MDA-Based Framework for Automatic Generation of Consistent Firewall ACLs with NAT

Sergio Pozo, A.J. Varela-Vaca, and Rafael M. Gasca

QUIVIR Research Group, Department of Computer Languages and Systems
Computer Engineering College, University of Seville
Avda. Reina Mercedes S/N, 41012 Sevilla, Spain
{sergiopozo, ajvarela, gasca}@us.es
http://www.lsi.us.es/~quivir

Abstract. The design and management of firewall ACLs is a very hard and error-prone task. Part of this complexity comes from the fact that each firewall platform has its own low-level language with a different functionality, syntax, and development environment. Although several high-level languages have been proposed to model firewall access control policies, none of them has been widely adopted by the industry due to a combination of factors: high complexity, no support of important features of firewalls, no common development process, etc. In this paper, a development process for Firewall ACLs based on the Model Driven Architecture (MDA) framework is proposed. The framework supports the market leaders firewall platforms and is user-extensible. The most important access control policy languages are reviewed, with special focus on the development of firewall ACLs. Based on this analysis a new DSL language for firewall ACLs, AFPL2, covering most features other languages do not cover, is proposed. The language is then used as the platform independent metamodel, the first part of the MDA-based framework.

Keywords: firewall, acl, ruleset, framework, language.

1 Introduction

A firewall is a network element that controls the traversal of packets across different network segments. It is a mechanism to enforce an Access Control Policy, represented as an Access Control List (ACL), or a rule set. Firewalls use obligation policies, (also known as Event Condition Action Rules (ECA) that must perform certain actions when certain events occur. By contrast, authorisation policies permit or deny actions based upon the action, the source of the action and the target of the action. Thus, a layer 3 Firewall ACL is in general a list of linearly ordered (total order) condition/action rules. Let ACL_f be a firewall ACL consisting of $f+1$ rules, $ACL_f = \{R_0, ...R_f\}$. Consider as a rule $R_j \in ACL_f =< H, Action >, H \subseteq Z, 0 \leq j \leq f$, $Z = protocol \times srcIP \times srcPrt \times dstIP \times dstPrt$, where $Action = \{allow, deny\}$ is its action. A selector of a firewall rule R_j is defined as $R_j[k], k \in H, 0 \leq j \leq f$. A rule R_j matches a packet p when the values of each field of the header of a packet

O. Gervasi et al. (Eds.): ICCSA 2009, Part II, LNCS 5593, pp. 130–144, 2009.

$p[k], k \in H$ are subsets or equal to the values of the rule selector $R_j[k], k \in H, 0 \leq j \leq f$ (i.e. $p[k] \subseteq R_j[k], \forall k \in H$).

Firewalls have to face many problems in modern networks. Two of the most important ones are the high complexity of ACL design [1] and ACL consistency diagnosis [2, 3]. Networks have different access control requirements which must be translated by a network administrator into firewall ACLs. Writing and managing ACLs are tedious, time-consuming and error-prone tasks for a wide range of reasons [4]. Low-level firewall languages are, in general, hard to learn, use and understand. In addition, each firewall platform has its own low-level language which usually is very different from other vendors' ones. Changing from one firewall platform to another often means a complete rewrite of the ACL. In this translation process, inconsistencies and redundancies can be introduced [2, 3]. Figs. 1 and 2 present two fragments of ACLs written in IPTables and Cisco PIX respectively to give an idea of the complexity and differences of these languages. Note that the number of rules of a Firewall ACL may range between a few ones and 5000 [5].

Many third-party domain specific languages (DSLs) have been proposed to abstract the network administrator from the underlying firewall platform details and language syntax [6, 7, 8, 9, 10, 11]. A domain specific language provides more possibilities to network administrators, since it can raise the abstraction level of the problem domain using its own concepts. However, as we are going to show in the next section, the proposals of DSLs have different problems regarding different aspects of design and deployment of ACLs.

The result is that these languages are not widely adopted by industry. This possibly is not only due to their problems, but also to the need of companies to maintain a large user base tied to a particular low-level language and firewall platform.

We think that there is a clear need of a DSL for Firewalls with the expressive power of existing low-level firewall-specific languages, but with significantly less

```
-A FORWARD -i   -s 192.168.1.0  -d 170.0.1.10  -p tcp  -m tcp --sport any  --dport 21  -p tcp  -j ACCEPT
-A FORWARD -i   -p tcp  -p tcp  -j DROP
-A FORWARD -i   -s 192.168.1.0  -d 170.0.1.10  -p udp  -m udp --dport 53  -p udp  -j ACCEPT
-A FORWARD -i   -d 170.0.1.10  -p udp  -m udp --dport 53  -p udp  -j ACCEPT
-A FORWARD -i   -s 192.168.2.0  -d 170.0.2.0  -p udp  -p udp  -j ACCEPT
-A FORWARD -i   -p udp  -p udp  -j DROP
```

