POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Rappresentazione astratta delle funzionalità di controlli di sicurezza

Relatori
prof. Antonio Lioy
prof. Cataldo Basile

Candidato
Nicolò Noceti

Anno accademico 2018-2019

Indice

Capitolo 1

Introduzione

La virtualizzazione è una tecnologia che consente di creare servizi sfruttando risorse tradizionalmente vincolate all'hardware. Permette di sfruttare tutte le capacità di una macchina fisica distribuendo le funzionalità tra più utenti o ambienti.

Si ritiene che il concetto di virtualizzazione abbia le sue origini alla fine degli anni '60, quando IBM investì molto tempo e sforzi nello sviluppo di solide soluzioni di condivisione delle risorse nel tempo. La condivisione delle risorse del tempo si riferisce all'uso condiviso delle risorse del computer tra un grande gruppo di utenti, con l'obiettivo di aumentare l'efficienza sia degli utenti sia delle costose risorse del computer che condividono. Questo modello ha rappresentato un importante passo avanti nella tecnologia informatica: il costo di fornitura della capacità di elaborazione è diminuito considerevolmente ed è diventato possibile per le organizzazioni, e anche le persone, utilizzare un computer senza possederne effettivamente uno. Ragioni simili guidano la virtualizzazione per standard industriali di settore: la capacità in un singolo server è così grande che è quasi impossibile per la maggior parte dei carichi di lavoro utilizzarla in modo efficace. Il modo migliore per ottimizzare l'utilizzo delle risorse e allo stesso tempo semplificare la gestione dei data center è attraverso la virtualizzazione. Oggi i data center utilizzano tecniche di virtualizzazione per realizzare l'astrazione dell'hardware fisico, creare grandi aggregati di risorse logiche costituite da CPU, memoria, dischi, archiviazione di file, applicazioni, reti e offrire tali risorse ad utenti o clienti sotto forma di macchine virtuali agili, scalabili e consolidate. Anche se la tecnologia e i casi d'uso si sono evoluti, il significato principale della virtualizzazione rimane lo stesso: consentire ad un ambiente informatico di eseguire più sistemi indipendenti contemporaneamente.

Le tecnologie che hanno consentito la virtualizzazione, come gli hypervisor, sono state sviluppate al fine di fornire a più utenti l'accesso simultaneo a computer. Tuttavia, inizialmente, il problema della presenza di più utenti su un unico computer è stato affrontato scegliendo soluzioni diverse dalla virtualizzazione. Tra queste, la condivisione temporale, che isolava gli utenti all'interno dei sistemi operativi e che ha portato inavvertitamente alla nascita di altri sistemi operativi, come UNIX, fino ad arrivare a Linux. Tutto questo, mentre la virtualizzazione restava una tecnologia di nicchia scarsamente utilizzata. Negli anni '90 gran parte delle aziende utilizzava server fisici e stack IT associati ad un singolo fornitore, che non consentivano l'esecuzione delle applicazioni esistenti sull'hardware di altri fornitori. Le aziende iniziarono ad aggiornare i propri ambienti IT con *commodity server*, sistemi operativi e applicazioni meno costosi di vari fornitori. Tuttavia, erano vincolate ad hardware fisici scarsamente utilizzati, poiché ciascun server poteva eseguire soltanto un'attività. A questo punto ha iniziato a diffondersi la virtualizzazione. Era la soluzione naturale a due problemi: permetteva alle aziende di ottenere partizioni sui propri server e di eseguire le applicazioni esistenti su più tipologie e versioni di sistemi operativi. L'utilizzo dei server è stato reso più efficiente, in alcuni casi abbandonato, riducendo così i costi associati all'acquisto, alla configurazione, al raffreddamento e alla manutenzione. L'ampia applicabilità della virtualizzazione ha contribuito a ridurre il blocco del venditore e ha posto le basi del cloud computing. Oggi è utilizzata da moltissime aziende, ed è spesso necessario disporre di software specifici per la gestione della virtualizzazione che consentano di monitorare l'ambiente.

I progressi nelle tecnologie di cloud computing e virtualizzazione delle funzioni di rete (Network Functions Virtualization) hanno reso possibile fornire servizi di rete tramite funzioni di servizio

virtuale in esecuzione su server cloud. È anche possibile esternalizzare queste funzioni di servizio virtuale a fornitori di soluzioni di terze parti. Questo modello di fornitura dei servizi basato su cloud offre numerosi vantaggi come il risparmio sui costi e l'utilizzo flessibile ed efficiente delle risorse. Questo modello di servizio è particolarmente utile per fornire agli utenti servizi di sicurezza di rete. Ad esempio, nello scenario di un attacco DDoS (Distributed Denial of Service), è possibile in modo rapido e flessibile rispondere all'intenso traffico di attacchi aumentando dinamicamente il numero di istanze di mitigazione DDoS. Inoltre, questo modello di servizio di sicurezza basato su cloud facilita l'implementazione di varie funzioni di sicurezza sviluppate da più fornitori. Questo è adatto per soddisfare le crescenti esigenze dei sistemi di rete aziendali di integrare queste funzioni di sicurezza per la creazione di sistemi più sicuri. Le funzioni di sicurezza di rete (Network Security Function) sviluppate da diversi fornitori hanno interfacce diverse per la loro configurazione e gestione perché non esiste uno standard industriale di interfacce per le NSF. Questa eterogeneità introduce complessità nella gestione delle NSF di più fornitori, con conseguente aumento dei costi di gestione. Pertanto, la standardizzazione è essenziale per implementare con successo le NSF offerte da vari fornitori. Di recente, alcune attività di standardizzazione hanno iniziato un processo di sviluppo come il gruppo di lavoro Internet Engineering Task Force Interface to Network Security Functions (IETF I2NSF) per soddisfare queste esigenze.

Il gruppo di lavoro IETF I2NSF sviluppa una serie di modelli informativi e modelli di dati standard che sono la chiave per costruire le interfacce standard dell'architettura I2NSF. Sulla base dei modelli definiti da I2NSF questa tesi ha come punto di partenza la realizzazione di un modello generico delle capacità di sicurezza di una rete. Il modello viene realizzato utilizzando la rappresentazione grafica UML ed alcuni pattern di design per l'ottimizzazione delle funzionalità espressive del modello. Dal modello viene ricavata la rappresentazione XML schema per la validazione di NSF configurate secondo capacità arbitrarie. Infatti l'obiettivo primario della tesi è trovare un metodo per rappresentare, in modo generico, le funzionalità che vengono offerte dai servizi di sicurezza. Avendo a disposizione la rappresentazione generica delle capacità dei dispositivi, si possono svolgere diverse attività essenziali sia nell'ambito della ricerca che nel campo dell'amministrazione della sicurezza di rete. Ad esempio, si possono confrontare in maniera formale e precisa dispositivi di sicurezza diversi per verificare se hanno caratteristiche in comune per, ad esempio, poter applicare le stesse politiche di sicurezza. Per verificare l'espressività del modello si effettueranno test con dispositivi di filtering tra cui l'infrastruttura di packet filtering integrata nei kernel Linux iptables.

La definizione di modello di dati generico è simile alla definizione di un linguaggio naturale. Ad esempio, un modello di dati generico può definire tipi di relazione come una "relazione di classificazione", essendo una relazione binaria tra un singolo elemento ed un tipo di elemento (una classe) ed una "relazione parte-intero", essendo una relazione binaria tra due elementi, una con il ruolo della parte, l'altra con il ruolo del tutto, indipendentemente dal tipo di elementi che sono correlati. Dato un elenco estensibile di classi, ciò consente di classificare ogni singolo elemento e di specificare relazioni per ogni singolo oggetto. Standardizzando un elenco estensibile di tipi di relazioni, un modello di dati generico consente l'espressione di un numero illimitato di tipi di dato e si avvicinerà alle capacità dei linguaggi naturali. I modelli di dati convenzionali, d'altra parte, hanno un ambito di dominio fisso e limitato, poiché l'istanza di tale modello consente solo espressioni di tipi di dati predefiniti nel modello. I modelli di dati generici sono sviluppati come approccio per risolvere alcune carenze dei modelli di dati convenzionali. Ad esempio, diversi modellatori di solito producono diversi modelli di dati convenzionali dello stesso dominio. Ciò può comportare difficoltà nel riunire i modelli creati da persone diverse ed è un ostacolo allo scambio e all'integrazione dei dati. Questa differenza è attribuibile a diversi livelli di astrazione nei modelli e differenze nei tipi di dati che possono essere istanziati. I modellisti devono comunicare e concordare alcuni elementi che devono essere resi in modo più concreto, al fine di rendere le differenze meno significative.

Rendere generico un modello, anche in un'area di applicazione specialistica, può presentare difficoltà. I requisiti essenziali di una descrizione generica devono essere identificati per primi e deve essere istituito un quadro adeguato per fornire la flessibilità necessaria per consentire a una serie di esigenze più specifiche di soddisfare tale rappresentazione generica. Potrebbe essere necessario rappresentare un sistema a diversi livelli di dettaglio nelle diverse fasi di un progetto di design e ciò deve essere possibile anche con l'approccio generico. Ciò significa che possono essere

necessari sottomodelli, che rappresentano parti specifiche dell'intero sistema fisico, a diversi livelli di complessità, che vanno dalle forme puramente funzionali nella fase iniziale a modelli altamente dettagliati e pienamente convalidati nelle fasi successive del progetto. Idealmente, le strutture per i diversi livelli del modello saranno direttamente correlate e i modelli a diverse risoluzioni formeranno un gruppo integrato. La relazione tra i diversi livelli di ciascun sotto-modello all'interno della struttura generica deve essere pienamente compresa dagli utenti. Il vantaggio più importante dell'approccio generico è probabilmente un processo di sviluppo più rapido e meno costoso per i nuovi modelli rispetto all'approccio tradizionale, che prevede lo sviluppo, su base una tantum, di un nuovo modello specifico per ogni nuova attività di progettazione. Altri vantaggi sono probabilmente dovuti allo sviluppo ed all'applicazione di un modello generico che richiede un approccio più sistematico e rigoroso ai problemi di convalida del modello, unitamente a una migliore documentazione. Avere un modello generico permette di poter ragionare ad un livello molto astratto tale che renda visibile per ogni elemento le sue funzionalità e relazioni. Il modello generico fornisce indicazioni sul comportamento del sistema.

Sulla base di questi modelli standard di informazioni e di dati, l'architettura I2NSF è in grado di fornire un ambiente di servizio di sicurezza efficiente e flessibile guidato da politiche di sicurezza. Il paradigma Software-Defined Networking (SDN) consente cambiamenti dinamici e flessibili nel comportamento della rete controllando e gestendo le configurazioni delle risorse di rete, come switch e router, a livello di codice. Questa capacità rende possibile applicare alcune regole di filtraggio dei pacchetti agli switch controllando le loro regole di inoltro dei pacchetti. In particolare, gli switch applicano semplici regole di filtraggio dei pacchetti che possono essere tradotte nelle loro regole di inoltro dei pacchetti, mentre le NSF applicano regole di sicurezza relative alle capacità di sicurezza disponibili tra le NSF. Pertanto, se gli switch possono prendere decisioni su alcuni pacchetti ricevuti in base alle loro regole di inoltro dei pacchetti programmate da un controller di switch, possiamo evitare la latenza non necessaria per i pacchetti presi da una NSF per un'attività di ispezione che richiede tempo. Inoltre, poiché tutti i pacchetti non passano necessariamente attraverso una NSF, possiamo ridurre la possibilità di congestione in una NSF.

I2NSF definisce due tipi di interfacce a due livelli differenti: un livello di servizio ed un livello di capacità. Il livello di servizio specifica come le politiche di sicurezza di un cliente possono essere espresse ad un responsabile della sicurezza. Il livello di capacità specifica come controllare e monitorare le funzioni di sicurezza basate sul flusso (NSF) a livello di implementazione funzionale. Le politiche sull'interfaccia del livello di servizio non si preoccupano di quali NSF vengano utilizzate per applicare le politiche. Potrebbero esserci più NSF per applicare un criterio del livello di servizio. Le politiche sull'interfaccia del livello di capacità riguardano specifiche NSF. Per esprimere le politiche di sicurezza basate sul flusso viene utilizzato il paradigma evento-condizione-azione (ECA) sia sull'interfaccia del livello di servizio sia sull'interfaccia del livello di capacità.

Il progetto Simplified Use of Policy Abstractions, SUPA, definisce un modello informativo per rappresentare le politiche usando un framework estensibile che è indipendente da linguaggio, protocollo, repository e livello di astrazione del contenuto e del significato di una qualsiasi politica. Il modello informativo definito da SUPA è composto da due serie di elementi principali: un framework generico per la definizione del concetto di politica chiamato SUPA Generic Policy Information Model (GPIM) ed un framework per la definizione di un modello di policy che utilizza il paradigma event-condition-action chiamato SUPA ECA Policy Rule Information Model (EPRIM). Quest'ultimo estende i concetti del GPIM.

Da analisi in letteratura, confermate durante questa tesi, si evince che i linguaggi messi a disposizione dai fornitori di dispositivi per esprimere politiche di sicurezza sono eterogenei, quindi possono esserci incompatibilità o incomprensioni nell'utilizzo di dispositivi per controlli di sicurezza prodotti da fornitori differenti; quindi per esprimere politiche di sicurezza a dispositivi di produttori differenti è necessario conoscere i dettagli dei linguaggi di ogni produttore di funzionalità di sicurezza. Per questo motivo, durante questa tesi ci si è occupati di trovare un metodo per esprimere le politiche di sicurezza con una rappresentazione generica basata sulle capacità di sicurezza del dispositivo che fosse anche indipendente dalla tecnologia e da eventuali caratteristiche determinate dal produttore. Di conseguenza è sorto il problema di trovare un sistema per la traduzione nel linguaggio specifico del dispositivo.

Le trasformazioni di modello vengono utilizzate per automatizzare le attività di progettazione permettendo di passare da un modello astratto, indipendente dalla tecnologia, ad uno concreto, dipendente dalla piattaforma. Questo concetto è espresso dal paradigma di model driven transformation. Il lavoro di questa tesi sviluppa un tool di conversione di policy di sicurezza per le NSF da un modello generico ad un modello specifico. Questo permette ad un utente di non dover conoscere il "linguaggio specifico" col quale verrà espressa ogni singola regola, dato che potrà fornire le regole in un "linguaggio generico" ad un controllore, il quale si occuperà di tradurre le regole nel "linguaggio specifico" della NSF che le applicherà.

Questa tesi, seguendo i principi appena esposti, porta come risultato quanto segue. Una NSF mette a disposizione determinate capacità di sicurezza. Il modello informativo delle capacità (CapIM) definisce una NSF come un aggregato di capacità di sicurezza. Quindi per rendere generale il concetto si può considerare una NSF generica come un contenitore delle capacità di sicurezza che le sono state assegnate. In questo modo si genera una relazione tra una NSF ed una capacità di sicurezza. Per poter gestire e manipolare ogni potenzialità di cui è composta una funzionalità di sicurezza è stata realizzata una rappresentazione astratta delle capacità di sicurezza in un modello dei dati (CapDM), utilizzando il linguaggio di modellazione grafico UML. In tale modello, una capacità di sicurezza è rappresentata come una classe che mette a disposizione degli attributi. Definendo la tipologia di dati che una capacità può gestire è possibile specificare eventuali restrizioni o si possono determinare i valori che può rappresentare, ad esempio utilizzando espressioni regolari o enumerazioni. Ogni capacità di sicurezza è stata raggruppata in sottoinsiemi di competenza, ed eredita caratteristiche in funzione della categoria in cui si trova. L'elemento più astratto delle capacità di sicurezza è una classe chiamata "SecurityCapability" che permette di definire attributi o funzionalità comuni a tutte le capacità di sicurezza che ereditano da essa.

Delineato il metodo per definire le capacità di sicurezza di un dispositivo o di una funzione di sicurezza è stato progettato un sistema per la generazione di linguaggi astratti per controlli di sicurezza a partire da un'istanza del CapDM che descrive il dispositivo stesso. Tale linguaggio permette di esprimere politiche di sicurezza utilizzando una sintassi generica, usufruendo della semantica definita dalle capacità di sicurezza dal CapDM o da eventuali dettagli specificati durante l'assegnazione delle capacità alla NSF. Per questo scopo e per lo sviluppo del linguaggio generico è stato utilizzato il linguaggio XML. Le politiche di sicurezza espresse nel linguaggio generico della NSF non possono essere applicate dai dispositivi perché non espresse nel loro linguaggio specifico, di conseguenza è stato ideato un metodo di traduzione dal linguaggio generico al linguaggio concreto della NSF. Tale sistema utilizza un meccanismo di traduzione basato sulla definizione della sintassi e della semantica, specifiche di una NSF, al momento dell'assegnazione di ogni capacità di sicurezza. In questo modo lo strumento realizzato durante il lavoro di tesi, permette la generazione di un file di configurazione contenente le regole, nel linguaggio concreto della NSF, della politica espressa nel linguaggio generico della stessa.

Nel capitolo 2 viene presentato il background relativo agli argomenti affrontati per la progettazione e lo sviluppo dell'elaborato, con approfondimenti riguardo le caratteristiche più utilizzate. Nel capitolo 3 vengono presentate le tecnologie ed i lavori correlati che formano la base dello sviluppo del lavoro di progettazione del modello. Il capitolo 4 definirà in modo più dettagliato il problema che si vuole affrontare, saranno esposte le tecnologie ed i criteri utilizzati e si renderà nota l'architettura progettata per la soluzione. Il capitolo 5 mostra il modello informativo delle capacità ed il modello dei dati realizzati. Il capitolo 6 analizza un caso concreto di tecnologia di filtering applicato all'architettura sviluppata, e verifica che la rappresentazione proposta permette di coprire le funzionalità del caso concreto. Il capitolo 7 illustra le conclusioni derivate da questo lavoro di tesi. Inoltre saranno specificate nelle appendici i manuali utente e programmatore per ogni strumento sviluppato nell'ambito della tesi.

Capitolo 2

Concetti di base

2.1 Rappresentazioni grafiche - UML

L'Unified Modelling Language (UML) è un linguaggio universale di modellazione standardizzato da Object Management Group (OMG) e International Organization for Standardization (ISO).

