text{ for } 1 \leq i \leq n \quad (25)$$

Where $e_1, \ldots, e_n$ is the sequence of processes and connectors of dp_1. But the hypothesis in Eq. (24) states that all elements of dp_1 pertain to the same AS, so that the structure of AEP elements (described in Definition 4.1.9) implies that there are no connectors in dp_1 (i.e. $n = m$). Hence, the right-hand term of the disjunction in Eq. (25) is false for all elements — since only connectors can be contained in both an element of ATP and an element of AEP. Equation (25) then becomes

$$(pc_i, v_i) \in RPc \text{ for } 1 \leq i \leq m \tag{26}$$

Considering now a par of subsequent DAS elements in ap_1, said v_j and v_{j+1}, Cond. 4.4.2.5 asserts that:

$$\forall\ pc_a, pc_b \in Pc : (pc_a, v_j) \in RPc\ \wedge\ (pc_b, v_{j+1}) \in RPc \Rightarrow \\ \exists\ p = \langle pc_a, \ldots, pc_b \rangle : lconn[p] \tag{27}$$

Clearly, the universal quantifier in Eq. (27) implies that also for pc_j and pc_{j+1} that are respectively connected to v_j and v_{j+1} by Eq. (26):

$$\exists\ p = \langle pc_j, \ldots, pc_{j+1} \rangle : lconn[p] \tag{28}$$

Since pc_j and pc_{j+1} were chosen without loss of generality, Eq. (28) is valid for each pair of subsequent processes in dp_1. Definition 4.4.5 asserts that a local-connected path is also connected, so that we prove, as intended, that for the basis case:

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

For the general case, let us assume:

$$dp_2 \subseteq aep \in AEP :\ pc_1, \ldots, pc_o \text{ is the process sequence in } dp_2,\ \ dp_2 \text{ spans } k \text{ ASs} \tag{29}$$

The induction hypothesis then reads:

$$\textit{Theorem 4.4.4.1 is valid for all } dp \subseteq atp \in ATP : dp \textit{ spans up to } k-1 \textit{ ASs} \tag{30}$$

Proceeding in analogy with the basis case, from Eq. (29), Cond. 4.4.1.2, and Definition 4.4.1 follows that there is a corresponding ap_2 in DAS, such that:

$$ap_2 \subseteq atp_2 \in ATP,\ ap_2 = \langle v_1, \ldots, v_p \rangle :\ (e_i, v_i) \in RPc\ \vee\ (e_i = v_i) \text{ for } 1 \leq i \leq p \tag{31}$$

Where $e_1, \ldots, e_p$ is the sequence of processes and connectors of dp_2. Now let us consider that $e_r = v_r$ is the first connector in this sequence (thus $e_1 = pc_1, \ldots, e_{r-1} = pc_{r-1}$, since they are all processes), and dsp_1 is the subpath that contains the first elements of dp_2 until process pc_{r-1}. According to the Definition 4.1.9, dsp_1 is thus completely enclosed in one subsystem; i.e. $sub[pc_1] = sub[pc_2] = \cdots = sub[pc_{k-1}]$. Furthermore, the same Definition 4.1.9 implies

$e_{r+1} = v_{r+1}$ is also a connector, and pertains to a different subsystem. Hence, we can apply the basis case to dsp_1, achieving that:

$$\exists\ p = \langle pc_j, \ldots, pc_{j+1} \rangle : conn[p] \text{ for } 1 \leq j < r-1 \tag{32}$$

Now, consider dsp_2 the subpath of dp_2 that starts at process e_{r-1}, passes through the connectors e_r and e_{r+1} and ends at the subsequent process e_{r+2}. From Eq. (31) there is a corresponding DAS subpath in ap_2:

$$asp_1 = \langle v_{r-1}, v_r, v_{r+1}, v_{r+2} \rangle :\ (v_{r-1}, v_r) \in E\ \wedge\ (v_r, v_{r+1}) \in E\ \wedge\ (v_{r+1}, v_{r+2}) \in E \tag{33}$$

And:

$$(e_{r-1}, v_{r-1}) \in RPc\ \wedge\ e_r = v_r \in C\ \wedge\ e_{r+1} = v_{r+1} \in C\ \wedge\ (e_{r+1}, v_{r+1}) \in RPc \tag{34}$$

Thus, applying Eq. (33) to Cond. 4.4.3.1 comes:

$$\forall\ pc_x, pc_y \in Pc : (pc_x, v_x) \in RPc\ \wedge\ (pc_y, v_y) \in RPc \Rightarrow \exists\ p = \langle pc_x, \ldots, pc_y \rangle : cconn[p] \tag{35}$$

Clearly, the universal quantifier in Eqs. (35) and (34) imply that:

$$\exists\ p = \langle e_{r-1}, \ldots, e_{r+2} \rangle : cconn[p] \tag{36}$$

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 4.4.5 we can then extend Eq. (32) to:

$$\exists\ p = \langle pc_j, \ldots, pc_{j+1} \rangle : conn[p] \text{ for } 1 \leq j < r \tag{37}$$

Now, let dsp_3 be the subpath of dp_2 that goes from $e_{r+2} = pc_r$ to the end (e_p). Since Eq. (29) states that dp_2 spans k subsystems, and dsp_3 is the result of removing the subpath contained in the first subsystem of dp_2 (i.e. the subpath $\langle e_1, \ldots, e_r \rangle$), dsp_3 thus spans $k-1$ subsystems. Hence, the induction hypothesis Eq. (30) can be applied to dsp_3 achieving that:

$$\exists\ p = \langle pc_l, \ldots, pc_{l+1} \rangle : conn[p] \text{ for } r \leq l < n \tag{38}$$