Fig. 1. IPTables ACL fragment

```
access-list acl-out permit gre host  192.168.201.25  host 192.168.201.5
access-list acl-out permit tcp host  192.168.201.25  host 192.168.201.5  eq 1723
static (inside,outside) 192.168.201.5  10.48.66.106 netmask 255.255.255.255 0 0
access-group acl-out in interface outside
access-list acl-out permit udp host  192.168.201.25  host 192.168.201.5  eq 1701
static (inside,outside) 192.168.201.5  10.48.66.106 netmask 255.255.255.255 0 0
access-group acl-out in interface outside
```

Fig. 2. Cisco PIX ACL fragment

complexity than currently proposed high-level policy languages. In addition, this language must have the possibility of automatic compilation to the market-leader low-level firewall languages, and be easily user-extensible in order to support new features and firewall platforms. Recently, we have proposed a new high level firewall DSL with all these capabilities, called Abstract Firewall Policy Language (AFPL) [12]. The DSL was defined in the XML technological space. In this paper we propose an extension of AFPL with Network Address Translation (NAT) [16], a must-have feature of firewall languages. To the best of our knowledge, AFPL2 is the first firewall DSL to support NAT.

However, with AFPL some advanced features of firewall platforms cannot be modeled, since the language is an abstraction based on the common features of the market-leaders. For this reason, we propose a framework where AFPL2 can be used along with the integration of other lower-level concepts related to particular platforms. This framework is heavily based on the concepts of MDA extended with a model consistency stage to guarantee the quality of the resulting compiled ACL. The framework is extensible by end-users, in the sense that more concepts can be added to the meta-models, as well as modifications to the transformations between them, in order to represent more features and/or low-level firewall platforms and languages.

The structure of this paper is as following: in section 2 related works are described. In section 3 the MDA-based framework for Firewalls is described. In section 4 a DSL with Network Address Translation support is proposed as the MDA-based framework PIM. We conclude in section 5 and propose a research direction for our future works. Finally, in Appendix I it is presented the analysis of firewall platforms (NAT features).

2 Related Works

In [14] the authors propose a high-level language, Firmato, which models ACLs as ERDs in order to automatically generate low-level firewall ACLs. However, the complexity of Firmato is similar to that of many low-level languages. Two major limitations of Firmato are that (1) it does not support NAT and (2) can only represent knowledge in positive logic (allow rules), which complicates the specification of exceptions (a rule with a general allow action, immediately preceded by a more restrictive rule with a deny action). This could result in the need to write a lot of rules to express them. However, as a lateral effect, rules are always consistent and order-independent. FLIP [15] is a recently proposed firewall language which can also be compiled into several low-level ones, although in the paper no more information about his feature is provided. Their authors claim that ACLs expressed in FLIP are always consistent. In fact they are because of one of its limitations: it does not support overlapping between rule selectors. Prohibiting the use of overlaps is a major limitation, since it is impossible to express exceptions. In addition, its syntax is even more complex than Firmato's one. However, due to this lack of expressiveness, FLIP ACLs are order independent. Finally, NAT is not supported in FLIP. In [7] the authors provide a general language, Ponder, to represent network policies (in general), which cannot compile to any low-level language. A re-engineered version, Ponder2, is also available. However, the complexity of Ponder surpasses the needs of firewall ACLs. In theory, a language that can express any network policy could express a firewall

Table 1. Survey of features of access control languages

	Firmato	Ponder2	FLIP	SRML	Rule-ML	PCIM	XACML	AFPL
FW-Specific	*x*	*x*	√	*x*	*x*	*x*	*x*	√
User extensible	*x*	*x*	*x*	*x*	√	*Partial*	√	√
Consistency diagnosis	*x*	*x*	√	*x*	*x*	*x*	*x*	*x*
Redundancy diagnosis	*x*	*x*	√	*x*	*x*	*x*	*x*	√
Support stateful rules	√	*N/A*	√	*N/A*	*N/A*	*N/A*	*N/A*	√
Support stateless rules	√	*N/A*	√	*N/A*	*N/A*	*N/A*	*N/A*	√
Support NAT	*x*	*N/A*	*x*	*N/A*	*N/A*	*N/A*	*N/A*	*x*
Positive logic	√	√	√	√	√	√	√	√
Negative logic	*Partial*	√	√	√	√	√	√	√
Selector-values overlap	√	√	*x*	√	√	√	√	√
User-controlled rule order	*x*	√	*N/A*	*x*	*x*	√	√	√
Topology/logic separation	√	*N/A*	√	*N/A*	*N/A*	*N/A*	√	√
Relative complexity	*High*	*High*	*Low*	*Low*	*Low*	*Medium*	*Low*	*Low*
Compilation to low-level	√	*x*	√	*x*	*x*	*x*	*x*	√
Low-level lang. import	*x*	*x*	*x*	*x*	*x*	*x*	*x*	*Partial*

access control policy. However, concepts such as NAT cannot be expressed with Ponder. AFPL [12] is a language developed after an analysis of the features of major firewall languages, supporting most of their functionality at a fraction of their complexities. It can express stateful and stateless rules (although an administrator does not need to know these kind of details, since complexity is hidden in the language), positive and negative rules, overlappings, exceptions, and can be compiled to six market-leader firewall languages.