L'UML è un linguaggio comunemente usato per la modellazione di processi, l'analisi, la progettazione e l'implementazione di sistemi. Permette di descrivere, specificare, progettare e documentare processi, strutture o comportamenti di elementi di un sistema software. L'UML è gestito da OMG. La prima versione fu creata da i "three Amigos", Gady Booch (creatore del metodo booch), Ivar Jacobson (ingegnere del software), Jim Rumbaug (Object-Modeling Technique)[1]. L'OMG è un consorzio internazionale di standard tecnologici senza scopo di lucro, aperto nel 1989. Gli standard OMG sono guidati da fornitori, utenti finali, istituzioni accademiche ed agenzie governative. Le task force OMG sviluppano standard di integrazione aziendale. Gli standard di modellazione di OMG, tra cui UML e MDA (model driven architecture) consentono progettazione visiva, esecuzione e manutenzione di software ed altri processi. Numerose specifiche OMG sono state adottate integralmente dall'ISO come standard ISO . ISO è una organizzazione internazionale indipendente e non governativa. ISO definisce le specifiche per prodotti, servizi, sistemi per garantire qualità, sicurezza ed efficienza [1].

Il linguaggio UML è caratterizzato dai diagrammi. Un diagramma UML è una parziale rappresentazione grafica (vista) di un modello del sistema preso in oggetto. Un diagramma contiene elementi grafici (simboli) che rappresentano gli elementi del modello per il sistema di progettazione. Ad oggi si è arrivati alla versione di UML 2.5.1 adottata nel dicembre 2017 [2].

I tipi di diagrammi definiti da UML 2.5 sono suddivisibili in due macro-categorie [2]:

- diagrammi strutturali: con questo tipo di diagrammi si mette in mostra la struttura statica di un sistema e delle sue parti in diversi livelli di astrazione ed implementazione, inoltre viene rappresentato anche come le entità sono relazionate tra di loro. Gli elementi in un diagramma strutturale rappresentano i concetti significativi di un sistema e possono includere definizioni astratte, reali e di implementazione.

  I grafici che fanno parte di questa categoria sono:

  - class diagram;
  - object diagram;
  - package diagram;
  - composite structure diagram;

[1] https://www.uml-diagrams.org/

[2] https://www.uml-diagrams.org/uml-25-diagrams.html

  - component diagram;
  - deployment diagram;
  - profile diagram.

- diagrammi comportamentali: questa categoria di diagrammi viene utilizzata per mostrare le dinamiche di comportamento tra gli oggetti nel sistema, che possono descrivere una serie di cambiamenti nel tempo riguardanti l'ambiente, rappresentando inoltre le varie necessità legate al sistema, come il flusso di controllo ed i cambiamenti interni allo stato. I grafici appartenenti a questa categoria sono:
  - use case diagram;
  - activity diagram;
  - state machine diagram;
  - interaction diagram; questa tipologia di diagrammi può essere considerata come una categoria a se, della quale fanno parte i seguenti diagrammi:
    * sequence diagram;
    * sommunication diagram;
    * timing diagram;
    * interaction overview diagram.

Esistono delle sottocategorie dei precedenti diagrammi che non sono state riportate perchè non sono parte della norma di UML 2.5.

Di seguito riporto più nel dettaglio il principale diagramma utilizzato nello sviluppo di questa tesi.

2.1.1 Class diagram

Questo tipo di diagramma descrive la struttura di un sistema mostrando le classi e le relazioni tra esse. Una classe modella un insieme di entità, chiamate istanze della classe, che condividono le stesse caratteristiche, limitazioni e semantiche (ovvero gli stessi attributi, associazioni e operazioni). La classe è un tipo di classificatore il cui scopo è quello di dare una classificazione di oggetti e di indicare le caratteristiche che ne descrivono la struttura ed il comportamento. Una classe si distingue dalle altre classi da un nome e un insieme di proprietà ed operazioni. Le proprietà rappresentano le caratteristiche strutturali di una classe e possono essere rappresentate mediante gli attributi. Un attributo descrive una proprietà con una riga di testo, indicando il nome ed altre informazioni opzionali come la visibilità, il tipo, la molteplicità ed il valore di default. La definizione di un attributo è locale alla classe in cui è definito, in questo modo un'altra classe può avere un attributo col medesimo nome, ma di diversa definizione. Una classe può essere raffigurata con un rettangolo suddiviso in tre parti distinte: nella parte più in alto viene indicato il nome, nella parte centrale le proprietà ed in quella più in basso la lista delle operazioni come presentato in Figura 2.1.

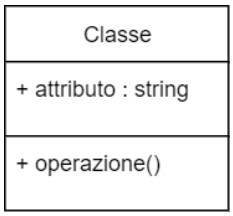

Figura 2.1. Esempio di classe.

Nel caso in cui, per praticità, si vogliano omettere gli attributi e le operazioni, basterà rappresentare la classe solo con il simbolo del rettangolo (o box) con il nome al suo interno.

Due classi possono essere messe in relazione tra di loro mediante simbologia definita. Di seguito riporto le principali tipologie di relazioni:

- associazione: un'associazione è una relazione tra due o più classificatori ed indica anche una connessione tra le loro istanze. È rappresentata da una linea continua che collega i due classificatori e può essere bidirezionale, come in Figura 2.3 o avere una singola navigabilità, come in Figura 2.2, ovvero può essere orientata dalla classe sorgente a quella destinazione.

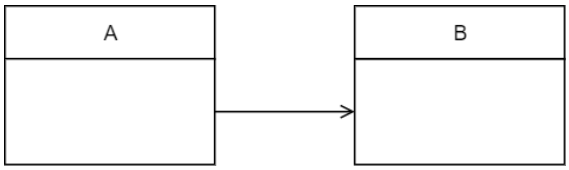

Figura 2.2. Esempio di associazione monodirezionale.

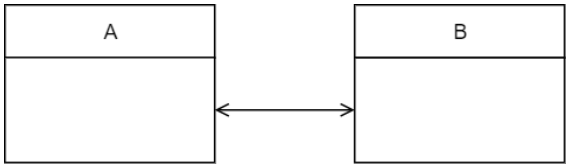

Figura 2.3. Esempio di associaizone bidirezionale.

  Assieme all'arco dell'associazione vengono indicati il nome dell'associazione stessa ed, agli estremi, le relative molteplicità. Quest'ultima è l'indicazione di quanti oggetti possono partecipare alla relazione e viene indicata con la seguente terminologia:

  – [1] indica la partecipazione all'associazione di esattamente un solo oggetto;
  – [0..1] indica la partecipazione opzionale di un oggetto;
  – [*] o [0..*] indica la partecipazione di zero o più oggetti;
  – [1, *] indica la partecipazione di almeno un oggetto.

  In generale le molteplicità si possono definire indicando gli estremi inferiori e superiori di un intervallo (per esempio [3,5]);

- aggregazione: l'aggregazione è un tipo particolare di associazione che esprime il concetto "è parte di". Questo tipo di relazione si ha quando un insieme è relazionato con le sue parti. In parole povere è un'associazione che esprime il concetto "A ha un B" e quindi richiede che B possa essere una parte di A. In letteratura l'aggregazione è infatti indicata, in modo semplicistico, come "relazione intero-parte" e si rappresenta con un diamante vuoto dalla parte della classe contenitore come in Figura 2.4;

- composizione: la composizione è un caso particolare di aggregazione in cui le parti esistono in funzione del "tutto", ovvero la parte può essere inclusa al massimo in un intero e solo l'oggetto intero può creare e distruggere le sue parti. Se un componente viene distrutto, normalmente vengono distrutti tutti i componenti allo stesso tempo. La composizione si raffigura utilizzando un diamante pieno dalla parte della classe contenitore come in Figura 2.5;

- generalizzazione: la generalizzazione è una relazione che collega un elemento più generico ad uno più specifico, ovvero descrive la relazione di ereditarietà tra le varie classi di un progetto, come in Figura 2.6. L'elemento più specifico deve essere pienamente consistente con quello più generico, cioè ne eredita tutte le caratteristiche, ma può definirne ulteriori attributi, operazioni e relazioni;

- classe di associazione: la classe di associazione è un elemento che possiede caratteristiche sia di una associazione sia di una classe. Può essere vista come una associazione con proprietà di classe, o come una classe con proprietà di associazione. Questo elemento non solo connette due classificatori ma ne definisce un insieme di caratteristiche che appartengono alla relazione stessa e non ad uno specifico classificatore. Viene rappresentato come in Figura 2.7;

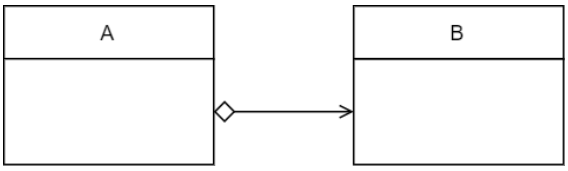

Figura 2.4. Esempio di aggregazione.

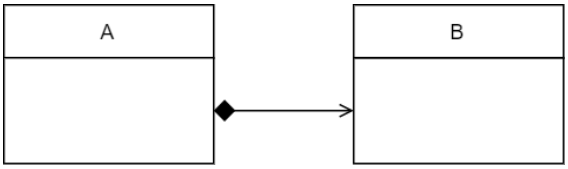

Figura 2.5. Esempio di composizione.

- classe <<Enumeration>>: la classe enumerazione è un tipo particolare di classe, essa lavora come container di "enumeration literal" cioè di elementi stringa ben definiti e prefissati alla creazione dell'enumerazione. In questo modo si possono creare combinazioni per permettere ad una variabile di ottenere solo determinati valori. Questa particolare classe è rappresentata da un rettangolo diviso in due parti, come in Figura 2.8, la prima contenente il nome dell'enumerazione e la seconda contenente tutti i valori dell'enumerazione stessa.

2.2 Rappresentazioni dei dati

2.2.1 XML

Extensible Markup Language (XML) è un semplice formato basato su testo per rappresentare informazioni strutturate: documenti, dati, configurazioni, libri, transazioni, fatture e molto altro [3]. È stato derivato da un vecchio formato standard chiamato SGML [4], al fine di essere più adatto per l'uso attraverso il Web [5]. XML è uno standard approvato da W3C che fornisce una sintassi generica utilizzata per contrassegnare i documenti con tag, leggibile sia dall'uomo sia dalla macchina [6].

XML è una specifica per scopi generici per la definizione di linguaggi di markup, quindi un linguaggio basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento di testo. È estensibile perché XML consente di creare nuovi elementi di markup. Il linguaggio XML è un formato comunemente usato quando si condividono dati. Lo scopo principale di XML è semplificare la condivisione e la distribuzione di informazioni tra vari sistemi indipendenti. Questo formato è basato su testo, semplice e molto flessibile, quindi per creare o modificare documenti XML è sufficiente un editor di testo. Oggi viene utilizzato non solo per la memorizzazione di informazioni ma anche per la serializzazione dei dati, come formato di trasporto dei dati e persino per i database.

XML ha una serie di vantaggi rispetto a molti altri formati [5]. Per ogni particolare scenario, si potrebbe essere in grado di trovare un formato migliore, ma poi bisognerebbe includere i costi di conversione ed elaborazione del formato, dell'istruzione e dello strumento di modifica e ricerca specifico di XML che ora sono ampiamente disponibili. Alcuni dei vantaggi di XML includono:

- ridondanza: il markup XML è molto dettagliato. Ad esempio, ogni tag di fine deve essere fornito. Ciò consente al computer di rilevare errori comuni come l'annidamento errato;
- autodescrittivi: la leggibilità di XML e la presenza di nomi di elementi e attributi in XML significa che le persone che guardano un documento XML possono spesso ottenere un vantaggio sulla comprensione del formato e agevola l'individuazione di eventuali errori;

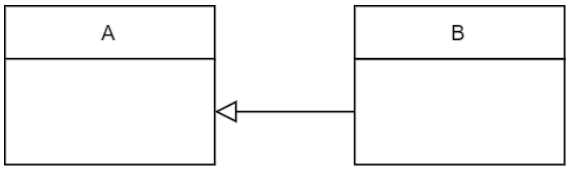

Figura 2.6. Esempio di generalizzazione.

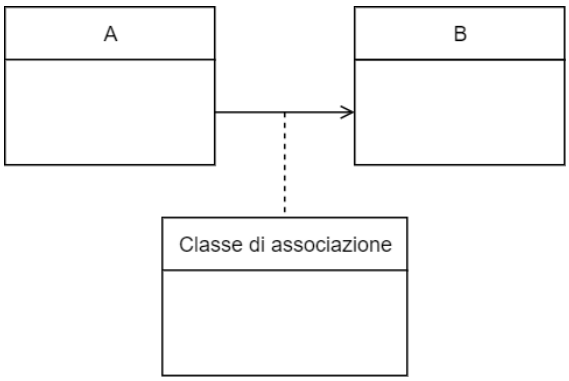

Figura 2.7. Esempio di classe di associazione.

- effetto di rete e XML promise: qualsiasi documento XML può essere letto ed elaborato da qualsiasi strumento XML. Naturalmente, alcuni strumenti XML potrebbero richiedere un markup XML specifico, ma il formato XML stesso può essere letto da qualsiasi parser XML.

I documenti XML dovrebbero iniziare con una dichiarazione XML che specifichi la versione di XML utilizzata ed eventuale codifica. Un documento XML ha una struttura simile ad un albero, comprende un elemento radice, delle divisioni o rami ed infine elementi foglia. Esiste esattamente un elemento, chiamato root, o elemento del documento, il quale contiene, internamente ai suoi tag di inizio (<Tag>) e di fine (</Tag>), tutti gli altri elementi chiamati elementi figli. In questo modo si può generare un albero di elementi annidati, con dettagli sempre più specifici. Per ogni elemento figlio, il tag di inizio deve essere contenuto nell'elemento padre ed il tag di fine deve essere contenuto nello stesso elemento padre. Se l'elemento non contiene a sua volta nessun altro elemento il suo tag può rappresentare anche la chiusura dell'elemento stesso (<Tag/>).

Le informazioni contenute nei file in XML sono facilmente utilizzabili dai linguaggi di programmazione. Esistono API implementate apposta per la gestione di questo genere di dati. Ad esempio per il linguaggio JAVA esiste Java Architecture for XML Binding (JAXB), che mette a disposizione un set di API per semplificare l'accesso e la realizzazione di documenti in formato XML.

2.2.2 XML schema definition

La struttura di un documento XML può variare molto e quindi ci sono molti modi diversi di esprimere le stesse informazioni. Questo potrebbe non essere un problema per le persone, ma ciò potrebbe impedire la corretta comunicazione tra diversi sistemi informatici. Pertanto è buona norma creare un insieme di regole per il documento che definisca quali elementi e attributi sono consentiti. In questo modo entrambe le parti comunicanti sanno cosa anticipare e possono preparare la loro logica applicativa di conseguenza.

Document Type Definition (DTD) è il linguaggio di modellazione originale per descrivere la struttura ad albero dei documenti XML. Questa è una caratteristica significativa che XML eredita da SGML. Utilizzando i DTD, XML è in grado di creare un modello per il markup dei documenti in modo che il posizionamento degli elementi e i loro attributi possano essere controllati

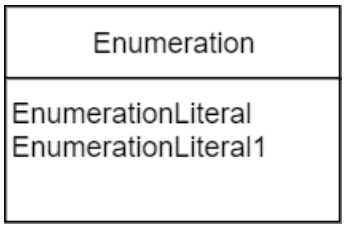

Figura 2.8. Esempio di enumerazione.

e validati. Un DTD è costituito da un numero di dichiarazioni che sono raggruppate insieme. Dal 2009 vennero introdotti metodi più adatti per creare definizioni di schema. Tra cui XMLSchema, Relax NG[3] o Schematron[4].

Il più conosciuto è il linguaggio XML Schema Definition (XSD) il cui scopo è definire la natura degli schemi e le loro parti, fornire un inventario dei costrutti di markup XML con cui rappresentare gli schemi e definire l'applicazione degli schemi ai documenti XML [7].

Lo scopo di un XSDschema è definire e descrivere una classe di documenti XML usando i componenti dello schema per vincolare e documentare il significato, l'uso e le relazioni delle loro parti costituenti: tipi di dati, gli elementi, il loro contenuto, gli attributi e i loro valori. Gli schemi possono anche fornire la specifica di informazioni aggiuntive sul documento, come la normalizzazione e il valore di default degli attributi e degli elementi. Gli schemi dispongono di strutture per l'autocertificazione. Pertanto le strutture possono essere utilizzate per definire, descrivere e catalogare i vocabolari XML per le classi di documenti XML.

In un XMLschema vengono specificati la struttura generale di un documento XML e i vincoli per le entità contenute. I seguenti componenti di un documento XML sono i principali elementi descritti dallo schema [8]:

- elemento: ogni elemento utilizzato nel documento XML è definito da una dichiarazione che include il nome, lo spazio dei nomi e il tipo dell'elemento. Lo spazio dei nomi dell'elemento non deve essere specificato esplicitamente ma può essere derivato dal suo elemento padre. L'elemento è di tipo semplice o complesso;
- attributo: ogni attributo utilizzato all'interno di un elemento XML è definito da una dichiarazione di attributo. L'attributo ha uno spazio dei nomi di destinazione (derivato) ed è sempre di tipo semplice. Inoltre, è possibile dichiarare il valore fisso o predefinito;
- tipo semplice: le istanze di tipo semplice (chiamate anche tipi di dati) sono valori singoli, ovvero in stringhe o numeri generali. Con l'uso di restrizioni è possibile specificare il loro formato o possibili valori;
- tipo complesso: un tipo complesso descrive il contenuto di un elemento, ciò significa quali elementi figlio sono consentiti, in quale ordine e quantità. Specifica inoltre gli attributi dell'elemento. I tipi complessi possono essere limitati o estesi. Usando il metalinguaggio XML Schema possiamo caratterizzare questi elementi, attributi e tipi di un documento XML.

2.2.3 Yang

Yet Another Next Generation (YANG) è un linguaggio di modellazione dei dati sviluppato e standardizzato dal gruppo di lavoro sul linguaggio di modellazione dei dati (NETMOD) di Internet Engineering Task Force (IETF) nell'anno 2010. YANG è usato per modellare la configurazione e

[3] https://relaxng.org/spec-20011203.html

[4] http://schematron.com/

dichiarare i dati manipolati dal Network Configuration Protocol (NETCONF), dalle chiamate di procedura remota NETCONF e dalle notifiche NETCONF. YANG viene utilizzato per modellare le operazioni e i livelli di contenuto di NETCONF [9]. Il protocollo NETCONF definisce un semplice meccanismo attraverso il quale è possibile gestire un dispositivo di rete, recuperare le informazioni sui dati di configurazione, caricare e manipolare nuovi dati di configurazione. Il protocollo consente al dispositivo di esporre un'interfaccia di programmazione dell'applicazione (API) completa e formale. Le applicazioni possono utilizzare questa semplice API per inviare e ricevere set di dati di configurazione completi e parziali [10]. YANG resiste alla tendenza a risolvere tutti i possibili problemi, limitando lo spazio problematico per consentire l'espressione di modelli di dati per protocolli di gestione della rete come NETCONF, non documenti XML arbitrari o modelli di dati arbitrari [11].