Therefore, from Eqs. (37) and (38) follows:

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

B.2 Proof of GT2

Relying upon Definition 4.4.5, the theorems' assertion can be expanded into two subgoals to be proved:

$$\forall\ pc_x, pc_y \in Pc : \exists\ p_1 = \langle pc_x, \ldots, pc_y \rangle \wedge\ lconn(p_1) \Rightarrow\ \exists\ aep_1 \in AEP : pc_1, \ldots, pc_m \text{ is the sequence of processes in } aep_1 \wedge\ pc_x = pc_i\ \wedge\ pc_y = pc_{i+1} \text{ for } 1 \leq i < m \tag{39}$$

And:

$$\forall\ pc_z, pc_w \in Pc : \exists\ p_2 = \langle pc_z, \ldots, pc_w \rangle \wedge\ cconn(p_2) \Rightarrow\ \exists\ aep_2 \in AEP : \\ pc_1, \ldots, pc_o \text{ is the sequence of processes in } aep_2 \\ \wedge\ pc_z = pc_j\ \wedge\ pc_w = pc_{j+1} \text{ for } 1 \leq j < o \quad (40)$$

Beginning with Eq. (39), from Cond. 4.4.2.6 follows:

$$\exists\ v_x, v_y \in V : (pc_x, v_x) \in RPc\ \wedge\ (pc_y, v_y) \in RPc\ \wedge\ (v_x, v_y) \in E \quad (41)$$

Cond. 4.3.2.3 and Eq. (41) thus imply:

$$\exists\ atp_1 \in ATP : \langle v_x, v_y \rangle \subseteq atp_1 \quad (42)$$

Therefore, if we consider $atp_1 = \langle v_1, \ldots, v_n \rangle$ then $v_x = v_i$ and $v_y = v_{i+1}$ for some $1 \leq i < n$. Now, from Cond. 4.4.1.1 follows:

$$\forall\ w_1 \in Pc,\ w_m \in Pc :\ (w_1, v_1) \in RPc \wedge\ (w_m, v_n) \in RPc \Rightarrow \\ \exists\ aep \in AEP,\ aep = \langle w_1, \ldots, w_n \rangle : expand[aep, atp_1] \quad (43)$$

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 4.1.3) follows that $v_i = v_1 \in A$ (it is an *Actor*) and $v_{i+1} = v_n \in T$ (it is a *Target*). Thus, the universal quantifier in Eq. (43) implies, that also for pc_x and pc_y correspondingly related to $v_x = v_i$ and $v_y = v_{i+1}$ by Eq. (41):

$$\exists\ aep_1 \in AEP,\ aep_1 = \langle pc_x, \ldots, pc_y \rangle \quad \blacksquare$$

In the second case, $i > 1$ and $i + 1 < n$, and Definition 4.1.3 implies $v_i \in M$ and $v_{i+1} \in M$ (both are *Mediators*). Hence, Condition 4.4.2.2 implies that there is only one process related to each mediator, so that from Eq. (41), Eq. (43) and Definition 4.4.1 (definition of function *expand*) follows that:

$$\exists\ aep_1 \in AEP,\ aep = \langle w_1, \ldots, pc_x, \ldots, pc_y, \ldots, w_n \rangle \quad (44)$$

The third and fourth cases (namely $i = 1\ \wedge\ i + 1 < n$; and $i > 1\ \wedge\ i + 1 = n$) can be analogously demonstrated by a combination of the two previous cases. ∎

Now, to prove Eq. (40) we must apply Cond. 4.4.3.2, achieving:

$$\exists\ v_z, v_w \in V, c_1, c_2 \in C :\ (pc_z, v_z) \in RPc\ \wedge\ (pc_w, v_w) \in RPc \\ \wedge\ (v_z, c_1) \in E\ \wedge\ (c_1, c_2) \in E\ \wedge\ (c_2, v_w) \in E \quad (45)$$

From Eq. (45) and Cond. 4.3.2.4 comes:

$$\exists\ atp_2 \in ATP : \langle v_z, c_1, c_2, v_w \rangle \subseteq atp_2 \quad (46)$$

Therefore, if we consider $atp_2 = \langle v_1, \ldots, v_n \rangle$ then $v_z = v_k$, $c_1 = v_{k+1}$, $c_2 = v_{k+2}$, and $v_w = v_{k+3}$ for some $1 \leq k < n-3$. From Cond. 4.4.1.1 thus follows:

$$\forall\ w_1 \in Pc,\ w_o \in Pc :\ (w_1, v_1) \in RPc \wedge\ (w_o, v_n) \in RPc \Rightarrow \\ \exists\ aep \in AEP,\ aep = \langle w_1, \ldots, w_o \rangle : expand[aep, atp_2] \quad (47)$$

Hence, four analogous subcases to the ones in the first part of this proof must be considered, in order to cover the different positions that v_z and v_w may occupy in atp_2 (namely i) $k = 1\ \wedge k+3 = n$; ii) $k > 1\ \wedge k+3 < n$; iii) $k = 1\ \wedge k+3 < n$; and iv) $k > 1\ \wedge k+3 = n$). Proceeding in the same manner as before, we can prove for all four subcases that:

$$\exists\ aep_2 \in AEP,\ aep_2 = \langle w_1, \ldots, pc_z, \ldots, pc_w, \ldots, w_o \rangle \quad \square$$

C List of Figures

List of Figures