Some organizations have even proposed languages to represent access control policies as XML documents, such as XACML [8], PCIM [9], Rule-ML [10], and SRML [11]. However, none of these languages is specific enough for firewall access control policies, resulting in a complexity to express firewall concepts, or in an impossibility to express them at all (this is the case of NAT for all these languages). Even UML has been proposed to model access control policies [17]. However, in our problem domain, UML could be an aid for the requirements definition stage, but then these models need to be translated into a DSL. These models and languages are very generic and are not intended for the area of any particular access control problem. Table 1 presents a survey of the most important features of the reviewed languages (related to express firewall ACL knowledge).

With respect to commercial or Open Source applications, the two most important ones are Firewall Builder and Solsoft ChangeManager. However, these kind of applications have been left out of the comparison, because they are not based on a model of a firewall, but rather on an abstraction of their command-line syntax. For example, in these two solutions, the firewall platform must be specified upon firewall instantiation, since each firewall platform supports a different set of features. The reviewed proposals and the one proposed in this paper are focused towards a unique model for all firewalls.

3 MDA for Firewall ACL Design

In the last few years Model-Driven Development (MDD) has promoted the use of models and transformations in software development processes [19]. The Model-Driven

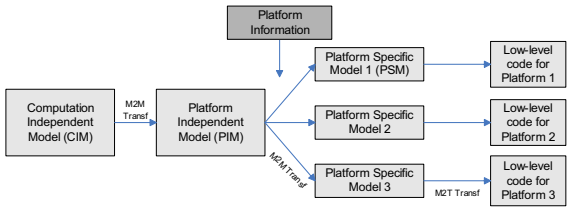

Fig. 3. MDA Framework

Architecture [18] is the approach for MDD promoted by the Object Management Group (OMG) (Fig. 3).

However, the use of the MDA framework does not guarantee that the model is free of inconsistencies and redundancies. Moving the verification stage to earlier stages in the process, prior to code generation, can reduce budget dedicated to consistency diagnosis and correction of the final ACL. In an MDD approach, this is even more important, since models are the core of the methodology, and executable code will be automatically generated from them. It is important to note that, since the focus of MDD paradigm is on the creation of static models, there are no execution details. Thus, the proposed diagnosis stage is in reality a (static) verification one. This diagnosis stage has been proposed in earlier works [1], and is not the focus of the paper.

Fig. 4 shows the proposed framework, which follows MDA, but with the inclusion of two diagnosis stages, one for the PIM and other for the PSM. Note that the PSM diagnosis stage is only necessary (1) if the information regarding the PIM is modified; or (2) when information is modified or added by an end-user to the PSM. If an inconsistency is found, then it must be corrected before applying the next model transformation.

3.1 Modeling Considerations

Firewall platforms are very different from one vendor to another, and even among the available Open Source platforms. These differences range from differences in the number, type, and syntax of selectors that each platform's filtering algorithm can handle, to huge differences in rule-processing algorithms that can affect the design of the ACL. Fortunately, the vast majority of filtering actions can be expressed with any of the filtering languages and platforms, with the only difference on the number of rules needed, and/or in their syntax. For example, an IP address range can be translated into several blocks of single IP addresses, so both syntaxes are equivalent.

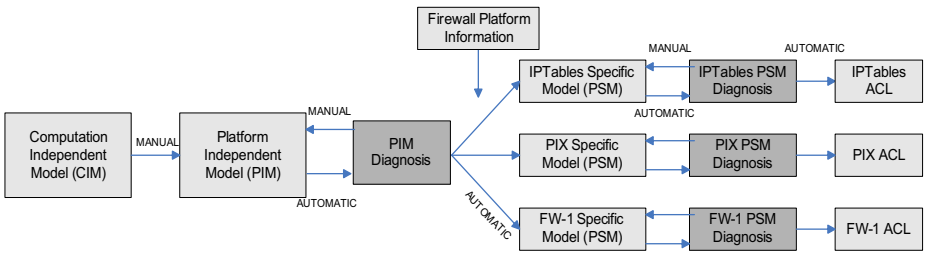

Fig. 4. MDA Framework for Firewalls (with Model Verification)