YANG è costituito da una serie di parole chiave integrate, generalmente denominate istruzioni e tipi, che possono essere utilizzate per costruire definizioni dettagliate dei dati modellati. Inoltre, poiché YANG è un linguaggio gerarchico, offre una sintassi gerarchica simile a JavaScript Object Notation (JSON).

Un modello YANG contiene un modulo che definisce la struttura ad albero gerarchica dei dati di configurazione e di stato, gli argomenti delle chiamate di procedura remota (RPC) e il contenuto della notifica. Per permettere la gestione separata e per agevolare l'uso dei modelli, la struttura dei modelli di dati si sviluppa in moduli, che possono importare altri moduli e sottomoduli, che a loro volta possono essere inclusi in moduli. A ogni modulo viene assegnato uno spazio dei nomi univoco. I dati descritti nel modello YANG possono essere identificati con identificatori di istanza univoci in cui ogni nodo ha un nome e un valore o un insieme di nodi figlio. YANG fornisce descrizioni chiare e concise dei nodi, nonché l'interazione tra tali nodi. Inoltre tutti i dati che possono contenere un valore vengono definiti in base al tipo e ne sono descritti i valori consentiti.

Di seguito vengono esposte le strutture principali del linguaggio:

- leaf: una foglia è l'elemento base, non contiene figli ed indica un singolo valore. L'istruzione foglia deve contenere quanto segue:
  - name: nome della foglia;
  - type: tipo di dato che rappresenta la foglia. Il tipo può essere un tipo predefinito da YANG, o un tipo derivato specificato dall'utente;

  una foglia può contenere informazioni opzionali riguardo il data model.
  - default: valore di default della foglia. Quando non viene configurato in modo esplicito il valore della foglia otterrà il valore di default;
  - mandatory: valore booleano che indica se vero il valore della foglia deve essere configurato, se falso la foglia può essere configurata oppure no;
  - constraint: permette di aggiungere al campo Type ulteriori restrizioni riguardo al valore della foglia. Ad esempio permette di limitare il range di valori che può contenere la foglia, oppure permette di impostare un pattern espresso come stringa che il valore della foglia deve rispettare;
- container: un contenitore viene usato per raggruppare un numero di nodi figlio. Un contenitore può avere foglie, elenchi, informazioni e persino altri contenitori come figli. I contenitori organizzano solo i dati, non contengono dati da soli e vengono eliminati quando non contengono nodi figlio;
- list: una lista raggruppa nodi figlio e può contenere foglie, contenitori e altre liste come elementi figlio. La differenza tra liste e contenitori è che una lista può avere molte istanze. Un'istanza della lista è chiamata voce della lista. L'istruzione key specifica le chiavi della lista. Le chiavi sono foglie speciali che vengono utilizzate per identificare una particolare voce della lista;

- type: YANG specifica diversi tipi di dati incorporati, come uint8, uint16, int8, int16, bit, string ecc. L'utente può imporre ulteriori restrizioni sui dati, usando parole chiave come range, pattern ecc. Oltre a ciò, l'utente può anche definire tipi completamente nuovi dai tipi derivati, usando l'istruzione typedef. Il tipo può essere utilizzato nel modello specificando il nome dichiarato nel typedef;
- YANG extensions: il linguaggio YANG può essere esteso con istruzioni aggiuntive, utilizzando la parola chiave extension. I moduli YANG possono utilizzare estensioni definite in alcuni altri moduli, importando tali moduli. Le estensioni sono costituite da una parola chiave e un argomento;

Negli ultimi anni YANG si è diffuso sempre di più e sono stati sviluppati dei tools per la gestione di questo nuovo linguaggio. I più diffusi sono:

- yangvalidator[5]: tool che permette la validazione di un file yang caricandolo online.
- pyang: dal sito web del progetto [16]: "Pyang è un validatore, trasformatore e generatore di codice YANG, scritto in Python". Pyang è un progetto open source che utilizza la nuova licenza BSD. Pyang è uno strumento da riga di comando che opera su file YANG, ne convalida la correttezza e permette di trasformare i moduli YANG in altri formati, ad esempio JSON e XML. Attraverso lo sviluppo di plugin, l'utente di pyang può accedere alle sue strutture di dati interne. Da lì, è facile estrarre parti del modello YANG o attraversarle. L'esempio più semplice di un plug-in pyang è il plug-in tree fornito con pyang il quale stampa semplicemente il modello YANG in un albero ASCII gerarchico.
- libyang: "libyang è un parser di linguaggio di modellazione dei dati YANG ed è toolkit scritto e che fornisce API in C" [17]. libyang permette la formattazione e la parsificazione di moduli yang. A partire da libyang è stata sviluppata una libreria NETCONF per il linguaggio C destinata alla costruzione di client e server NETCONF.
- YDK: YANG Development Kit (YDK) è un kit di sviluppo software che fornisce API modellate in YANG. L'obiettivo principale di YDK è ridurre la curva di apprendimento dei modelli di dati YANG esprimendo la semantica del modello in un'API e astraendo i dettagli del protocollo e di codifica [18].

2.2.4 Yin

I moduli YANG possono essere tradotti in una sintassi XML equivalente chiamata YANG Independent Notation (YIN), consentendo alle applicazioni che utilizzano parser XML e script XSLT (Extensible Stylesheet Language Transformations) di operare sui modelli. La conversione da YANG a YIN è semanticamente senza perdita, quindi il contenuto in YIN può essere restituito in YANG. Esiste una corrispondenza uno a uno tra le parole chiave YANG e gli elementi YIN. Il nome locale di un elemento YIN è identico alla parola chiave YANG corrispondente [9].

La notazione YIN consente agli sviluppatori di rappresentare modelli di dati YANG in XML e quindi utilizzare il ricco set di strumenti basati su XML per il filtraggio e la convalida dei dati, la generazione automatizzata di codice e documentazione e altre attività.

La Figura 2.9 mostra un modulo yang molto semplice con alcuni elementi base del linguaggio. Mediante il tool pyang è stato generato il modulo YIN rappresentato in Figura 2.10, dalla quale si nota la completa somiglianza con il linguaggio XML. In un modulo YIN si può notare la presenza delle parole chiave caratteristiche del linguaggio YANG, ad esempio il termine "leaf", parole che non sono caratteristiche del linguaggio XML. Gli elementi YIN corrispondenti alle parole chiave YANG appartengono allo spazio dei nomi il cui URI associato è "urn:ietf:params:xml:ns:yang:yin:1". Sempre mediante il tool pyang è stato generato il tree diagram in Figura 2.11 che rappresenta sotto un altro punto di vista lo stesso esempio.

[5] http://yangvalidator.org/

```
module yang1 {
   yang-version 1;
   namespace "urn:opendaylight:params:xml:ns:yang:hello";
   prefix "hello";

   revision "2015-02-24" {
      description "Basic Hello World YANG model.";
   }
   rpc hello-world {
      input {
         leaf name {
            type string;
         }
      }
      output {
         leaf greeting {
            type string;
         }
      }
   }
}
```

Figura 2.9. Esempio modulo YANG.

```
<?xml version="1.0" encoding="UTF-8"?>
<module name="yang1"
       xmlns="urn:ietf:params:xml:ns:yang:yin:1"
       xmlns:hello="urn:opendaylight:params:xml:ns:yang:hello">
   <yang-version value="1"/>
   <namespace uri="urn:opendaylight:params:xml:ns:yang:hello"/>
   <prefix value="hello"/>
   <revision date="2015-02-24">
      <description>
         <text>Basic Hello World YANG model.</text>
      </description>
   </revision>
   <rpc name="hello-world">
      <input>
         <leaf name="name">
            <type name="string"/>
         </leaf>
      </input>
      <output>
         <leaf name="greeting">
            <type name="string"/>
         </leaf>
      </output>
   </rpc>
</module>
```

Figura 2.10. Esempio modulo YIN.

```
module: yang1

   rpcs:
      +---x hello-world
         +---w input
         | +---w name? string
         +--ro output
         +--ro greeting? string
```

Figura 2.11. Esempio tree diagram.

2.3 Information model vs data model

I modelli informativi (IM o Information Model) vengono utilizzati per modellare oggetti gestiti a livello concettuale, indipendentemente da qualsiasi protocollo specifico utilizzato per trasportare i dati. Il grado di dettaglio delle astrazioni definite nel modello informativo dipende dalle esigenze di modellazione dei suoi progettisti. Al fine di rendere il progetto complessivo il più chiaro possibile, i modelli informativi dovrebbero nascondere i dettagli sia del protocollo sia dell'implementazione. I modelli informativi si concentrano sulle relazioni tra gli oggetti interessati. I modelli informativi sono utili principalmente per i progettisti per descrivere l'ambiente gestito, per gli operatori per comprendere gli oggetti modellati e per gli implementatori come guida per le funzionalità che devono essere descritte e codificate nei DM. I termini "modelli concettuali" e "modelli astratti", che sono spesso usati in letteratura, si riferiscono ai modelli informativi. Una caratteristica importante degli IM è che possono (e generalmente dovrebbero) specificare le relazioni tra gli oggetti. Le organizzazioni possono utilizzare i contenuti di un modello informativo per delimitare le funzionalità che possono essere incluse in un modello dei dati (Data Model o DM). I modelli informativi possono essere definiti in modo informale, utilizzando lingue naturali. In alternativa, gli IM possono essere definiti usando un linguaggio formale o un linguaggio strutturato semi-formale. Una delle possibilità per specificare formalmente i messaggi istantanei è utilizzare il diagramma delle classi di UML. Un importante vantaggio dei diagrammi delle classi UML è che rappresentano gli oggetti e le relazioni tra loro in modo grafico standard. Grazie a questa rappresentazione grafica, i progettisti e gli operatori potrebbero trovare più semplice comprendere il modello di gestione sottostante. Sebbene esistano altre tecniche per rappresentare graficamente oggetti e relazioni (ad esempio diagrammi Entità-Relazione (ER)), UML presenta il vantaggio di essere ampiamente adottato nel settore e insegnato nelle università [15].

Rispetto agli IM, i DM definiscono gli oggetti gestiti a un livello inferiore di astrazione. Sono destinati agli implementatori ed includono dettagli specifici relativi all'implementazione e al protocollo, ad esempio regole che spiegano come mappare oggetti gestiti su costrutti di protocollo di livello inferiore. I modelli di dati sono spesso rappresentati in linguaggi formali di definizione dei dati specifici del protocollo di gestione utilizzato. La maggior parte dei modelli di gestione standardizzati finora sono DM e per esprimere le relazioni tra oggetti le tecniche come i diagrammi UML ed ER danno ancora i risultati migliori, perché sono i diagrammi più facili da capire [15].

Poiché i modelli concettuali possono essere implementati in diversi modi, è possibile derivare più modelli di dati da un singolo modello informativo, come rappresentato in Figura 2.12. Sebbene i modelli informativi e i modelli di dati abbiano scopi diversi, non è sempre facile decidere quali dettagli appartengano a un modello informativo e quali appartengano a un modello di dati. Allo stesso modo, a volte è difficile determinare se un'astrazione appartiene ad un modello informativo o ad un modello di dati.

2.4 Design pattern

Nella progettazione dei software ad oggetti è facile incontrare problemi ricorrenti, per risolvere i quali è utile trovare soluzioni riutilizzabili: A questo servono i design pattern. Christopher

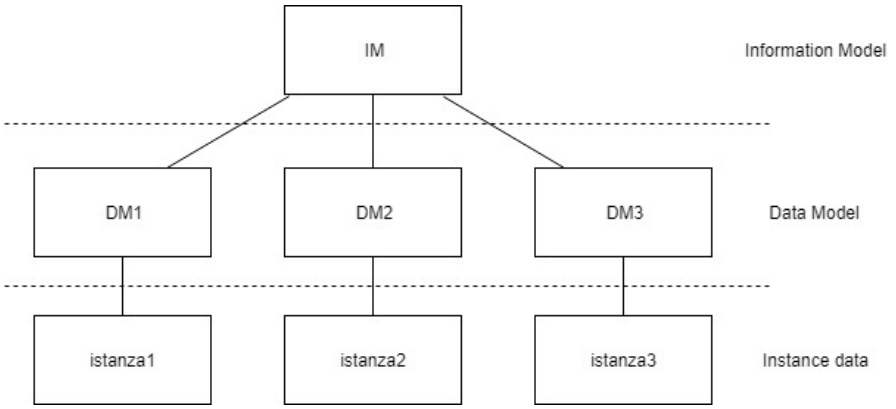

Figura 2.12. IM vs DM.

Alexander disse: "Ogni pattern descrive un problema che si verifica ripetutamente nel nostro ambiente, e quindi descrive il nucleo della soluzione a quel problema, in modo tale da poter utilizzare questa soluzione un milione di volte, senza mai farlo allo stesso modo due volte" [13].

Sono stati definiti 23 tipi di design pattern nel libro "Design Patterns: Elements of Reusable Object-Oriented Software" da un gruppo di quattro progettisti software (la "Gang of Four") nel 1994 [12], grazie al quale si iniziarono a studiare metodi e progettazioni di software riutilizzando soluzioni architetturali precedentemente testate. In particolare sono stati catalogati i pattern utilizzando un formalismo ben preciso. Ogni pattern infatti, viene presentato tramite il nome del pattern, il problema a cui può essere applicato, la soluzione (non in un caso particolare), le conseguenze dovute all'applicazione del pattern. Gamma definì il termine Design pattern come "una soluzione riutilizzabile generale al problema che si verifica comunemente nella progettazione del software" [12]. Un modello di progettazione è una descrizione per come risolvere un problema che può essere utilizzato in molte situazioni diverse. I design patterns migliorano la documentazione del software, permettono di accelerare il processo di sviluppo, consentono il riutilizzo su larga scala di architetture software, permettono di acquisire conoscenze specialistiche, creano compromessi di progettazione e possono aiutare a ristrutturare i sistemi.

Con i design pattern è possibile trovare un aiuto per la strutturazione del progetto, evitando la possibile creazione di soluzioni azzardate con architetture poco consistenti o che non rispettano i paradigmi della programmazione orientata agli oggetti.

Le categorie definite dalla "Gang of Four" in cui si possono suddividere i design pattern sono:

- pattern creazionali;
- pattern strutturali;
- pattern comportamentali.

2.4.1 Pattern creazionali

I pattern creazionali astraggono il processo di istanziazione, aiutano a rendere un sistema indipendente dal modo in cui i suoi oggetti vengono creati, composti e rappresentati. Definiscono il modo migliore per la creazione degli oggetti considerando il riutilizzo e la modificabilità ed il miglior modo per gestire le istanze. Utilizzando questo tipo di pattern viene nascosto il modo in cui le istanze delle classi vengono create e messe insieme.

I pattern creazionali definiti dalla "Gang of Four" sono:

- abstract factory: pattern con l'intento di fornire un'interfaccia per la creazione di famiglie di oggetti correlati o dipendenti senza specificare le loro classi concrete;

- builder: pattern con l'intento di separare la costruzione di un oggetto complesso dalla sua rappresentazione in modo che lo stesso processo di costruzione possa creare; rappresentazioni diverse;
- factory method: pattern con l'intento di definire un'interfaccia per la creazione di un oggetto, ma lasciare che le sottoclassi decidano di quale classe creare l'istanza;
- prototype: pattern con l'intento di specificare i tipi di oggetti da creare usando un prototipo e creare nuovi oggetti copiando questo prototipo;
- singleton: pattern con l'intento di assicurare che una classe abbia solo un'istanza e fornendo un punto di accesso globale ad essa.

2.4.2 Pattern strutturali

I pattern strutturali descrivono come gli oggetti e le classi possono essere combinati per formare delle strutture più grandi. Si possono distinguere modelli di oggetti e modelli di classe. I primi descrivono come gli oggetti possono essere associati ed aggregati per formare strutture più grandi e complesse; i secondi descrivono le relazioni e strutture con l'aiuto dell'ereditarietà.

I pattern strutturali definiti dalla "Gang of Four" sono:

- adapter: pattern con l'intento di convertire l'interfaccia di una classe in un'altra interfaccia che i client si aspettano. L'adapter consente alle classi di lavorare insieme, altrimenti non potrebbero a causa di interfacce incompatibili;
- bridge: pattern con l'intento di disaccoppiare un'astrazione dalla sua implementazione in modo che i due possano variare in modo indipendente;
- composite: pattern con l'intento di comporre gli oggetti in strutture ad albero per rappresentare gerarchie intere. Il composito consente ai client di trattare in modo uniforme singoli oggetti e composizioni di oggetti;
- decorator: pattern con l'intento di associare in modo dinamico responsabilità aggiuntive ad un oggetto. I decoratori forniscono un'alternativa flessibile all'utilizzo delle sottoclassi per estendere le funzionalità degli oggetti;
- facade: pattern con l'intento di fornire un'interfaccia unificata a un set di interfacce in un sottosistema. Facade definisce un'interfaccia di livello superiore che semplifica l'utilizzo del sottosistema;
- flyweight: pattern con l'intento di utilizzare la condivisione per supportare in modo efficiente un gran numero di oggetti concreti;
- proxy: pattern con l'intento di fornire un surrogato o un segnaposto per un altro oggetto per controllarne l'accesso.

2.4.3 Pattern comportamentali

I pattern comportamentali riguardano l'assegnazione delle responsabilità tra gli oggetti, descrivono il comportamento di un insieme di oggetti che cooperano e descrivono i modelli di comunicazione tra loro. I pattern comportamentali utilizzano l'ereditarietà per distribuire il comportamento tra le classi.