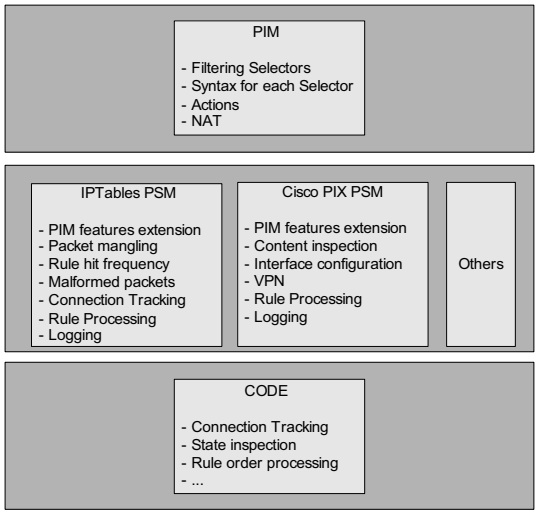

Fig. 5. PIM and PSM meta-models proposal

With the focus on modelling firewall languages and platform functionality, some questions may arise. The first one is whether all firewall platforms analyzed share a common set of filtering selectors (or a common set of functionality). Another one is, for the common set of selectors, if there is at least one common syntax among all firewall platforms (in order to be able to use the functionality); or if not, if the available syntaxes for each platform have equivalencies in the other ones (i.e. are emulable). This part is in fact, the configuration of the ACL an administrator can design.

For the design of AFPL2 we use the same methodology as for AFPL [12]: first a DSL with a set of selectors (or features) and syntaxes supported by all the analyzed firewall platforms and languages is created. Then, the non common selectors and syntaxes are analyzed and only added to AFPL2 if they comply to a criterion that is going to be defined in the next section. This methodology yielded in the case of AFPL to a lightweight language with a very simple syntax that would satisfy the vast majority of administrators. We expect the same for AFPL2. The methodology is described in the next sections with more detail, but has been described in detail in [12]. We propose to consider AFPL2 part as the PIM in our framework.

However, for each analyzed firewall language and platform there is a set of functionality that has not been modelled in the PIM. There are basically two options for it: to model it in another (lower) level of the framework, or to completely remove it. Removal would cause an advanced administrator needing to use it to modify the generated ACL. However, if this part is introduced in a lower level meta-model, then the administrator can model this advanced behaviour without modifying the ACL. For this reason, we propose to consider this platform-specific functionality as the PSM for each platform. Note that a PSM could modify in some way the PIM (during the automatic M2M transformation, or with a direct modification of the administrator over the PSM), and thus new inconsistencies could be introduced (this is the reason why in

Fig. 4 there is a diagnosis stage before the M2T transformations). With this approach the PIM is as simple as possible, serving for the vast majority of administrators, while the PSM facilitates the use of platform specific features (all of them if needed).

Besides filtering, firewall platforms have other specific features. These are for example how each platform threats connection tracking (that is, stateful or stateless connections), how the rule processing is performed (forward, backward, with jumps), etc. However, part is related to how each platform executes the ACL, and thus an administrator cannot modify this behaviour. For these reasons, this behaviour is considered only in the M2T transformation from PSM to low-level ACL.

We think that the proposed concept separation fits well with the MDA approach. Fig. 5 shows the proposed feature division for Firewalls.

4 AFPL with NAT. PIM Meta-model

In this work, we depart from previous results for the DSL design, where various alternative models for a Firewall DSL were created in a bottom-up process, and discussed. The result was Abstract Firewall Policy Language (AFPL) [12]. The main goals of AFPL were simplicity, ease of use, and support of a common set of functionality valid for most administrators. In this section, AFPL is extended to support NAT. This new DSL, AFPL2, serves as the PIM meta-model in the proposed MDA framework.

4.1 AFPL Extended with NAT: AFPL2

NAT is a must-have feature of modern low-level firewall languages (it was defined in the year 1999), and nowadays all firewall platforms support it. The main idea behind NAT is to change *(translate)* the values of some headers of TCP/IP packets in different situations. These changes are specified using rules *(translation rules)* in a similar way as filtering rules are specified. There are mainly two modes of NAT (as defined in RFC2663 [16]).

- **Source NAT (SNAT).** Also know as Outbound NAT, Network Address Port Translation (NAPT), or Masquerading, in which the source of a packet is translated when it traverses an outbound interface of a firewall. Response packets are translated back to their real address.
- **Destination NAT (DNAT).** Also known as Packet Forwarding, in which the destination of a packet is translated when it traverses an inbound interface of a firewall. Response packets are translated back to their real address.

However, although NAT has been defined in an RFC, it has not been standardized. For this reason, an analysis of the NAT features supported by the market-leader firewall platforms is needed for the design of AFPL2 in order to satisfy the vast majority of administrators.

The considered firewall platforms in the NAT analysis for AFPL2 are the same ones as for AFPL: IPTables 1.4.2, Cisco PIX 8, FreeBSD 8 IPFilter, FreeBSD 8 IPFirewall, OpenBSD 4.1 Packet Filter, and Checkpoint Firewall-1 4.1. The analysis is presented in Appendix I and shows the supported NAT modes, its filtering selectors, and available syntaxes.

Table 2(a). AFPL2 Source NAT

Translated Selector	Obligation	Syntax	Comments
Source IP Address	Mandatory	-Host IP -Interface name	If the interface name is given, the interface IP is used (it could be dynamic link)

Table 2(b). AFPL2 Destination NAT

Translated Selector	Obligation	Dependencies	Syntax
Destination IP Address	Mandatory		-Host IP
Destination Port	Optional	Destination port must be specified in the original packet	-Number -Range: [p1,p2]

Our start point is the factorized AFPL2 model presented in Table 2. A factorized model is a model where only common NAT modes, selectors and syntaxes are defined. We take it as a basic NAT model. Note that NAT use both filtering rules for matching packets and translation rules for defining which selectors of the matched packets and how will be translated. In Table 2 only the part related with translation rules is presented, since filtering was covered in AFPL [12]. In the next sections, non-common functionality is going to be analyzed for its inclusion in AFPL2.

4.1.1 Addition of Uncommon NAT Modes

Although there are a lot of ways of expressing translations, only two kinds of translation rules are supported in AFPL2 (Table 2). In fact, the analyzed firewall languages support more NAT modes (analyzed in Appendix I). There, we show that all these modes are in reality variations of the two basic NAT modes defined in RFC2663 (Source and Destination NAT) and can be reproduced in one way or another using these two basic types. Although RFC2663 defines more NAT modes (like Twice NAT or Multi-homed NAT), they are not supported in any of the analyzed firewall platforms. For these reasons, no more NAT modes are necessary in AFPL2.

4.1.2 Addition of Uncommon Selectors

An uncommon selector can be added if its functionality can be reproduced (emulated) with the selectors of the factorized model, and it also adds new functionality to the model. A selector *adds new functionality* to the model if it cannot be emulated with translation rules which do not contain it. The conclusion is that using this criterion, no more selectors can be added to AFPL2. An exhaustive list of selectors per firewall is presented in Appendix I.

4.1.3 Addition of Uncommon Syntaxes

In this section, we analyze the possibility of supporting uncommon syntaxes in the considered selectors. In general, we consider an uncommon syntax as a candidate for

its addition to the collection of supported syntaxes for that selector in AFPL2, if it can be emulated with the common syntaxes of the same selector and it provides clear usability improvements for human users. A syntax provides clear usability improvements if its use by a human cannot introduce inconsistencies [3] in an ACL created with AFPL2 and it provides compactness. Again, we will base this analysis on results presented in Appendix I.

- **SNAT Source IP address.** The uncommon syntaxes of this selector are identifiers, block IPs, IP ranges and collections of IPs. Note that the use of identifiers provides a clear usability improvement and does not introduce inconsistencies in the ACL, and thus will be considered for AFPL2. All the other syntaxes provide ACL compactness, and also represent usability improvements. As they cannot cause ACL inconsistencies, they will also be considered (except IP ranges and IP collections, which are redundant). Block IPs, IP ranges, and in general collections of IPs can be emulated in low-level languages that do not support them by decomposing these collections of IPs into several unique IPs, and defining one NAT rule for each.
- **DNAT Destination IP address.** The uncommon syntaxes of this selector are identifiers and IP ranges. Identifiers are included for the same reasons stated for SNAT source IPs. However, IP ranges cannot be included because it is only used for load balancing, a non-emulable feature not supported by all the analyzed platforms.
- **DNAT Destination port.** Many range syntaxes are possible in many firewall platforms, as is the case of ranges ‘<p’, ‘<=p’, ‘>p’, ‘>=p’, ‘(p1, p2)’ and ‘)p1, p2(‘. These syntaxes provide no new functionality or a clear usability improvement, and can be easily emulated with the common ‘[p1, p2]’ syntax without loss of functionality. For this reason they will not be included in AFPL2.

In order to match the original packet, the same selectors and rule format used for filtering in AFPL can be used without restrictions, since all firewall languages support them in at least one of their NAT modes. For translation selectors, selectors presented in Table 3 must be used, with the presented constraints about their syntax. This table represents the final AFPL2 language (NAT part).

4.2 PIM Meta-model

It is necessary to clarify that MDA does not require the use of UML to specify PIMs or PSMs, it is just a recommendation. When a developer has to define a meta-model, she has to choose the meta-modelling technique: a UML-based profile (also named lightweight extension) or a MOF-based meta-model (or heavyweight extension). There are different reasons for selecting one of them [13].

AFPL2 PIM meta-model (from now, PIM) is composed of a hierarchical structure of meta-classes (Fig. 6). The PIM root element is the Policy which represents the ACL concept. An instance of the Policy meta-class could have one or more Rule children meta-classes, and zero or more SNAT and/or DNAT rules. Thus AFPL2 supports three kinds of rules: filtering (also present in AFPL), SNAT, and DNAT. SNAT and DNAT are not mandatory, but at least one filtering rule must be specified (for the default policy). Note that the left part of the figure (grey boxes) represents AFPL meta-model (without NAT), and the right part (white boxes) the NAT extension.