I pattern comportamentali definiti dalla "Gang of Four" sono:

- chain of responsibility: pattern con l'intento di evitare di accoppiare il mittente di una richiesta al suo destinatario dando a più di un oggetto la possibilità di gestire la richiesta. Incatenare gli oggetti riceventi e passare la richiesta lungo la catena fino a quando un oggetto non lo gestisce;

- command: pattern con l'intento di incapsulare una richiesta come oggetto, permettendo così di parametrizzare i client con diverse richieste, salvandole in una coda;
- interpreter: pattern con l'intento di dato un linguaggio, definire una rappresentazione o la sua grammatica insieme ad un interprete che utilizza la rappresentazione per interpretare le frasi nel linguaggio;
- iterator: pattern con l'intento di fornire un modo per accedere in sequenza agli elementi di un oggetto aggregato senza esporre la sua rappresentazione sottostante;
- mediator: pattern con l'intento di definire un oggetto che incapsula il modo in cui un insieme di oggetti interagisce. Il mediatore promuove l'accoppiamento libero impedendo agli oggetti di riferirsi esplicitamente tra loro e consente di variare la loro interazione in modo indipendente;
- memento: pattern con l'intento di acquisire ed esternare lo stato interno di un oggetto in modo che l'oggetto possa essere ripristinato in questo stato in un secondo momento, senza violare l'incapsulamento;
- observer: pattern con l'intento di definire una dipendenza uno a molti tra gli oggetti in modo tale che quando un oggetto cambia stato, tutti i suoi dipendenti vengano notificati e aggiornati automaticamente;
- state: pattern con l'intento di consentire ad un oggetto di modificare il suo comportamento quando cambia il suo stato interno. L'oggetto sembrerà cambiare la sua classe;
- strategy: pattern con l'intento di definire una famiglia di algoritmi, incapsulare ciascuno di essi e renderli intercambiabili. La strategia consente all'algoritmo di variare indipendentemente dai client che lo utilizzano;
- template Method: pattern con l'intento di definire lo scheletro di un algoritmo in un'operazione, rimandando alcuni passaggi alle sottoclassi. il metodo template consente alle sottoclassi di ridefinire determinati passaggi di un algoritmo senza modificare la struttura dell'algoritmo;
- visitor: pattern con l'intento di rappresentare un'operazione da eseguire sugli elementi di una struttura di oggetti. Il visitatore consente di definire una nuova operazione senza modificare le classi degli elementi su cui opera.

2.4.4 Decorator pattern

In questa sezione approfondisco il decorator pattern perchè è il pattern più utilizzato per la realizzazione dei modelli della tesi.

A volte si vuole aggiungere responsabilità ai singoli oggetti e non ad un'intera classe. Un modo per aggiungere responsabilità è con l'eredità. Ereditare proprietà fa si che le caratteristiche del padre vengano aggiunte a tutte le istanze delle sottoclassi. Ciò, tuttavia, non è flessibile poiché la scelta delle proprietà aggiuntive viene effettuata staticamente. Un client non può controllare come e quando applicare nuove funzionalità al nuovo componente ma può solo basarsi sulla creazione delle classi esistenti. Un approccio più flessibile è racchiudere il componente, al quale si vogliono aggiungere responsabilità, in un altro oggetto che aggiunge effettivamente la caratteristica desiderata. L'oggetto che racchiude si chiama decoratore. Il decoratore è conforme all'interfaccia del componente che decora in modo che la sua presenza sia trasparente per il client del componente. Il decoratore inoltra le richieste al componente e può eseguire azioni aggiuntive prima o dopo l'inoltro. La trasparenza consente di annidare i decoratori in modo ricorsivo, permettendo così un numero illimitato di responsabilità aggiuntive.

E. Gamma, nel libro "Design Patterns: Elements of Reusable Object-Oriented Software" definisce il decorator pattern come in Figura 2.13.

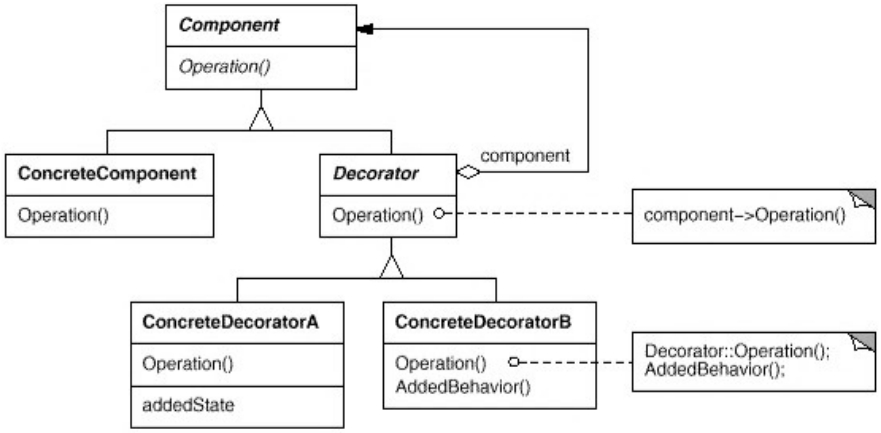

Figura 2.13. Decorator pattern (fonte: codeproject).

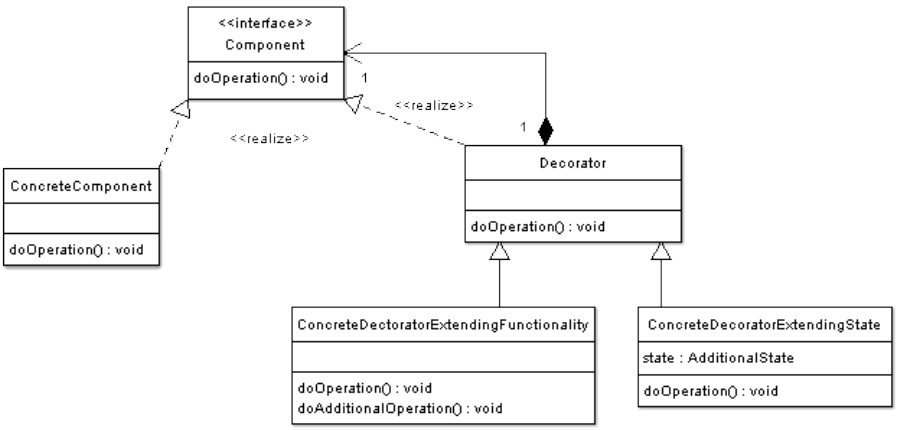

Figura 2.14. Decorator pattern (fonte: oodesign).

Un modo simile per rappresentare la struttura di un decorator pattern è rappresentata in Figura 2.14, nella quale viene espresso in linguaggio UML il fatto che la classe Component è stata creata come interfaccia per ottenere la trasparenza degli oggetti che ereditano da essa.

Gli elementi principali del Decorator pattern e comuni nelle due rappresentazioni sono:

- component: interfaccia per gli oggetti a cui è possibile aggiungere responsabilità in modo dinamico;
- concreteComponent: oggetto al quale è possibile aggiungere responsabilità;
- decorator: mantiene un riferimento all'oggetto Component e definisce un'interfaccia conforme all'interfaccia del componente;
- concreteDecorator: i ConcreteDecorator sono le classi che effettivamente estendono la funzionalità del Component al quale sono collegati.

2.4.5 Antipattern

La maggior parte delle opere pubblicate nel campo delle scienze del software si è concentrata su soluzioni positive e costruttive. Oltre ai design pattern sono stati studiati e documentati una serie di "antipattern". Un AntiPattern è una forma letteraria che descrive una soluzione comune a un problema che genera conseguenze decisamente negative. Gli AntiPattern sono un metodo per mappare in modo efficiente una situazione generale a una specifica classe di soluzioni. Forniscono un vocabolario comune per identificare i problemi e discutere le soluzioni. Gli AntiPattern, come le loro controparti i design pattern, definiscono un vocabolario industriale per i comuni processi e implementazioni difettosi all'interno delle organizzazioni [14].

2.5 Model driven

2.5.1 Model Driven Engineering

Nell'ingegneria guidata dai modelli (Model Driven Engineering, MDE), spesso chiamata anche sviluppo guidato dai modelli (model-driven development, MDD), i modelli vengono utilizzati per raggiungere un livello più elevato di astrazione rispetto a quanto sia possibile con i linguaggi di programmazione [25]. Model-Driven Engineering (MDE) si riferisce all'uso sistematico di modelli come entità di prima classe durante l'intero ciclo di vita dell'ingegneria del software. Gli approcci basati sui modelli spostano il focus dello sviluppo dai codici del linguaggio di programmazione ai modelli espressi in linguaggi di modellazione specifici del dominio, quindi il termine MDE viene utilizzato per i processi di sviluppo incentrati sul modello anziché incentrati sul codice [53]. L'obiettivo è aumentare la produttività e ridurre il *time to market*, consentendo lo sviluppo di sistemi complessi mediante modelli definiti con concetti che sono molto meno legati alla tecnologia di implementazione sottostante e sono molto più vicini al dominio del problema. Ciò rende i modelli più facili da specificare, comprendere e mantenere [26], aiutando a comprendere problemi complessi e le loro potenziali soluzioni attraverso le astrazioni. Il concetto di Model Driven Engineering è emerso come una generalizzazione della Model Driven Architecture (MDA) proposta da OMG nel 2001 [20]. Kent definisce MDE sulla base di MDA aggiungendo la nozione di processo di sviluppo software e spazio di modellizzazione per l'organizzazione di modelli [28]. Favre propone una visione di MDE come un approccio aperto e integrativo che si collega con molti altri spazi tecnologici (TS) in modo uniforme [27]. Un aspetto importante di MDE è l'enfasi che pone sui ponti tra spazi tecnologici e sull'integrazione di corpi di conoscenza sviluppati da diverse comunità di ricerca. Gli esempi di TS includono non solo MDA e MOF, ma anche Grammarware e BNF, Documentware eXML, Dataware e SQL, Modelware e UML, ecc. In ogni spazio, i concetti di modello, metamodello e trasformazione assumono un diverso significato, ma tutti fornito un insieme di standard per esprimere modelli, relazioni tra modelli e trasformazioni da modello a modello. Un modello viene anche definito come un insieme di dichiarazioni su un sistema in fase di studio [31]. La guida MDA definisce un modello di un sistema come una descrizione o una specifica di quel sistema e del suo ambiente per un certo scopo. Un modello viene spesso presentato come una combinazione di disegni e testo. Il testo può essere in un linguaggio di modellazione o in un linguaggio naturale [20]. Inoltre, c'è l'idea che "tutto è un modello" [29], il che significa che tutti gli artefatti utilizzati per descrivere e sviluppare il sistema - comprese le descrizioni del codice e del linguaggio naturale - sono considerati e trattati come modelli. In particolare, ciò significa che non esiste una distinzione tra un insieme di artefatti - ad esempio il codice sorgente - che sono il sistema e un insieme di modelli che rappresentano il sistema. Gli strumenti e i processi dell'ingegneria basata sui modelli sono sfruttati per gestire tutti i manufatti coinvolti. Dato che di solito vi sono sovrapposizioni nelle informazioni rappresentate nei diversi modelli, se si estraggono dalle stesse "informazioni" per scopi diversi e in modi diversi, è necessario un meccanismo per propagare le modifiche ad altri modelli. I modelli vengono aggiornati se cambiano altri modelli che contengono informazioni sovrapposte. Questo approccio è anche chiamato ingegneria di andata e ritorno (Round-Trip Engineering RTE [30]) e consente allo sviluppatore di analizzare e comprendere continuamente il sistema in evoluzione da punti di vista pertinenti.

2.5.2 Model Driven Architecture

Model Driven Architecture (MDA) è una metodologia di progettazione e sviluppo del sistema per assistere lo sviluppo del sistema stesso con un approccio indipendente dalla tecnologia. MDA è stata descritta per la prima volta dall'OMG nel 2001 [19]. MDA incarna la visione di presentare un quadro generale per supportare l'interoperabilità con le specifiche durante l'intero ciclo di vita di un sistema. La filosofia di progettazione alla base della MDA deve riguardare la descrizione della logica aziendale, la modularizzazione, la costruzione e l'integrazione dei componenti di un sistema, nonché l'implementazione, la gestione e l'evoluzione. Un concetto chiave di MDA è la separazione delle specifiche di funzionalità dalle specifiche di implementazione, integrazione e distribuzione [20]. Ciò si ottiene applicando l'astrazione al processo di progettazione. L'astrazione è intesa come soppressione di dettagli irrilevanti come motivata dal modello di riferimento per l'elaborazione distribuita aperta . La controparte dell'astrazione in MDA è la specializzazione, indicata anche come raffinamento. Una volta definita la funzionalità di un sistema a livello astratto, questa definizione viene migliorata da un numero sempre maggiore di dettagli di implementazione fino a quando il sistema non viene specificato al suo livello di implementazione. L'MDA affronta tre obiettivi principali:

- portabilità;
- interoperabilità;
- riutilizzabilità.

Pertanto, OMG fornisce concetti e strumenti tramite MDA per specificare un sistema indipendentemente dalle piattaforme di distribuzione indirizzate, modellare le piattaforme stesse, scegliere una piattaforma particolare per il sistema progettato tra più candidati e trasformare le specifiche di sistema in una compatibile con la piattaforma selezionata [20].

L'elemento fondamentale in MDA è il modello come già sottolinea il termine Model Driven Architecture. All'interno della MDA, un modello si riferisce a "una rappresentazione di una parte della funzione, struttura e / o comportamento di un sistema" [19]. Questa rappresentazione può descrivere specifiche arbitrarie della struttura, del comportamento o dell'ambiente di un sistema e può essere presentata come testo o disegni. Pertanto, un modello è una descrizione formale di un artefatto di un'applicazione complessa, come uno schema di database, un'interfaccia di applicazione, un modello UML, un'ontologia o un formato di messaggio [21]. Una specifica di sistema tipica è costituita da molti modelli. In tal modo ogni modello riflette una vista o sottosistema diverso. Per guidare la creazione di modelli, MDA fornisce due approcci a uno sviluppatore:

- perfezionamento del modello: concetto che fornisce i mezzi per aggiungere informazioni più dettagliate sul sistema a un modello esistente. Queste informazioni possono espandere aspetti formalmente astratti del sistema o aggiungere specifiche di implementazione. Il perfezionamento del modello può essere visto come un'operazione sul posto in cui il modello stesso viene modificato;
- trasformazione del modello: la trasformazione del modello, creerà nuovi modelli basati su quelli esistenti. Applicando le trasformazioni del modello, gli elementi di un modello esistente verranno convertiti o mappati in elementi di un modello appena creato.

Utilizzando i concetti di modello, perfezionamento e trasformazione, l'MDA accompagna l'intero ciclo di vita di un sistema fornendo mezzi per utilizzare i modelli per specificare l'evoluzione dei sistemi, implicando una sintassi e una semantica ben definite di questi modelli [20].

Per MDA, OMG ha identificato quattro livelli di modellazione, denominati M3, M2, M1 e M0. Il livello più alto, e quindi più astratto, è M3, il livello Meta-Meta-Model. Tale meta-meta-modello definisce le primitive e i concetti più fondamentali che sono necessari per descrivere un modello. Nel contesto di MDA, questo livello è descritto da Meta Object Facility (MOF) [22]. Il MOF è un framework di modellizzazione e gestione dei metadati da utilizzare per specificare i

linguaggi di modellazione stessi. Tali linguaggi di modellazione sono rappresentati nel successivo livello inferiore M2, il livello Meta-modello. Un meta-modello rappresenta un'incarnazione di una lingua specifica del dominio. Un esempio di spicco di tale definizione di metamodello è l'UML. Sulla base di metamodelli, un progettista di sistema specifica un modello di sistema nel livello M1, il livello Modello. Infine, sullo strato M0, lo strato di sistema, il sistema concreto è descritto come un'implementazione conforme al modello nello strato M1.

MDA classifica i modelli in quattro categorie a seconda della relazione di un modello con una piattaforma, fornendo diversi livelli di astrazione del sistema modellato. In termini di MDA, una piattaforma è definita come "un insieme di sottosistemi e tecnologie che forniscono un insieme coerente di funzionalità attraverso interfacce e schemi di utilizzo specifici, che qualsiasi applicazione supportata da quella piattaforma può utilizzare senza preoccupazioni per i dettagli su come viene implementata la funzionalità fornita dalla piattaforma" [20].

Il documento "MDA Guide Version 1.0.1, Object Management Group (OMG)" definisce le seguenti quattro categorie [20]:

- Computation Independent Model: il Computation Independent Model (CIM) è una visione del sistema modellato focalizzata sul ruolo che svolge all'interno dell'ambiente di sistema. Il CIM nasconde qualsiasi informazione strutturale e comportamentale degli elementi interni al sistema. Il CIM viene anche definito modello di dominio o modello di business;
- Platform Independent Model: un Platform Independent Model (PIM) è una vista del sistema modellato che presenta informazioni sulla struttura e / o sul comportamento del sistema ma omette qualsiasi conoscenza dettagliata di come la sua piattaforma di supporto sta realizzando le sue funzionalità di basso livello. Un PIM di un sistema può essere visto come una realizzazione di sistema basata su una macchina virtuale. Sebbene la macchina virtuale stessa rappresenti una piattaforma e sebbene il modello non sia indipendente dalla piattaforma, la macchina virtuale nasconde dettagli su come vengono realizzate le sue primitive funzionali su una vera piattaforma informatica;
- Platform Model: un Platform Model contiene entità di modellazione che descrivono le singole parti funzionali di una particolare piattaforma, come API (Application Programming Interface), blocchi funzionali o concetti tecnici;
- Platform Specific Model: contrariamente al PIM, un modello specifico di piattaforma (PSM) è una vista del sistema modellato che include informazioni dettagliate sulla piattaforma su cui è realizzato il sistema. Il PSM combina il PIM e il modello di piattaforma. Pertanto, il PSM descrive le conoscenze su come il sistema utilizza una particolare piattaforma e su come viene mappato ai blocchi costitutivi della piattaforma.

2.5.3 Model Driven Transformation

La guida MDA definisce una trasformazione del modello come "il processo di conversione di un modello in un altro modello dello stesso sistema" [20]. Kleppe definisce una trasformazione come la generazione automatica di un modello target da un modello sorgente, secondo una definizione di trasformazione [23]. Una definizione di trasformazione è un insieme di regole di trasformazione che descrivono come un modello nella lingua di origine può essere trasformato in un modello nella lingua di destinazione. Una regola di trasformazione è una descrizione di come uno o più costrutti nella lingua di origine possono essere trasformati in uno o più costrutti nella lingua di destinazione.

Un modello acquisisce una progettazione concreta del sistema in un determinato punto del ciclo di vita del sistema, fornendo un livello specifico di astrazione della struttura o del comportamento del sistema. La progettazione di un sistema si evolve in base alle attività di perfezionamento del modello, pertanto ai modelli del sistema vengono aggiunte ulteriori conoscenze e dettagli sul sistema in fase di sviluppo. A completamento di queste attività di modellazione, la trasformazione del modello è la seconda attività di progettazione in MDA. Pertanto, una trasformazione del modello viene utilizzata per automatizzare le attività di progettazione, molto spesso per derivare

un PSM da un PIM. Applicando le trasformazioni del modello, il modello di un sistema avanza in modo iterativo da un modello astratto, indipendente dalla tecnologia, a uno concreto, dipendente dalla piattaforma. Alla fine, il codice del sistema (sorgente) e le informazioni sulla distribuzione vengono generati come risultato finale del processo di progettazione MDA.

Una trasformazione del modello ha effetto su PIM o PSM. Pertanto, sono possibili quattro diverse combinazioni di modelli sorgente e target [20]:

- trasformazione da PIM a PIM: le trasformazioni da PIM a PIM sono correlate alla raffinatezza del modello. Questa trasformazione viene applicata quando un modello viene filtrato, adattato o espanso senza aggiungere ulteriori informazioni rilevanti sulla piattaforma;
- trasformazione da PSM a PSM: le trasformazioni da PSM a PSM vengono utilizzate per la modularizzazione e la distribuzione di componenti di runtime. Ad esempio, i processi sono distribuiti e assegnati a nodi di calcolo indipendenti della piattaforma di destinazione. Le trasformazioni da PSM a PSM possono anche essere viste come una variante del raffinamento del modello rispetto alle caratteristiche della piattaforma. Contrariamente alle trasformazioni da PIM a PIM, questo tipo di trasformazione non influenza né la struttura del sistema né le definizioni di comportamento;
- trasformazione da PIM a PSM: una trasformazione da PIM a PSM è la forma più rilevante di trasformazioni in MDA. Con la presente, un modello, privo di conoscenza dipendente dalla piattaforma, viene proiettato sulla sua infrastruttura di esecuzione. Questa proiezione si basa sulle caratteristiche della piattaforma fornite da un modello di piattaforma;
- trasformazione da PSM a PIM: le trasformazioni da PSM a PIM rappresentano una sorta di reingegnerizzazione delle attività. Viene analizzato un sistema concreto e le sue dipendenze dalla piattaforma vengono rimosse per astrazione. Questa è la trasformazione inversa di una trasformazione da PIM a PSM.

Indipendentemente dal tipo di modelli sorgente e target per una trasformazione, l'OMG ha identificato diversi approcci da applicare per le trasformazioni del modello [20]:

- marcatura: per la marcatura, gli elementi del modello di origine sono annotati con speciali meta tag. All'esecuzione della trasformazione del modello, questi elementi del modello di origine vengono selezionati dai loro segni per controllare la loro mappatura agli elementi del modello target;
- trasformazione Meta Model: per le trasformazioni dei metamodelli, le regole di trasformazione sono definite in base rispettivamente ai metamodelli del modello di origine e di destinazione. Definendo le regole di trasformazione basate su metamodelli, la trasformazione diventa generica in quel senso, poiché può essere applicata a modelli di sorgente arbitrari (istanze del metamodello di origine) per generare modelli di destinazione (istanze del metamodello di destinazione) senza la necessità per preparazioni speciali dei modelli sorgente;
- trasformazione pura del modello: la trasformazione del modello puro viene utilizzata quando gli elementi del modello di origine vengono proiettati su elementi del modello di destinazione sulla base di una mappatura dedicata degli elementi tra il modello di origine e di destinazione. Contrariamente alle trasformazioni del metamodello, i tipi dal modello sorgente vengono mappati direttamente ai tipi del modello target anziché mappare i concetti del metamodello. Ad esempio, i tipi di dati primitivi nel modello di origine vengono sostituiti direttamente dai tipi corrispondenti per il modello di destinazione. Applicazione del modello. L'applicazione del modello viene applicata se il modello di origine riflette uno specifico modello di progettazione del sistema e il modello di destinazione supporta nativamente la realizzazione di tale modello. Quindi tutti gli elementi del modello sorgente che formano quel modello sono mappati come un gruppo alle entità di realizzazione del modello target del modello;
- unione di modelli: la fusione dei modelli può essere utilizzata se due o più modelli di un sistema hanno lo stesso livello di astrazione ma affrontano aspetti ortogonali del sistema. Quindi la fusione dei modelli fornisce un modo per combinare questi aspetti in un solo modello.

Per eseguire trasformazioni di modello in modo automatizzato, è necessario un linguaggio formale per specificare le regole di mappatura necessarie. Esistono molti approcci possibili per questo compito. Ad esempio, è possibile utilizzare un linguaggio di programmazione generico per implementare un'applicazione che esegue un insieme concreto di regole di trasformazione, oppure ci si può affidare a linguaggi specifici di dominio. Un linguaggio specifico per il dominio (domain-specific language DSL) è un "linguaggio di programmazione per computer di limitata espressività incentrato su un determinato dominio" [24].

Capitolo 3

Stato dell'arte

In questa sezione vengono introdotte e spiegate alcune tecniche ed alcuni progetti rilevanti per lo sviluppo della tesi.

3.1 Network Function Virtualization, Software-defined networking, Network Security Functions

Il termine virtualizzazione è stato coniato negli anni '60, per riferirsi a una macchina virtuale. La creazione e la gestione di macchine virtuali è stata definita virtualizzazione di piattaforme o server [32]. Un sistema di virtualizzazione è un'architettura in grado di separare un sistema operativo dalle risorse della piattaforma sottostante. In sostanza, la virtualizzazione è l'atto di separare il servizio effettivo dall'hardware che lo fornisce.

3.1.1 Network Function Virtualization

Network Function Virtualization, NFV, è un concetto che propone di integrare e, a lungo termine, sostituire dispositivi di rete fisici con funzioni di rete virtuale. Ad esempio, invece di vendere un router di pacchetti come unità hardware con circuiti stampati specializzati, NFV consente alla parte software di questo router di pacchetti di essere venduta separatamente come macchina virtuale pronta per essere istanziata in un ambiente cloud. Ciò consente ai produttori di apparecchiature di rete di innovare più rapidamente, poiché il software ha cicli di sviluppo più brevi dell'hardware. I venditori possono risparmiare denaro riducendo i loro costi di produzione.

NFV mira a trasformare il modo in cui gli operatori di rete progettano le reti evolvendo la tecnologia di virtualizzazione IT standard per consolidare molti tipi di apparecchiature di rete su server, switch e storage ad alto volume standard del settore, che potrebbero trovarsi in datacenter, nodi di rete e nei locali dell'utente finale. NFV implica l'implementazione di funzioni di rete in software che possono essere eseguite su una gamma di hardware e server standard del settore e che possono essere spostati o istanziati in varie posizioni della rete come richiesto, senza la necessità di installazione di nuove attrezzature.

Network Function Virtualization è un paradigma di rete emergente che mira a ridurre i costi e il *time to market*, migliorare la gestibilità e favorire la concorrenza e i servizi innovativi. NFV sfrutterà le tecnologie di virtualizzazione e cloud computing per trasformare le funzioni di rete in funzioni di rete virtualizzate (VNF), che verranno implementate nel software e funzioneranno come macchine virtuali (VM) su hardware di prodotti situati in data center ad alte prestazioni, in particolare Network Function Virtualization Infrastructures (NFVI). Vi è un'offerta crescente e un numero elevato di venditori NFV che competono su VNF, gestione e orchestrazione NFV (MANO) e prodotti per l'accelerazione della rete [33]. Le soluzioni NFV devono competere non solo in termini di costi e gestibilità, ma anche in termini di prestazioni e affidabilità. Le normative in materia di telecomunicazioni impongono requisiti di trasporto alle funzioni di rete, che devono

raggiungere costi di elaborazione dei pacchetti estremamente bassi, latenza controllata ed efficiente commutazione virtuale, insieme a rapidità e ripristino automatico da guasti (nell'ordine di pochi secondi) e disponibilità estremamente elevata. È noto che questi requisiti sono soddisfatti dalle tradizionali funzioni di rete (basate su hardware), che si sono dimostrate molto affidabili negli ultimi decenni. Tuttavia, le prestazioni e l'affidabilità sono definitivamente una grande sfida da raggiungere nella prossima generazione di funzioni di rete, dove la maggior parte della logica di controllo sarà implementata mediante software e tecnologie di virtualizzazione.

I requisiti desiderati dalla virtualizzazione sono trattati nel documento di ETSI GS NFV "Network Functions Virtualization (NFV); Virtualization Requirements" [34]. Tali requisiti non includono aspetti quali la definizione di protocolli ed interfacce, ma vogliono analizzare quali sono le differenze introdotte da NFV rispetto al passato e valutare quali caratteristiche risultano desiderabili nella definizione di un'architettura. Di seguito vengono riportati i principali:

- portabilità: capacità di caricare ed eseguire le funzioni di rete in ambienti eterogenei, ovvero su datacenter differenti anche se popolati da dispositivi standard;
- prestazioni: definire in modo completo quali sono le caratteristiche necessarie alla descrizione dei requisiti in termini di performance per le funzioni di rete;
- gestione ed orchestrazione: definire i meccanismi secondo i quali viene gestito ed orchestrato il ciclo di vita delle VNF, delle risorse infrastrutturali e delle operazioni su di esse;
- elasticità: provvedere a dei meccanismi per scalare (scale up / scale down) in modo semplice le risorse hardware in relazione al traffico;
- sicurezza: necessità di analizzare quali possono essere gli eventuali attacchi all'infrastruttura NFV;
- resilienza e stabilità delle reti: assicurare la capacità di un servizio di rete nel resistere ai guasti e tornare alla normale operatività, limitando eventuali disservizi;
- continuità del servizio: analizzare quali meccanismi permettano di assicurare due delle caratteristiche fondamentali di un servizio di rete, ovvero disponibilità e continuità, in conformità alle specifiche dello stesso e del Service Level Adreement (SLA);
- operazioni: capacità di automatizzare le funzioni operative, quali capacità della rete di adattarsi al carico, aggiornamenti software ed interventi per i guasti;
- efficienza energetica: utilizzare tecniche per minimizzare il consumo di energia, in considerazione di uno scenario reali di server su vasta scala;
- migrazione e coesistenza con le piattaforme esistenti: tener conto dell'attuale periodo di transizione degli scenari di rete, dove coesistono reti non virtualizzate con reti virtualizzate e far si che questo non abbia un impatto negativo sui servizi erogati all'utente.

L'architettura di un sistema NFV è caratterizzato da tre tipi di componenti [35]:

- le funzioni di rete virtualizzate (VNF) sono funzioni di rete implementate nel software. I VNF vengono utilizzati per elaborare il traffico di rete (in base al protocollo di rete specifico e alla topologia di rete). Usano risorse sia virtuali che fisiche. Inoltre, ci sono componenti chiamati Element Management (EM) che svolgono funzioni di gestione per un VNF specifico (come monitoraggio, configurazione, ecc.);
- NFV Infrastructure (NFVI) estrae e gestisce l'accesso alle risorse fisiche. Include le risorse hardware, un livello di virtualizzazione per creare risorse virtuali sull'hardware disponibile e le risorse virtuali stesse;
- NFV Management and Orchestration (MANO) funge da coordinatore dell'intero sistema NFV. Comprende tre tipi di sottocomponenti: un orchestrator, che alloca e rilascia risorse dell'NFVI alle VNF, utilizzando il VIM, e gestisce il ciclo di vita dei servizi di rete (creazione, ridimensionamento, configurazione, aggiornamento, terminazione); i gestori VNF,

che vengono utilizzati per gestire il ciclo di vita delle VNF. Ogni VNF è collegata ad un gestore VNF; i Virtualized Infrastructure Manager (VIMs), che sono controllati da NFV Orchestrator e VNF Managers per gestire le risorse fisiche e virtuali in NFVI.

Il paradigma NFV si basa su tecnologie cloud e di virtualizzazione, che aggiungono maggiore complessità e nuovi rischi. È noto che i servizi di telecomunicazione dovrebbero essere altamente disponibili e, non appena si verifica un guasto o un'interruzione, devono essere recuperati entro un breve periodo di tempo, utilizzando mezzi di recupero automatico. Questi problemi sono affrontati da ETSI GS NFV-REL nel documento "Network Functions Virtualisation (NFV); Resiliency Requirements" [36], dove vengono identificati i casi d'uso, i requisiti e le architetture che servono da riferimento per le tecnologie emergenti NFV.

I VNF devono garantire che la qualità del servizio fornita sia la stessa delle funzioni di rete basate su hardware (ovvero reti legacy). Le istanze VNF possono essere ridimensionate per adattarsi a volumi elevati di traffico e per il ripristino da guasti. Per quanto riguarda i VNF con stato, richiedono meccanismi per l'archiviazione e il ripristino dello stato delle sessioni di rete e delle connessioni in modo affidabile. In particolare, ETSI identifica due condizioni principali che possono far deviare un servizio dal normale funzionamento: condizioni di congestione e condizioni di guasto. Le condizioni di congestione si verificano quando un volume insolito di traffico satura la capacità di VNF. Questa situazione può derivare da eventi speciali (ad es. Festival del Capodanno cinese, programmi TV, ecc.) o da attacchi informatici (ad es. Attacco DDoS). Al contrario, in condizioni di guasto, un servizio potrebbe essere interrotto o non disponibile a causa di componenti difettosi. Una VNF pu'o essere interconnessa con altre VNF attraverso delle interfacce ben definite per creare delle catene di funzioni, questa concatenazione viene definita Service Function Chain (SFC) [37].

NFV porta notevoli vantaggi, permette di ottenere una maggiore flessibilità ed agilità riguardo le capacità di lanciare e gestire servizi di rete. Le novità introdotte da NFV riguardano principalmente il disaccoppiamento del software dall'hardware, ciò permette di riassegnare e condividere le risorse dell'infrastruttura tra le varie funzioni di rete. Permette di configurare in modo dinamico le connessioni in un'infrastruttura e permette di instanziare qualsiasi tipo di funzione virtualizzata in ogni nodo compatibile con l'architettura NFV. Questo favorisce uno scalamento dinamico dovuto alla realizzazione delle funzioni mediante software e quindi all'allocazione dinamica delle risorse in funzione delle necessità.

3.1.2 Software-defined networking

Il paradigma Software Defined Network, noto anche come SDN, riguarda la gestione della rete in cui la preoccupazione principale è estrarre quanta più logica possibile dall'hardware. Il concetto chiave è la separazione della logica di controllo della rete dalle apparecchiature di rete che inoltrano e trasportano il traffico [38]. Quindi l'obiettivo principale è quello di disaccoppiare il control plane, ovvero la parte che si occupa della gestione delle regole di inoltro dei pacchetti, e il data plane ovvero l'effettivo flusso di tali pacchetti attraverso la rete. Disaccoppiando queste due entità è possibile avere un'estrema flessibilità nella gestione delle topologie di rete ed inoltre prevedere dei meccanismi automatici di configurazione, ad esempio basati su policy di alto livello. SDN crea un livello di controllo extra sopra l'infrastruttura fisica. Questo livello definisce alcune semplici regole per gli switch, centralizzando l'intelligenza di rete sul controller SDN. Gli switch comunicano con il piano di controllo per fornire statistiche ed informazioni nonché per ricevere istruzioni dal controller.

I principi fondamentali di tale paradigma sono:

- programmabilità: grazie al disaccoppiamento il controllo della rete può essere programmato in modo semplice ed eventualmente anche in modo automatico essendo separato dalle funzioni di inoltro;
- agilità: flessibilità nella gestione del traffico che permette agli amministratori di adattare repentinamente i piani di inoltro a seconda delle esigenze;

- gestione centralizzata: un solo controller ha la capacità di gestire una topologia complessa e tutti gli apparati, anche virtuali, al fine di ottenere un comportamento desiderato del flusso di pacchetti;
- basata su standard liberi: basandosi su standard aperti ed indipendenti dai fornitori, un singolo controller può gestire dispositivi differenti utilizzando istruzioni non specifiche del venditore.

Da tali principi viene definita l'architettura SDN, dove sono presenti tre strati diversi:

- application layer: questo livello è costituito da tutte le componenti applicative che servono a gestire il control plane ad un livello d'astrazione molto alto, questo livello ha poi la possibilità di interfacciarsi con il livello sottostante attraverso le API messe a disposizione dal protocollo SDN utilizzato ad esempio OpenFlow;
- control layer: questo livello è rappresentato dall' SDN Controller, anch'esso dipendente dalla tipologia di protocollo utilizzato, uno dei più quotati è OpenFlow. Il controller è il punto centrale dell'architettura SDN e comunica attraverso due interfacce, la northbound interface verso l'application Layer e la southbound interface verso l'infrastructure layer; rispettivamente una con il livello superiore per ottenere i comandi da uno o più applicativi e l'altra con il livello inferiore per innestare le regole all'interno dei dispositivi;
- infrastructure layer: questo livello contiene tutti i dispositivi che vengono programmati attraverso il controller sui quali sarà presente un agent, che può dipendere dal particolare hardware o meno e quindi essere eseguito su dispositivi standard. Questi agent ricevono i comandi dal controller, comandi che in sostanza non sono altro che delle regole di inoltro da applicare al dataplane.

L'API principale utilizzata nella southbound interface è OpenFlow [40], standardizzata da Open Networking Foundation [39] e supportata da molti dei principali fornitori di apparecchiature di rete (come IBM, NetGear, NEC, HP). La NBI è responsabile di fornire mezzi per specificare e richiedere politiche e servizi di rete in modo astratto, indipendentemente dal modo in cui sono attuati dal controllore.

Uno dei principali vantaggi di questa architettura è che le applicazioni che utilizzano la rete non hanno bisogno di conoscere la topologia né l'hardware per poterle eseguire. Il paradigma SDN può trarre vantaggi da NFV, se il controller fosse implementato come una funzione di rete, quindi come parte della catena di servizio, godrebbe di tutti i benefici tipici di una VNF come elasticità e scalabilità. Anche VNF potrebbe trarre vantaggio disponendo di SDN per la gestione del dataplane delle VNF, far si che le VNF della catena e la topologia della rete vengano gestite in modo dinamico da un SDN controller, quindi con tutti i vantaggi del disaccoppiamento tra data plane e control plane.

3.1.3 Network Security Functions

Con il termine funzione di rete, o Network function, si intende un qualsiasi blocco funzionale all'interno di un'infrastruttura di rete che ha interfacce esterne ben definite ed un comportamento funzionale ben definito [41]. Nell'ambito di questa tesi si tratta di funzioni di sicurezza di rete ossia Network Security Function (NSF). Con NSF si intende una qualsiasi funzione di sicurezza di rete. Il termine NSF viene definito da I2NSF nel lavoro "Interface to Network Security Functions (I2NSF) Terminology" come: "Software che fornisce una serie di servizi relativi alla sicurezza" [42]. Una NSF può aiutare a supportare l'integrità e la riservatezza del flusso di comunicazione, oppure il rilevamento di attività indesiderate e il blocco o la mitigazione dell'effetto di tali attività indesiderate al fine di soddisfare i requisiti di servizio. Una NSF è una funzione utilizzata per garantire integrità, riservatezza o disponibilità delle comunicazioni di rete; rilevare attività di rete indesiderate; o per bloccare, o almeno mitigare, gli effetti di attività indesiderate [43]. Le funzionalità di sicurezza sono definite anche come funzioni responsabili del trattamento specifico dei pacchetti ricevuti. Una funzione di sicurezza di rete può agire su vari livelli di uno stack di protocollo (ad esempio, a livello di rete o altri livelli OSI). Esempi più sofisticati di NSF di servizio possono essere:

- firewall;
- sistema di prevenzione / rilevamento delle intrusioni (IPS / IDS);
- Deep Packet Inspection (DPI);
- visibilità e controllo delle applicazioni (AVC);
- scansione di virus e malware di rete;
- sandbox;
- Data Loss Prevention (DLP);
- mitigazione DDoS (Distributed Denial of Service);
- proxy TLS.