Table 3. Final AFPL2 model (only NAT part)

Source NAT

Translated Selector	Obligation	Dependencies	Common Syntax (Can be optimized)	Uncommon Syntax (Must be emulated)	Comments
Source IP Address	Mandatory		-Host IP -Interface name	-Identifier -Block	If the interface name is given, the interface IP is used (it could be dynamic link)

Destination NAT

Translated Selector	Obligation	Dependencies	Common Syntax (Can be optimized)	Uncommon Syntax (Must be emulated)	Comments
Destination IP Address	Mandatory		-Host IP	-Identifier	
Destination Port	Optional	Destination port must be specified in the original packet	-Number -Range: [p1,p2]	- Identifier	

Each instance of the Rule meta-class represents a condition/action rule of the ACL (this part is related to AFPL [12]). A rule can be applied to a particular interface of the underlying firewall platform (interface attribute), and with a particular direction of the flow of packets (direction attribute). These two attributes of the Rule meta-class are optional, since if no interfaces are defined, the rule is applied to all interfaces in all directions (in and out). The comment attribute is also optional and represents the documentation for a rule. In addition, the Rule meta-class has an action attribute representing the action that the firewall should take if a packet matches its condition part. Note that it must be at least one rule in a policy, stating that at least the default policy (allow all or deny all) is present.

The information regarding the condition part is represented in the Matches meta-class, which it is a child of the Rule meta-class, and the last meta-class of the hierarchy. Each Rule can have only one condition part and, for that reason, the cardinality is one. The Matches meta-class has a set of attributes representing the filtering selectors of AFPL [12]. These selectors are the fields which are considered during the filtering process, and are: source and destination IP addresses, source and destination ports, protocol, and ICMP type (only if protocol is ICMP). All these attributes have their own data types which represent the syntaxes allowed for a user. These data types are presented in the right part of Fig. 6. Some of them are enumerations and others are regular expressions.

At the same level of Rule meta-class there are the DstNATrule and SrcNATrule meta-classes (this is the part related to AFPL2). These meta-classes represent DNAT and SNAT rules (the extended part of AFPL2). Note that NAT rules are optional. Following the analysis for NAT of the previous section, a DNAT rule can be applied to an interface. Note that no information regarding direction can be modelled, since DNAT rules are always applied with incoming direction. In the same way, a SNAT rule can only be applied with outgoing direction. Again, for both kind of rules, it is

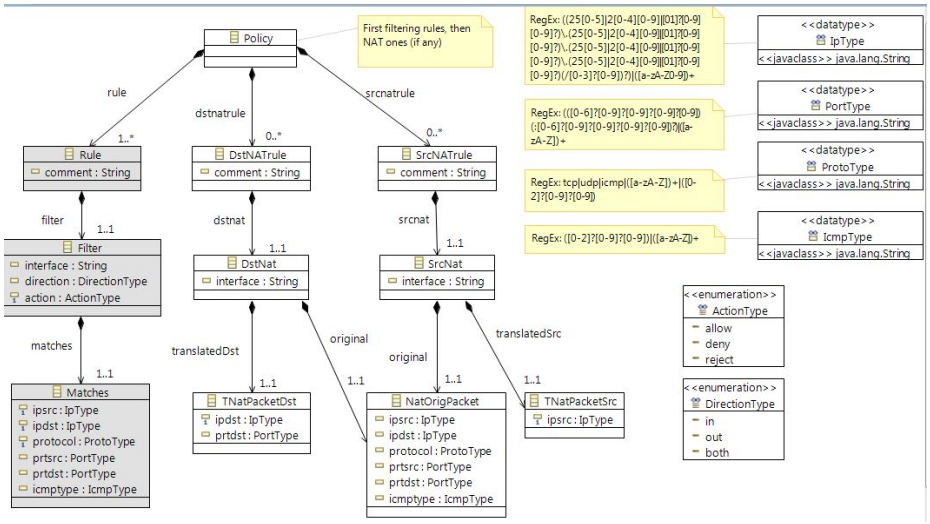

Fig. 6. AFPL2 PIM meta-model (in EMF)

possible to specify the characteristics of the packet being translated using the NatOrigPacket meta-class, which has the same attributes as the Matches meta-class, but with different cardinalities. However, translation information differs between SNAT and DNAT: their attributes represent the translation selectors explained in the previous subsection (Table 2).

Note that there is no way to represent rule priorities in the model. The reason is that rule priority is represented using the rule specification order in the PIM (i.e. rule priority is implicit in the model).

As we have noted before, for each analyzed firewall language and platform there is a set of advanced features that has not been included in the PIM meta-model. However, this if part is introduced in the PSM, then administrators can use the advances features without modifying the ACL and without sacrificing PIM simplicity. Note that AFPL2 can be compiled to any of the analyzed low-level firewall languages without any loss of information. In addition, if all non common features of each analyzed firewall platform and languages are supported in its PSM (as proposed), a lossless transformation from the low-level languages is also possible.

Furthermore there are also differences regarding how each firewall platform executes NAT (mainly before or after executing filtering rules). However, these differences must only be taken into account only in AFPL2→ACL compilers, and are not considered in this paper.

5 Conclusions

The contribution of this paper is twofold. First, we have proposed a new abstract language to represent firewall ACLs with NAT, AFPL2. To the best of our knowledge, AFPL2 is the first firewall DSL to support NAT. With AFPL2 an administrator is able to model the vast majority of features present in any of the analyzed firewall

platforms and languages (which are market-leaders). However, for an advanced administrator, there could be features that cannot be modelled with AFPL2 alone.

Second, AFPL2 is used as the PIM for a framework heavily based on the concepts of MDA extended for model consistency diagnosis stages. The framework can incorporate these advanced features of all the analyzed firewall languages in another (lower) modelling level, allowing administrators to use the features of a particular low-level firewall language not present in AFPL2. This MDA-based framework is extensible by end-users, in the sense that more concepts can be added to the meta-models, as well as modifications to the transformations between them, in order to represent more features and/or low-level firewall platforms and languages. In fact, we have identified the features not modelled in the PIM and propose to model them in the PSM of each firewall platform as a topic for future research.

In future works, we pretend to develop the full framework with automatic transformations and ACL generation.

Acknowledgements

This work has been partially funded by Spanish Ministry of Science and Education project under grant DPI2006-15476-C02-01, and by FEDER (under ERDF Program). Many thanks to T. Reina and J. Peña for their useful comments on early versions of the paper.

References

1. Pozo, S., Ceballos, R., Gasca, R.M.: Model Based Development of Firewall Rule Sets: Diagnosing Model Faults. Information and Software Technology Journal 51(5), 894–915 (2009)
2. Al-Shaer, E., Hamed, H.H.: Modeling and Management of Firewall Policies. IEEE eTransactions on Network and Service Management 1(1) (2004)
3. Pozo, S., Ceballos, R., Gasca, R.M.: A Heuristic Polynomial Algorithm for Local Inconsistecy Diagnosis in Firewall Rule Sets. In: International Conference on Security and Cryptography (SECRYPT), Porto, Portugal (2008)
4. Wool, A.: A quantitative study of firewall configuration errors. IEEE Computer 37(6), 62–67 (2004)
5. Taylor, D.E.: Survey and taxonomy of packet classification techniques. ACM Computing Surveys 37(3), 238–275 (2005)
6. Bartal, Y., Mayer, A., Nissim, K., Wool, A.: Firmato: A Novel Firewall Management Toolkit. ACM Transactions on Computer Systems 22(4), 381–420 (2004)
7. Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder Specification Language Workshop on Policies for Distributed Systems and Networks (POLICY), HP Labs Bristol, UK, pp. 29–31 (2001)
8. OASIS eXtensible Access Control Markup Language (XACML), `http://www.oasis-open.org/committees/xacml/`
9. Moore, B., Ellesson, E., Strassner, J., Westerinen, A.: Policy Core Information Model (PCIM), IETF RFC 3060 (2001)
10. Rule Markup Language (RuleML), `http://www.ruleml.org/`
11. Simple Rule Markup Language (SRML): A General XML Rule Representation for Forward-chaining Rules, ILOG S.A (2001)
12. Pozo, S., Ceballos, R., Gasca, R.M.: AFPL, An Abstract Language Model for Firewall ACLs. In: 8th International Conference on Computational Science and Its Applications (IC-CSA), Perugia, Italy. Springer, Heidelberg (2008)

13. Desfray, P.: UML Profiles versus Metamodeling Extensions... an Ongoing Debate. In: COM 2000, proceedings of the First Workshop on UML in the .COM Enterprise: Modeling CORBA, Components, XML/XMI and Metadata (2000)
14. Bartal, Y., Mayer, A., Nissim, K., Wool, A.: Firmato: A Novel Firewall Management Toolkit. ACM Transactions on Computer Systems 22(4), 381–420 (2004)
15. Zhang, B., Al-Shaer, E., Jagadeesan, R., Riely, J., Pitcher, C.: Specifications of a High-level Conflict-free Firewall Policy Language for Multi-domain Networks. In: ACM Symposium on Access Control Models and Technologies (SACMAT), Sophia Antipolis, France, pp. 185–194 (2007)
16. Srisuresh, P., Holdrege, M.: RFC 2663: IP Network Address Translator (NAT) Terminology and Considerations. IETF (August 1999)
17. Basin, D., Dorser, J., Lodderstedt, T.: Model Driven Security: from UML Models to Access Control Infrastructures. ACM Transactions on Software Engineering and Methodology 15(1), 39–91 (2006)
18. OMG. MDA guide version 1.0. Technical Report omg/2003-05-01, OMG (May 2003)
19. Hailpern, B., Tarr, P.: Model-driven development: The good, the bad, and the ugly. IBM Systems Journals 45(3), 451–462 (2006)

Appendix I

This analysis is related to NAT rules of market-leader firewall languages, and refers to the conditions that may be matched against a packet that arrives at the firewall, and how it must be translated.√ indicates that a selector is supported, and x that is not supported.