3.2 Interface to Network Security Functions

Le NSF vengono fornite e utilizzate in ambienti sempre più differenti. Gli utenti possono utilizzare i servizi di sicurezza della rete applicati da NSF fornite da uno o più provider, che possono essere la propria azienda, i fornitori di servizi o una combinazione di entrambi. Allo stesso modo, i fornitori di servizi possono offrire ai propri clienti servizi di sicurezza di rete che sono applicati da più prodotti, funzioni di diversi fornitori o tecnologie open source. Le NSF possono essere fornite da infrastrutture fisiche e / o virtualizzate. Senza interfacce standard per controllare e monitorare il comportamento delle NSF, è diventato praticamente impossibile per i fornitori di servizi di sicurezza automatizzare le offerte di servizi che utilizzano diverse funzioni di sicurezza di più fornitori.

Non solo i clienti aziendali, ma anche quelli residenziali e mobili stanno diventando sempre più consapevoli della necessità della sicurezza della rete, solo per scoprire che i servizi di sicurezza sono difficili da operare e diventano costosi nel caso di quelli ragionevolmente sofisticati. Questa tendenza generale ha fatto sì che numerosi operatori e venditori di sicurezza iniziassero a sfruttare i modelli basati su cloud per fornire soluzioni di sicurezza. In particolare, i metodi relativi alla virtualizzazione delle funzioni di rete (NFV) hanno lo scopo di facilitare la distribuzione elastica di immagini software che forniscono i servizi di rete e richiedono la gestione di varie risorse da parte dei clienti, che potrebbero non possedere o ospitare fisicamente tali funzioni di rete [44].

L'obiettivo di I2NSF è definire una serie di interfacce software e modelli di dati per il controllo e il monitoraggio di aspetti di NSF realizzate in modo fisico e virtuale, consentendo ai clienti di specificare insiemi di regole da una o più entità di gestione [45].

La definizione di tali interfacce comporta numerosi vantaggi. Gli operatori potrebbero fornire servizi di sicurezza più flessibili e personalizzati per utenti specifici e ciò fornirebbe una protezione più efficiente e sicura per ciascun utente.

I2NSF specifica le interfacce a due livelli funzionali per il controllo e il monitoraggio delle funzioni di sicurezza della rete [45]:

- I2NSF Capability Level: Il livello di capacità di I2NSF specifica come controllare e monitorare le NSF a livello di implementazione funzionale. Il termine "Implementazione funzionale" è usato per sottolineare che le regole (per il controllo e il monitoraggio) delle NSF devono essere implementabili dalla maggior parte delle NSF. I2NSF standardizza un insieme di interfacce mediante le quali un controller di sicurezza può invocare, utilizzare e monitorare una NSF;
- I2NSF Service Layer: Il livello di servizio I2NSF definisce come le politiche di sicurezza dei clienti possano essere espresse ad un controllore di sicurezza. Il controllore implementa le sue politiche in base alle varie funzionalità fornite dal livello di capacità I2NSF. Il livello di servizio I2NSF consente inoltre al client di monitorare i criteri specifici del client.

Solitamente un client può sfruttare l'interfaccia del livello di servizio I2NSF per esprimere le politiche di sicurezza ad un controllore di sicurezza, che a sua volta interagisce con una o più NSF tramite l'interfaccia del livello di capacità I2NSF. In alternativa, un client può anche interagire direttamente con una o più NSF tramite l'interfaccia del livello di capacità I2NSF.

3.2.1 Casi d'uso I2NSF

Le crescenti sfide e complessità nel mantenere un'infrastruttura sicura, nel rispetto dei requisiti normativi e nel controllo dei costi stanno inducendo le aziende a utilizzare le funzioni di sicurezza della rete ospitate dai fornitori di servizi. Il servizio di sicurezza ospitato è particolarmente interessante per le piccole e medie imprese che soffrono della mancanza di esperti di sicurezza per monitorare continuamente le reti, acquisire nuove competenze e proporre misure immediate di mitigazione a serie sempre crescenti di attacchi alla sicurezza.

Secondo Gartner [46], la domanda di servizi di sicurezza ospitati (o basati su cloud) è in crescita. Le piccole e medie imprese (PMI) stanno adottando sempre più servizi di sicurezza basati su cloud per sostituire gli strumenti di sicurezza locali, mentre le grandi aziende stanno implementando un mix di servizi di sicurezza tradizionali e basati su cloud.

Per soddisfare la domanda, sempre più fornitori di servizi forniscono soluzioni di sicurezza ospitate per fornire servizi di sicurezza economicamente convenienti ai clienti aziendali. I servizi di sicurezza ospitati sono rivolti principalmente alle imprese (soprattutto di piccole e medie dimensioni) ma potrebbero anche essere forniti a qualsiasi tipo di cliente del mercato di massa. Di conseguenza, le funzioni di sicurezza di rete (NSF) vengono fornite ed utilizzate in un'ampia varietà di ambienti. Gli utenti che utilizzano NSF possono utilizzare i servizi di sicurezza della rete ospitati da uno o più provider, che possono essere la propria azienda, i fornitori di servizi o una combinazione di entrambi.

Esistono molti tipi di NSF. Le NSF di diversi fornitori possono avere diverse funzionalità ed interfacce. Le NSF possono essere distribuite in più posizioni in una determinata rete e possono avere ruoli diversi. Di seguito sono riportati alcuni esempi di funzioni di sicurezza e posizioni o contesti in cui vengono spesso implementati:

- protezione da intrusioni e attacchi esterni: esempi di questa funzione sono firewall / ACL, IPS, IDS;
- funzioni di sicurezza in zona demilitarizzata (DMZ): esempi di questa funzione sono firewall / ACL, IDS / IPS, uno o tutti i servizi AAA, NAT, proxy di inoltro e filtraggio delle applicazioni;
- funzioni di sicurezza centralizzate o distribuite: le funzioni di sicurezza potrebbero essere implementate in modo centralizzato per facilitare la gestione e la progettazione della rete o in modo distribuito per esigenze di ridimensionamento. Indipendentemente dalla modalità di distribuzione di una funzione di sicurezza, è preferibile avere la stessa interfaccia per la distribuzione delle politiche di sicurezza; in caso contrario, il compito dell'amministrazione della sicurezza è più complesso e richiede la conoscenza del firewall e della progettazione della rete;
- analisi e report sulla sicurezza interna: esempi di questa funzione sono i log di sicurezza, la correlazione degli eventi e l'analisi forense;
- protezione dei dati e dei contenuti interni: esempi di questa funzione sono la crittografia, l'autorizzazione e la gestione delle chiavi pubbliche / private per i database interni.

Data la diversità delle funzioni di sicurezza, i contesti in cui queste funzioni possono essere implementate e la costante evoluzione di queste funzioni, standardizzare tutti gli aspetti delle funzioni di sicurezza è una sfida e probabilmente non è fattibile. Fortunatamente, non è necessario standardizzare tutti gli aspetti. Ad esempio, dal punto di vista I2NSF, non è necessario standardizzare il modo in cui viene creato o applicato il filtro di ogni firewall. Alcune funzioni

del filtro di un fornitore specifico potrebbero essere univoche per il prodotto del fornitore, quindi non è necessario standardizzare queste funzionalità [43].

Ciò che serve è un'interfaccia standardizzata per controllare e monitorare i set di regole che le NSF usano per trattare i pacchetti che attraversano queste NSF. Pertanto, la standardizzazione delle interfacce fornirà uno slancio per standardizzare le funzioni di sicurezza stabilite.

Di seguito vengono riportati alcuni casi d'uso identificati dal lavoro di I2NSF [43]:

- framework di base: i servizi di sicurezza richiesti dall'utente tramite client specifici e l'entità di rete Network Service Provider (NSP) appropriata invocheranno le NSF in base alla richiesta di servizio dell'utente verso una entità chiamata controller di sicurezza (security controller). L'interazione tra queste entità (client, controller di sicurezza, NSF) è mostrata in Figura 3.1:

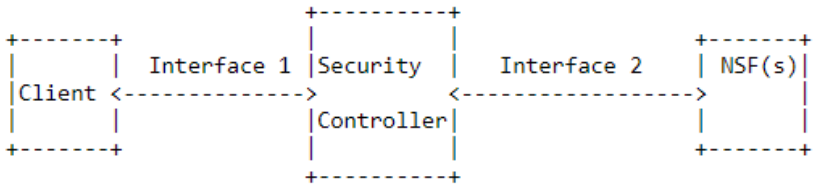

Figura 3.1. Modello I2NSF (fonte: "Use Cases and Requirements for an Interface to Network Security Functions" [44]).

L'interfaccia 1 viene utilizzata per ricevere i requisiti di sicurezza dal client e tradurli in comandi che le NSF possono comprendere ed eseguire. Inoltre, è anche responsabile di fornire al cliente feedback delle statistiche sulla sicurezza delle NSF. L'interfaccia 2 viene utilizzata per interagire con le NSF secondo i comandi e raccogliere informazioni sullo stato delle NSF.

I dispositivi o le applicazioni client possono richiedere al controller di sicurezza di aggiungere, eliminare o aggiornare le regole nella funzione del servizio di sicurezza per il loro traffico specifico.

Quando gli utenti desiderano ottenere lo stato di esecuzione di un servizio di sicurezza, possono richiedere lo stato della NSF dal client. Il controllore di sicurezza raccoglierà le informazioni riguardo la NSF tramite l'interfaccia 2, le consoliderà e fornirà un feedback al client attraverso l'interfaccia 1. Questa interfaccia può essere utilizzata per raccogliere non solo le informazioni sui singoli servizi, ma anche i dati aggregati adatti per attività come la valutazione della sicurezza dell'infrastruttura.

Gli utenti possono richiedere la convalida della disponibilità, della provenienza e dell'esecuzione di una NSF. Questo processo di convalida, particolarmente rilevante per le vNSF, include almeno:

  - integrità della NSF: garantire che la NSF non sia compromessa;
  - isolamento: assicurare che l'esecuzione della NSF sia autonoma per i requisiti di privacy in scenari *multi-tenancy*;
  - provenienza della NSF: potrebbe essere necessario fornire ai client garanzie rigorose sull'origine della NSF, sul suo stato (ad es. disponibile, inattivo, occupato e altri) e sui meccanismi di feedback in modo che un client possa essere in grado di verificare che una data NSF o set di NSF siano correttamente conformi ai requisiti del client e alle successive attività di configurazione.

A tal fine, il controllore della sicurezza può raccogliere misure di sicurezza e condividerle con una terza parte indipendente e affidabile (tramite l'interfaccia 1) al fine di consentire l'attestazione delle funzioni NSF utilizzando le informazioni aggiunte di terze parti. Ciò implica che potrebbero esserci i seguenti due tipi di client utilizzando l'interfaccia 1: l'utente finale e la terza parte indipendente ed affidabile. Questo determinerebbe che l'interfaccia 1 crei due interfacce secondarie per supportare questi due tipi di client;

- accesso alle reti: questo scenario descrive i casi d'uso per gli utenti (ad es. Utente residenziale, utente aziendale, utente mobile e sistema di gestione) che richiedono e gestiscono servizi di sicurezza ospitati nell'infrastruttura Network Service Provider (NSP). Dato che i client NSP sono essenzialmente utenti delle loro reti di accesso, lo scenario è essenzialmente associato alle loro caratteristiche e all'uso di vNSF.

  Questi casi d'uso definiscono l'interazione tra l'operatore e le vNSF attraverso interfacce automatizzate che supportano le comunicazioni a contratto tra cliente e fornitore o tra due entità commerciali. Client di accesso diversi possono avere richieste di servizio diverse:

  - residenziale: richieste di servizio per il controllo parentale, la gestione dei contenuti e la gestione delle minacce.

    La gestione dei contenuti delle minacce può includere l'identificazione e il blocco di attività dannose da contenuti Web, posta o file scaricati. La gestione delle minacce può includere l'identificazione e il blocco di botnet o malware;
  - azienda: richieste di servizio per politiche di sicurezza del flusso aziendale e servizi di sicurezza gestiti. Le politiche di sicurezza del flusso identificano e bloccano le attività dannose durante l'accesso a (o l'isolamento da) siti Web o applicazioni di social media. I servizi di sicurezza gestiti per un'azienda possono includere il rilevamento e l'attenuazione delle minacce esterne e interne. Le minacce esterne possono includere attacchi di applicazioni o phishing, malware, botnet, DDoS e altri;
  - fornitore di servizi: richieste di servizi per politiche che proteggono le reti dei fornitori di servizi da varie minacce (inclusi DDoS, botnet e malware). Tali politiche hanno lo scopo di fornire contenuti in modo sicuro e affidabile (ad es. Dati, voce e video) a vari clienti, inclusi clienti residenziali, mobili e aziendali. Queste politiche di sicurezza vengono inoltre applicate per garantire l'isolamento tra più tenant, indipendentemente dalla natura dei servizi di connettività corrispondenti;
  - mobile: richieste di servizio da interfacce che monitorano e garantiscono la qualità dell'esperienza dell'utente, la gestione dei contenuti, il controllo parentale e la gestione delle minacce esterne.

    La gestione dei contenuti per il dispositivo mobile include l'identificazione e il blocco di attività dannose da contenuti Web, posta e file caricati / scaricati. La gestione delle minacce per l'infrastruttura include il rilevamento e la rimozione di programmi dannosi come botnet, malware e altri programmi che creano attacchi DDoS).

  Ad alcuni client potrebbe non interessare quali NSF vengono utilizzate per ottenere i servizi richiesti. In questo caso, i sistemi di orchestrazione della rete del provider possono selezionare internamente le NSF (o vNSF) per applicare le politiche di sicurezza richieste dai client.

  Altri client, in particolare alcuni client aziendali, potrebbero voler contrattare separatamente per NSF dedicate (molto probabilmente vNSF) a fini di controllo diretto;

- data center cloud: in un data center, potrebbe essere necessario aggiungere o rimuovere dinamicamente meccanismi di sicurezza della rete come i firewall. Queste modifiche possono essere esplicitamente richieste dall'utente o attivate da un livello di richiesta prestabilito nell'accordo sul livello di servizio (Service Level Agreement, SLA) tra l'utente e il fornitore del servizio. Ad esempio, al fornitore di servizi potrebbe essere richiesto di aggiungere più capacità del firewall in un intervallo di tempo ogni volta che l'utilizzo della larghezza di banda raggiunge una determinata soglia per un periodo specificato. Questa espansione della capacità potrebbe comportare l'aggiunta di nuove istanze di firewall su macchine esistenti o la necessità di fornire un'istanza del firewall completamente nuova in una macchina diversa.

  La natura dinamica e su richiesta della fornitura di servizi di sicurezza incoraggia essenzialmente che i "dispositivi" di sicurezza della rete siano in software o in forme virtuali piuttosto che in una forma di dispositivo fisico. Questo requisito è una preoccupazione dal lato del provider. Gli utenti del servizio firewall sono agnostici (come dovrebbero essere) sul fatto che il servizio firewall sia eseguito o meno su una macchina virtuale o qualsiasi altra forma. In effetti, potrebbero anche non essere consapevoli del fatto che il loro traffico attraversa un firewall.

Inoltre, le nuove istanze del firewall devono essere collocate nella "zona giusta" (dominio). Il problema si applica non solo agli ambienti *multi-tenant* in cui è fondamentale il *tenant* nel dominio giusto, ma anche in ambienti di proprietà e gestiti da un'unica organizzazione con le proprie politiche di segregazione del servizio. Ad esempio, un'azienda può imporre che i firewall che servono il traffico Internet all'interno dell'organizzazione siano separati dal traffico tra organizzazioni. Un altro esempio sono i servizi IPS / IDS che dividono il traffico bancario di investimento da altro traffico di dati per conformarsi alle restrizioni normative per il trasferimento di informazioni bancarie di investimento.

  - distribuzione del firewall virtuale su richiesta: un data center cloud gestito da un fornitore di servizi potrebbe servire decine di migliaia di clienti. I server di calcolo dei client sono generalmente ospitati su macchine virtuali, che potrebbero essere distribuite su rack di server diversi situati in diverse parti del data center. Spesso non è tecnicamente e / o finanziariamente possibile implementare firewall fisici dedicati per soddisfare i requisiti delle politiche di sicurezza di ciascun cliente, che possono essere numerosi. Ciò che è necessario è la capacità di distribuire dinamicamente firewall virtuali per ogni insieme di server client sulla base di politiche di sicurezza stabilite e topologie di rete sottostanti;
  - firewall policy deployment automation: le regole del firewall si applicano al traffico solitamente identificato con indirizzi e porte. Diventa molto più complesso nelle reti cloud di proprietà dei provider che servono miriadi di clienti.

    Le regole del firewall oggi sono fortemente legate a porte e indirizzi che identificano il traffico. Ciò rende molto difficile per i clienti dei data center cloud costruire regole per il proprio traffico, poiché i client vedono solo le reti virtuali e gli indirizzi virtuali. Le reti e gli indirizzi virtuali visibili al cliente potrebbero essere diversi dai pacchetti effettivi che attraversano i firewall.

    Sebbene la maggior parte dei fornitori supporti funzionalità firewall simili, le parole chiave per la configurazione delle regole specifiche differiscono da fornitore a fornitore, il che rende difficile l'automazione. L'automazione funziona al meglio quando può sfruttare un insieme comune di standard che funzionerà su più NSF da più fornitori e utilizzerà la gestione dinamica delle chiavi;
  - politiche di sicurezza normative e di conformità: le organizzazioni devono proteggere le proprie reti dagli attacchi e devono anche aderire a vari regolamenti del settore: qualsiasi organizzazione che rientri in un regolamento specifico, come l'industria delle carte di pagamento - Standard di sicurezza dei dati (PCI-DSS) per il settore dei pagamenti o la legge sulla portabilità e responsabilità dell'assicurazione sanitaria per il settore sanitario deve essere in grado di isolare vari tipi di traffico. Devono inoltre mostrare i registri delle loro politiche di sicurezza ogni volta che vengono controllati.

    L'interfaccia lato client I2NSF potrebbe essere utilizzata per fornire politiche di sicurezza normative e relative alla conformità. Il responsabile della sicurezza terrà traccia di quando e dove viene applicata una politica specifica e se vi sono violazioni della politica; queste informazioni possono essere fornite in caso di controllo amministrativo come prova del fatto che il traffico è isolato tra endpoint specifici, nel pieno rispetto delle normative richieste;

- prevenzione di attacchi DDoS, malware e botnet: su Internet, dove tutto è collegato, la prevenzione del traffico indesiderato che può causare un attacco DoS o un attacco DDoS è diventata una sfida. Allo stesso modo, una rete potrebbe essere esposta ad attacchi malware e diventare una base di attacco che potrebbe compromettere il funzionamento di altre reti, ad esempio mediante comandi remoti. Molte reti che trasportano gruppi di informazioni (come reti di Internet of Things (IoT), reti basate su informazioni (ICN), reti di distribuzione di contenuti (CDN), reti di pacchetti Voice over IP (VoIP) e Voice over LTE (VoLTE)) sono anche esposti a tali attacchi remoti. Esistono molti esempi di attacchi remoti su queste reti. Un attacco mediante malware su una rete IoT che trasporta letture e istruzioni di un sensore può tentare di modificare le istruzioni del sensore al fine di disabilitare un sensore chiave. Un attacco mediante malware su reti VoIP o VoLTE coinvolge software che tenta

di effettuare chiamate interurbane non autorizzate. Le botnet possono sopraffare i nodi in ICN e CDN in modo che le reti non possano passare dati critici.

Per consentire alle organizzazioni di proteggere meglio le proprie reti da questo tipo di attacchi, il framework I2NSF dovrebbe fornire un'interfaccia lato client indipendente dal caso d'uso e indipendente dalla tecnologia. La tecnologia agnostica è definita generica, indipendente dalla tecnologia e in grado di supportare più protocolli e modelli di dati. Ad esempio, una tale interfaccia I2NSF potrebbe essere utilizzata per eseguire la fornitura delle informazioni di configurazione dei criteri di sicurezza che cercano indizi malware specifici. Allo stesso modo, gli attacchi botnet potrebbero essere facilmente prevenuti eseguendo la fornitura delle politiche di sicurezza utilizzando l'interfaccia lato client I2NSF che impedisce l'accesso ai server di comando e controllo botnet.

3.2.2 I2NSF framework

Il framework I2NSF consente l'utilizzo di NSF eterogenei sviluppati da diversi fornitori di soluzioni di sicurezza nell'ambiente NFV (Network Functions Virtualization) utilizzando le funzionalità di tali NSF tramite interfacce I2NSF.

I casi d'uso I2NSF esposti precedentemente ed ulteriori spiegati nell'elaborato "Interface to Network Security Functions (I2NSF): Problem Statement and Use Cases" [43], richiedono interfacce standard per gli utenti di un sistema I2NSF per informare il sistema su quali funzioni devono essere applicate a quale traffico (o schemi di traffico). Il sistema I2NSF lo realizza come un insieme di regole di sicurezza per il monitoraggio e il controllo del comportamento di traffico diverso. Fornisce inoltre interfacce standard per gli utenti per monitorare le funzioni di sicurezza basate sul flusso ospitate e gestite da diversi domini amministrativi.

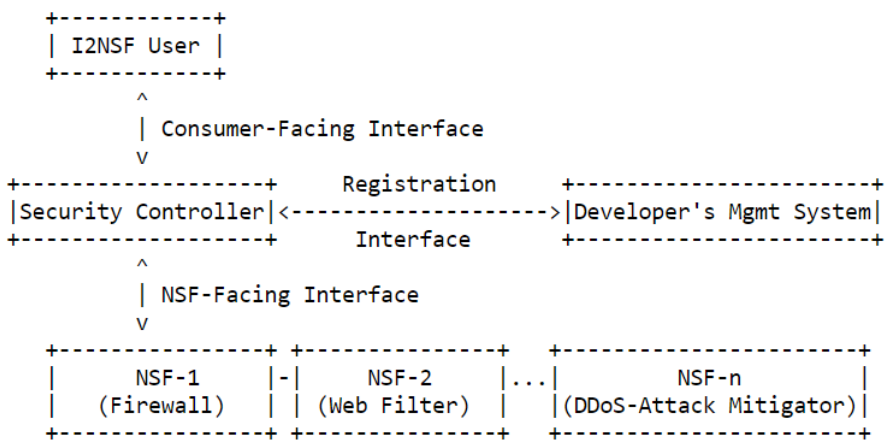

Figura 3.2. Modello I2NSF (fonte: "Framework for Interface to Network Security Functions" [47]).

Il modello di riferimento (compresi i principali componenti funzionali ed interfacce) per un sistema I2NSF è mostrato in Figura 3.2. Questa figura descrive il punto di vista degli operatori di rete verso il sistema di gestione; quindi, questa visione non assume alcuna particolare architettura di gestione né per le NSF né per come sono gestite le NSF dal lato dello sviluppatore.

I principi necessari per definire interfacce I2NSF sono i seguenti [47]:

- indipendenza dalla topologia di rete e dalla posizione delle NSF nella rete;
- indipendenza dal fornitore delle NSF, cioè indipendenza dal modo in cui il provider, rende disponibile la NSF, e dal modo in cui il provider consente la gestone della NSF;
- indipendenza da qualsiasi implementazione operativa, amministrativa e gestionale, specifica del fornitore, ambiente di hosting, e fattore di forma (fisico o virtuale);

- indipendenza dall'implementazione del piano di controllo della NSF;
- indipendenza dall'implementazione del piano dati della NSF.

In generale, tutte le interfacce I2NSF dovrebbero richiedere almeno l'autenticazione reciproca o l'autorizzazione di utenti esterni per il loro uso. Le interfacce presenti in Figura 3.2 sono:

- I2NSF Consumer-Facing Interface:

  l'interfaccia I2NSF per i consumatori viene utilizzata per consentire a diversi utenti di un determinato sistema I2NSF di definire, gestire e monitorare le politiche di sicurezza per flussi specifici all'interno di un dominio amministrativo. La collocazione e l'implementazione delle politiche I2NSF sono irrilevanti per gli utenti I2NSF.

  Alcuni esempi di utenti I2NSF includono:

  - un gestore di rete per videoconferenza che deve informare dinamicamente la rete di base per consentire, limitare la velocità o negare i flussi (alcuni dei quali sono crittografati) in base a campi specifici nei pacchetti per un determinato periodo di tempo;
  - amministratori di reti aziendali e sistemi di gestione che devono richiedere alla propria rete provider di applicare specifiche politiche I2NSF per flussi particolari;
  - un sistema di gestione IoT che invia richieste alla rete per bloccare i flussi che corrispondono a una serie di condizioni specifiche.

  Il modello di dati definito nel documento "I2NSF Consumer-Facing Interface YANG Data Model" [49], può essere utilizzato per l'interfaccia I2NSF Consumer-Facing. Si noti che un utente malintenzionato interno dell'utente I2NSF può utilizzare impropriamente il sistema I2NSF in modo che il sistema di rete nel sistema I2NSF sia vulnerabile agli attacchi di sicurezza. Per gestire questo tipo di minaccia, il Security Controller deve monitorare le attività di tutti gli utenti I2NSF e delle NSF tramite la funzionalità di monitoraggio I2NSF [48];

- NSF-Facing Interface:

  l'interfaccia I2NSF rivolata alle NSF (NSF-Facing Interface) viene utilizzata per specificare e monitorare le politiche di sicurezza basate sul flusso applicate da una o più NSF. Il sistema di gestione I2NSF può non utilizzare tutte le funzionalità di una determinata NSF, né deve utilizzare tutte le NSF disponibili. Quindi, questa astrazione consente alle funzioni NSF di essere trattate come elementi costitutivi di un sistema NSF; pertanto, gli sviluppatori sono liberi di utilizzare le funzioni di sicurezza definite da NSF indipendentemente dal fornitore e dalla tecnologia. Le NSF basate sul flusso ispezionano i pacchetti nell'ordine in cui vengono ricevuti. Tutti i gruppi di interfaccia richiedono che la NSF sia registrata utilizzando l'interfaccia di registrazione. L'interfaccia per le NSF basate sul flusso può essere classificata come segue:

  - interfaccia operativa ed amministrativa delle NSF: gruppo di interfacce utilizzato dal sistema di gestione I2NSF per programmare lo stato operativo dell'NSF; questo include anche funzioni di controllo amministrativo. Le I2NSF Policy Rule rappresentano un modo per modificare questo gruppo di interfacce in modo coerente;
  - interfaccia di monitoraggio: gruppo di interfacce utilizzato dal sistema di gestione I2NSF per ottenere informazioni di monitoraggio da una o più NSF selezionate. Ogni interfaccia in questo gruppo potrebbe essere basata su query o report. La differenza è che il sistema di gestione I2NSF utilizza un'interfaccia basata su query per ottenere informazioni, mentre NSF utilizza un'interfaccia basata su report per lo stesso scopo. Il sistema di gestione I2NSF può intraprendere una o più azioni in base alla ricezione delle informazioni; questo dovrebbe essere specificato da una I2NSF Policy Rule. Questo gruppo non modifica lo stato operativo di NSF.

Il modello di dati definito nel documento "I2NSF Network Security Function-Facing Interface YANG Data Model" [50], può essere utilizzato per l'interfaccia I2NSF NSF-Facing.

Con l'espressione "NSF basata sul flusso" si intende una NSF che controlla i flussi di rete in base a una politica di sicurezza. Sicurezza basata sul flusso significa anche che i pacchetti vengono ispezionati nell'ordine in cui vengono ricevuti e senza alterare i pacchetti a causa del processo di ispezione (ad es. Riscrittura del controllo di accesso medio (MAC), azione di decremento TTL o ispezione o modifiche NAT) [43].

- I2NSF Registration Interface:

  le NSF fornite da diversi fornitori possono avere capacità diverse. Al fine di automatizzare il processo di utilizzo di più tipi di funzioni di sicurezza offerte da diversi fornitori, è necessario disporre di un'interfaccia dedicata per i fornitori per definire (cioè registrare) le capacità delle loro NSF. Questa interfaccia è denominata I2NSF Registration Interface.

  Le funzionalità di una NSF possono essere preconfigurate o recuperate dinamicamente tramite l'interfaccia di registrazione I2NSF. Se una nuova funzione viene aggiunta ad una NSF, le capacità di quella nuova funzione devono essere registrate nel registro I2NSF tramite l'interfaccia di registrazione I2NSF, in modo che le entità di gestione e controllo interessate possano essere rese consapevoli della nuova registrazione. Questa interfaccia può essere utilizzata anche per la cancellazione di NSF.

  Il modello di dati definito nel documento "I2NSF Registration Interface YANG Data Model" [54], può essere utilizzato per l'interfaccia di registrazione I2NSF.

Come mostrato in Figura 3.2, un utente I2NSF può utilizzare le funzioni di sicurezza fornendo criteri di sicurezza di alto livello, che specificano i requisiti di sicurezza che l'utente I2NSF desidera applicare, al controllore tramite l'apposita interfaccia. Il Security Controller riceve ed analizza le politiche di sicurezza di alto livello da un utente I2NSF e identifica quali tipi di capacità di sicurezza sono necessarie per soddisfare queste politiche di alto livello. Il Security Controller quindi identifica le NSF che dispongono delle capacità di sicurezza richieste e genera politiche di sicurezza di basso livello per ciascuna delle NSF in modo che le politiche di sicurezza di alto livello vengano infine applicate da tali NSF. Infine, il Security Controller invia le politiche di sicurezza di basso livello generate alle NSF tramite l'interfaccia NSF-Facing Interface [50]. Il framework I2NSF può concatenare più NSF per implementare politiche di sicurezza di basso livello con l'architettura SFC [37].

Sebbene le funzioni di sicurezza presentino una varietà di fattori di forma e caratteristiche diverse, la disposizione per le NSF basate sul flusso può essere standardizzato utilizzando le Policy Rule.

Nella versione attuale di I2NSF, le Policy Rule sono limitate ai paradigmi imperativi. I2NSF utilizza una politica ECA (Event-Condition-Action), dove:

- una clausola Event viene utilizzata per attivare la valutazione della clausola Condition della regola I2NSF Policy Rule;
- una clausola Condition viene utilizzata per determinare se è possibile eseguire o meno il set di azioni nella regola I2NSF Policy Rule;
- una clausola Action definisce il tipo di operazioni che possono essere eseguite su questo pacchetto o flusso.

Ciascuna delle tre clausole precedenti è definita come clausole booleane. Ciò significa che ognuna è un'istruzione logica che valuta vero o falso.

3.2.3 I2NSF Information Model

Ogni NSF dovrebbe essere descritta con l'insieme di funzionalità che offre. Le capacità di sicurezza (security capability) consentono di descrivere le funzionalità di sicurezza in modo indipendente dal

fornitore. Cioè, non è necessario fare riferimento ad un prodotto o una tecnologia specifici durante la progettazione della rete; piuttosto, vengono considerate le funzioni caratterizzate dalle loro capacità. Questo permette di definire una descrizione inequivocabile di funzionalità di sicurezza offerte da una determinata NSF.

Un Capability Information Model (CapIM) è una formalizzazione delle funzionalità pubblicizzate da una NSF. Ciò consente la specifica di ciò che può fare una NSF in termini di applicazione delle politiche di sicurezza, in modo che le attività possano riferirsi, utilizzare, configurare e gestire in modo inequivocabile una NSF. Le capacità devono essere definite in modo indipendente dal fornitore e dalla tecnologia.

Il CapIM ha lo scopo di chiarire le ambiguità fornendo una descrizione formale della funzionalità di una NSF. L'insieme di funzioni pubblicizzate può essere limitato in base ai privilegi dell'utente o dell'applicazione che sta visualizzando tali funzioni. Le funzionalità I2NSF consentono una specifica inequivocabile delle capacità di sicurezza disponibili in un ambiente di rete (virtualizzato) e la loro elaborazione automatica.

Ciò include consentire al controllore di sicurezza di identificare e gestire correttamente le NSF e consentire alle NSF di dichiarare correttamente la loro funzionalità, in modo che possano essere utilizzate nel modo corretto.

Alcuni principi di progettazione di base per le capacità di sicurezza (security capability) e i sistemi che le gestiscono sono [51]:

- indipendenza: ogni funzionalità di sicurezza dovrebbe essere una funzione indipendente, con minima sovrapposizione o dipendenza da altre funzionalità. Ciò consente a ciascuna funzionalità di sicurezza di essere utilizzata ed assemblata insieme liberamente. Ancora più importante, le modifiche a una funzionalità non dovrebbero influenzare le altre funzionalità. Ciò segue il principio di responsabilità singola;
- astrazione: ogni capacità deve essere definita in modo indipendente dal venditore;
- pubblicità: un'interfaccia nota e dedicata deve essere utilizzata per pubblicizzare e registrare le capacità di ciascun NSF. Questa stessa interfaccia deve essere utilizzata da altri componenti I2NSF per determinare quali funzionalità sono attualmente disponibili per loro;
- esecuzione: un'interfaccia ben nota dedicata deve essere utilizzata per configurare e monitorare l'utilizzo di una funzionalità. Ciò fornisce una capacità standardizzata di descriverne la funzionalità e di riportarne i risultati di elaborazione. Ciò facilita l'interoperabilità multi-vendor;
- automazione: il sistema deve avere la capacità di scoprire, negoziare e aggiornare automaticamente le proprie capacità di sicurezza (cioè senza intervento umano). Queste funzionalità sono particolarmente utili per la gestione di un gran numero di NSF. Sono essenziali per l'aggiunta di servizi intelligenti al sistema di sicurezza impiegato. Queste funzioni sono supportate da molti modelli di progettazione, tra cui Observer Pattern, Mediator Pattern e una serie di Message Exchange Patterns;
- scalabilità: il sistema di gestione dovrebbe avere la capacità di scalare a seconda delle necessità, *scale up/down* oppure *scale in/out*. Pertanto, può soddisfare vari requisiti di prestazioni derivati dal traffico di rete modificabile o dalle richieste di servizio. Inoltre, le funzionalità di sicurezza interessate dalle modifiche alla scalabilità dovrebbero supportare la segnalazione delle statistiche al controllore di sicurezza per aiutare a decidere se deve invocare il ridimensionamento o meno.

Sulla base dei principi di cui sopra, I2NSF definisce un modello di capacità che consente a una NSF di registrare (e quindi pubblicizzare) la sua serie di capacità che altri componenti I2NSF possono utilizzare. L'insieme di funzionalità fornite da un determinato insieme di NSF definisce in modo inequivocabile la sicurezza offerta dall'insieme di NSF utilizzati. Il controller di sicurezza può confrontare i requisiti di utenti e applicazioni con l'insieme di funzionalità attualmente

disponibili al fine di scegliere quali funzionalità di quali NSF sono necessari per soddisfare tali requisiti.

Inoltre, quando una NSF segnala un evento sconosciuto, è possibile creare nuove funzionalità oppure aggiornare quelle esistenti. Ciò si traduce nel miglioramento delle NSF esistenti o nella creazione di nuove NSF per far fronte ai nuovi eventi. Le nuove funzionalità possono essere inviate ed archiviate in un repository centralizzato o archiviate separatamente nel repository locale di un fornitore. In entrambi i casi, un'interfaccia standard facilita il processo di aggiornamento.

Il lavoro I2NSF utilizza il modello di politica "Event-Condition-Action" (ECA), definito nel documento "Framework for Interface to Network Security Functions" [47], come base per la progettazione del modello di capacità. La struttura di una Policy Rule è definita da tre termini:

- evento: un evento è definito come qualsiasi occorrenza importante nel momento di una modifica nel sistema gestito e / o nell'ambiente del sistema gestito. Se utilizzato nel contesto delle Policy Rule I2NSF, viene utilizzato per determinare se la clausola Condizione della Policy Rule I2NSF può essere valutata o meno. Esempi di un evento I2NSF includono il tempo e le azioni dell'utente (ad es. accesso, disconnessione e azioni che violano un ACL).
- condizione: una condizione è definita come un insieme di attributi, caratteristiche e / o valori che devono essere confrontati con un insieme di attributi, caratteristiche e / o valori noti al fine di determinare se l'insieme di azioni in quella regola I2NSF può essere eseguita o meno. Esempi di condizioni I2NSF includono la corrispondenza degli attributi di un pacchetto o flusso e il confronto dello stato interno di un NSF con uno stato desiderato.
- azione: un'azione viene utilizzata per controllare e monitorare aspetti delle NSF basate sul flusso quando le clausole di evento e condizione sono soddisfatte. Le NSF forniscono funzioni di sicurezza eseguendo varie azioni. Esempi di azioni I2NSF comprendono la rilevazione e / o la protezione dalle intrusioni, il filtro web e dei flussi e l'ispezione approfondita dei pacchetti per pacchetti e flussi.