Table 4. Netfilter IPTables analysis

NAT Type	SNAT / MASQ	DNAT / PORT FW
Src IP Address	*Opt*	*Opt*
Translated Src IP Address	√* *-IP*	*x*
Src Port	*Opt*	*Opt*
Translated Src Port	*Opt*** *- Number, range*	*x*
Dst IP Address	*Opt*	*Opt*
Translated Dst IP Address	*x*	√* *-IP*
Dst Port	*Opt*	*Opt*
Translated Dst Port	*x*	*Opt* *- Number, range*
Protocol	*Opt*	*Opt*
Interface	*Opt (Outgoing)*	*Opt (Incoming)*
Comments	**It is not possible to do load balancing in K >=2.6.11* ** If port is given, NAPT is done instead of SNAT	**It is not possile to do load balancing in K >=2.6.11*

Table 5. Cisco PIX analysis

NAT Type	Dynamic NAT and PAT	Dynamic Policy NAT and PAT	Static NAT and PAT	Policy Static NAT and PAT
Address Pool	√	√	*x*	*x*
Src IP Address	√ Collection using the pool	√ Collection using the pool	√	√
Translated Src IP Address	√ -IP, Block -Range, Collection	√ -IP	√ -IP, Interface IP	√ -IP, Interface IP
Src Port	*x*	*Opt*	*x*	*Opt*
Translated Src Port	*Automatic* *(only for PAT)*	*Automatic* *(only for PAT)*	*Opt* *(only for PAT)*	*Opt* *(only for PAT)*
Dst IP Address	*x*	√ (one or more)	*x*	√ (one or more)
Translated Dst IP Address	*x*	*x*	√/*x* *(bidi)*	√/*x* *(bidi)*
Dst Port	*x*	*Opt*	*x*	*Opt*
Translated Dst Port	*x*	*x*	*Opt* *(only for PAT)*	*Opt* *(only for PAT)*
Protocol	*x*	√	TCP or UDP. Only for PAT	TCP or UDP. Only for PAT
Interface (outbound)	√ Only in less secure to more secure interface connections	√ Only in less secure to more secure interface connections	*x*	*x*
Interface (inbound)	*x*	*x*	√	√
Connection Settings	Misc options for TCP and UDP	Misc options for TCP and UDP	Misc options for TCP and UDP	Misc options for TCP and UDP

Table 6. OpenBSD Packet Filter analysis

Src IP Address	√	√
Translated Src IP Address	√ *- IP Collection*	*x*
Src Port	*Opt*	*Opt*
Translated Src Port	√	*x*
Dst IP Address	√	√
Translated Dst IP Address	*x*	√*
Dst Port	*Opt*	*Opt*
Translated Dst Port	*x*	*Opt*
Src/Dst Protocol	*x*	√
Interface	√	√
Comments	Bidirectional connections are possible	* If multiple translated dst IPs are specified, load-balancing is

Table 7. FreeBSD IPFirewall analysis

NAT Type	Outgoing connections	Incoming connections
Src IP Address	*Opt*	*Opt*
Translated Src IP Address	√	*x*
Src Port	*Opt*	*Opt*
Translated Src Port	*x*	*x*
Dst IP Address	*Opt*	*Opt*
Translated Dst IP Address	*x*	√***
Dst Port	*Opt*	*Opt* *(Port or range)*
Translated Dst Port	*Opt**	*Opt**
Src/Dst Protocol	*Opt***	*Opt***
Interface	√	√
Comments	* Mandatory if Dst Port is specified ** Mandatory if ports are specified	* Mandatory if Dst Port is specified, but could be the same by default ** Mandatory if ports are specified *** If multiple translated dst IPs are specified, load-balancing is accomplished

Table 8. OpenBSD IPFilter analysis

NAT Type	Basic NAT	Port Redirection
Src IP Address	√	*Opt*
Translated Src IP Address	√ *[Only IP or Block]*	*x*
Src Port	*Opt*	*Opt*
Translated Src Port	*Opt**	*x*
Dst IP Address	*Opt*	√* *[Only IP]*
Translated Dst IP Address	*x*	√*(host only)*
Dst Port	*Opt*	*Opt*
Translated Dst Port	*x*	√
Src/Dst Portocol	*Opt*	*Opt*
Interface	√	√
Comments	* If port is given, NAPT is done instead of SNAT	* If multiple dst IPs are specified, round-robbin load-balancing is accomplished

Table 9. Checkpoint FW-1 analysis

NAT Type	All types
Src IP Address	*Opt*
Translated Src IP Address	*Opt*
Src Port	*Opt*
Translated Src Port	*Opt*
Dst IP Address	*Opt*
Translated Dst IP Address	*Opt*
Dst Port	*Opt*
Translated Dst Port	*Opt*
Src/Dst Protocol	*Opt*
Interface	*x*