Una Policy Rule I2NSF è composta da tre clausole booleane: una clausola Event, una clausola Condition e una clausola Action. Questa struttura è anche chiamata ECA (Event-Condition-Action) Policy Rule. Una clausola booleana è un'istruzione logica che restituisce TRUE o FALSE. Può essere composta da uno o più termini; se è presente più di un termine, ciascun termine nella clausola booleana viene combinato utilizzando connettivi logici (ovvero, AND, OR e NOT). Una ECA Policy Rule I2NSF ha la seguente semantica:

```
IF <event-clause> is TRUE
  IF <condition-clause> is TRUE
    THEN execute <action-clause> [vincolato da metadati]
  END IF
END IF
```

Poiché possono esserci molti tipi di NSF che hanno molti tipi diversi di I2NSFSecurityCapabilities, la definizione di SecurityCapability deve essere effettuata utilizzando il contesto di una NSF. Questo è realizzato da una classe di associazione in UML. HasSecurityCapabilityDetail è una classe di associazione. Questo produce il design in Figura 3.3. Ciò consente alla classe di associazione HasSecurityCapabilityDetail di essere la destinazione di una Policy Rule. Cioè, la classe HasSecurityCapabilityDetail ha attributi e metodi che definiscono quali I2NSFSecurityCapabilities di questa NSF sono visibili e possono essere utilizzate.

Il concetto di una regola politica "abbinata", verificata, è definito come una regola in cui le sue clausole di evento e condizione vengono entrambe valutate come vere. Per descrivere con precisione cosa può fare una NSF in termini di sicurezza, le entità che devono essere descritte sono gli eventi che può catturare, le condizioni che può valutare e le azioni che può applicare.

Le proprietà definite da I2NSF nel drafit [51], che caratterizzano le capacità di un NSF sono le seguenti:

- Ac è l'insieme di azioni attualmente disponibili dalla NSF;

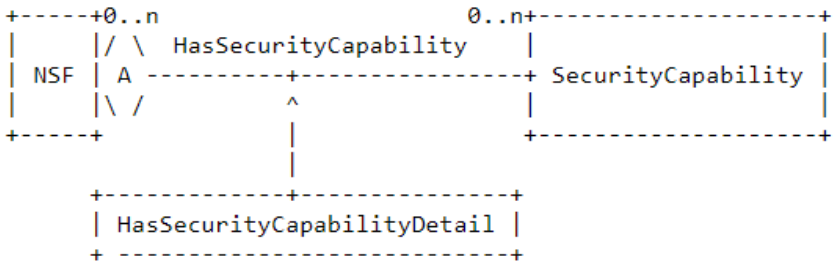

Figura 3.3. SecurityCapability di una NSF (fonte: "Information Model of NSFs Capabilities" [51]).

- Ec è il set di eventi che può essere catturato da una NSF. Si noti che per NSF (ad es. Un filtro pacchetti) che non sono in grado di reagire agli eventi, questo insieme sarà vuoto;
- Cc è l'insieme delle Condizioni attualmente disponibili dalla NSF;
- EVc definisce l'insieme delle Regole di valutazione delle clausole di condizione che possono essere utilizzate dalla NSF per decidere quando la clausola di condizione è vera, dato il risultato della valutazione delle singole Condizioni;
- RSc è l'insieme della strategia di risoluzione che può essere utilizzato per specificare come risolvere i conflitti che si verificano tra le azioni della stessa o diverse regole di politica che sono abbinate e contenute in questa particolare NSF;
- Dc definisce la nozione di un'azione Predefinita. Questa azione può essere un'azione esplicita che è stata scelta o un insieme di azioni.

Per caratterizzare il comportamento della Policy Rule come specificato da CapIM, una NSF può essere associata alle sue capacità di sicurezza mediante una 6-tupla Ac, Cc, Ec, RSc, Dc, EVc, dove Ac, Cc, Ec , RSc, Dc ed EVc sono stati descritti nelle sezioni precedenti.

Ad esempio, supponendo di definire una capacità di filtro di pacchetti, Cap_pf, le sue capacità possono essere definite come:

Cap_pf = (Apf, Cpf, Epf, RSpf, Dpf, EVpf)

dove:

Apf = {Consenti, Nega}

Cpf = {IPsrc, IPdst, Psrc, Pdst, protType}

Epf = {}

RSpf = {FMR}

Dpf = {A1}

EVpf = {DNF}

3.3 SUPA

Il progetto Simplified Use of Policy Abstractions, SUPA [52], definisce un modello informativo per rappresentare le politiche usando un framework estensibile che è indipendente da linguaggio, protocollo, repository e livello di astrazione del contenuto e del significato di una qualsiasi politica. Ciò consente di mappare un insieme comune di concetti definiti nel modello informativo SUPA in diverse rappresentazioni della politica (ad esempio rappresentazioni procedurali, imperative e dichiarative). Consente inoltre a diversi modelli di dati che utilizzano lingue, protocolli e repository diversi di ottimizzarne il loro utilizzo. La definizione di politiche comuni fornisce anche una

migliore interoperabilità garantendo che ciascun modello di dati possa condividere un insieme di concetti comuni, indipendentemente dal suo livello di dettaglio o dalla lingua, dal protocollo e / o dal repository che sta utilizzando. È inoltre indipendente dal modello di dati di destinazione che verrà generato.

Il modello informativo definito da SUPA si concentra sulla definizione di un tipo di regola politica, ossia la regola evento-condizione-azione (Event-Condition-Action ECA). Di conseguenza, definisce due serie di elementi del modello:

1. un framework per la definizione del concetto di politica, indipendentemente dal modo in cui la politica è rappresentata o utilizzata; questo è chiamato SUPA Generic Policy Information Model (GPIM);
2. un framework per la definizione di un modello di policy che utilizza il paradigma event-condition-action; questo si chiama SUPA ECA Policy Rule Information Model (EPRIM) ed estende i concetti del GPIM.

La combinazione di GPIM ed EPRIM fornisce un framework estensibile per la definizione di criteri che utilizzano una rappresentazione ECA indipendente dal repository di dati, dal linguaggio di definizione dei dati, dal linguaggio delle query, dal linguaggio di implementazione e dal protocollo.

SUPA definisce un'interfaccia per una funzione di gestione della rete che accetta Policy di alto livello come criteri di input, possibilmente a livello di rete, e crea frammenti di configurazione degli elementi come output. SUPA risponde alle esigenze degli operatori e degli sviluppatori di applicazioni per rappresentare molteplici tipi di Policy Rule, che variano nel livello di astrazione, per soddisfare le esigenze dei diversi attori.

3.3.1 SUPA GPIM

Lo scopo del SUPA Generic Policy Information Model (GPIM) è definire un framework comune per esprimere le politiche a diversi livelli di astrazione. SUPA utilizza GPIM come vocabolario comune per rappresentare concetti di politica indipendenti da lingua, protocollo, repository e livello di astrazione. Questo permette a diversi attori di creare ed utilizzare politiche a diversi livelli di astrazione. Ciò costituisce un modo di fare politico, in cui politiche più astratte possono essere tradotte in politiche più concrete e viceversa.

La maggior parte dei sistemi definisce la nozione di politica come un'unica entità. Ciò presuppone che tutti gli utenti della politica abbiano la stessa terminologia e utilizzino la politica allo stesso livello di astrazione. Questo è raramente vero nei sistemi moderni. SUPA definisce GPIM come un vocabolario standard e un insieme di concetti che consentono a diversi attori di utilizzare diverse formulazioni di politiche. Potrebbe essere necessario tradurre una politica da una forma generale a una forma più specifica (mantenendo lo stesso livello di astrazione).

Il GPIM definisce i seguenti elementi:

- una classe che definisce il vertice della gerarchia delle classi GPIM, chiamata SUPAPolicyObject;
- quattro sottoclassi di SUPAPolicyObject, che rappresentano:
  - la parte superiore della gerarchia relativa alle PolicyRule, chiamata SUPAPolicyStructure;
  - la parte superiore della gerarchia dei componenti relativi alla PolicyRule, chiamata SUPAPolicyComponentStructure;
  - PolicySource, che rappresenta l'insieme di entità che hanno creato o sono responsabili della SUPAPolicy;
  - PolicyTarget, che definisce l'insieme di entità a cui viene applicata la SUPAPolicy.

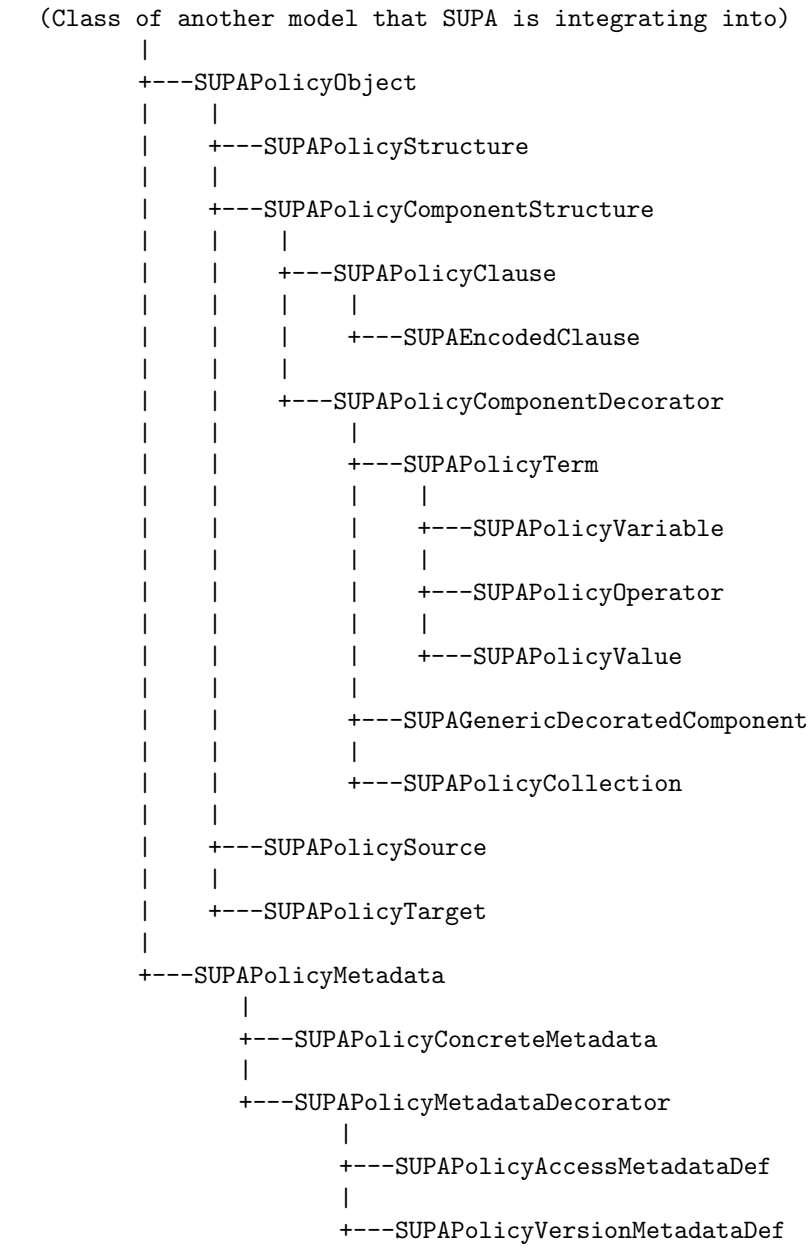

Figura 3.4. SUPA GPIM tree diagram.

La gerarchia delle classi definita per SUPA GPIM è rappresentata in Figura 3.4; per i dettagli relativi alle classi ed alle relazioni tra esse consultare il draft del progetto SUPA "Generic Policy Information Model for Simplified Use of Policy Abstractions (SUPA)" [52].

GPIM potrebbe essere utilizzato senza EPRIM. Tuttavia, per utilizzare EPRIM, è necessario utilizzare anche GPIM.

3.3.2 SUPA EPRIM

Concettualmente SUPA ECA Policy Rule Information Model, EPRIM, è un insieme di sottoclassi che specializzano le nozioni definite nel contesto GPIM per rappresentare i componenti di una politica che utilizza la semantica ECA. Nella Figura 3.5 sono mostrate le sottoclassi EPRIM relativamente alle loro superclassi GPIM.

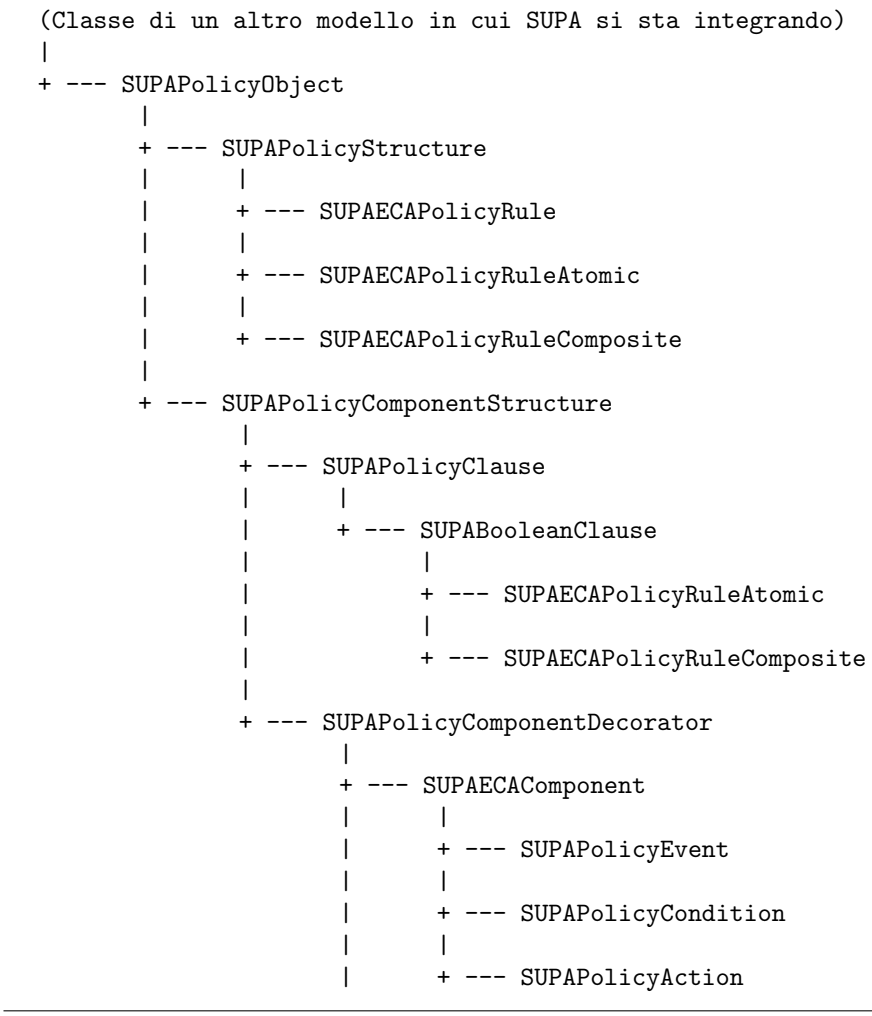

Figura 3.5. SUPA EPRIM tree diagram.

In particolare, EPRIM specifica delle nuove classi che specializzano la classe SUPAPolicyStructure per creare una regola denominata SUPAECAPolicyRule;e specifica due sottoclassi della classe SUPAPolicyComponentStructure per creare due nuove serie di componenti della Policy. Le due sottoclassi di SUPAPolicyComponentStructure sono:

- una nuova sottoclasse di SUPAPolicyClause, chiamata SUPABooleanClause, definita per la costruzione di clausole booleane specifiche per le esigenze delle regole della politica ECA;
- una nuova sottoclasse di SUPAPolicyComponentDecorator, chiamata SUPAECAComponent, definita per la costruzione di oggetti riutilizzabili che rappresentano Eventi, Condizioni e Azioni.

EPRIM fornisce nuove funzionalità, basate su GPIM, estendendo GPIM per definire nuove classi e relazioni. L'EPRIM non definisce nuove classi che non sono ereditate dalle classi GPIM esistenti. Ciò garantisce che la semantica di GPIM non venga modificata, anche se vengono definite nuove funzionalità per le regole e i componenti della politica ECA.

La strategia generale per affinare GPIM è la seguente:

- SUPAECAPolicyRule è definita come una sottoclasse della classe GPIM SUPAPolicyStructure;
- una regola SUPAECAPolicy ha clausole di evento, condizione e azione;

  - concettualmente, questo può essere visto come tre aggregazioni tra la SUPAECAPolicyRule e ciascuna clausola;
  - ogni aggregazione utilizza un'istanza di una sottoclasse concreta di SUPAPolicyClause; questo può essere un SUPABooleanClause (che lo rende specifico per ECA), un SUPAEncodedClause (che lo rende di natura generica) o una nuova sottoclasse di SUPAPolicyClause;
  - le sottoclassi concrete di SUPAPolicyClause possono essere decorate con zero o più sottoclassi concrete della classe SUPAPolicyComponentDecorator;

- è possibile definire un insieme opzionale di oggetti SUPAPolicySource GPIM per rappresentare la creazione di una SUPAECAPolicyRule;
- è possibile definire un insieme opzionale di oggetti SUPAPolicyTarget GPIM per rappresentare l'insieme di entità gestite che saranno interessate da questa regola SUPAECAPolicyRule;
- è possibile definire un insieme opzionale di SUPAPolicyMetadata per tutti gli oggetti che compongono una SUPAECAPolicyRule, inclusi i suoi componenti.

Esistono diversi modi per costruire una SUPAECAPolicyRule; questi differiscono a seconda di quale gruppo di componenti viene utilizzato per definire il contenuto di SUPAECAPolicyRule e se ciascun componente è decorato o meno. Di seguito sono riportati alcuni esempi di creazione di una SUPAECAPolicyRule:

- definire tre tipi di clausole SUPABoolean, una ciascuna per le clausole di evento, condizione e azione che compongono una regola SUPAECAPolicyRule;
- per una o più delle clausole precedenti, associare un insieme appropriato di oggetti SUPAPolicyEvent, SUPAPolicyCondition o SUPAPolicyAction e completare la clausola utilizzando un SUPAPolicyOperator appropriato e un SUPAPolicyValue o SUPAPolicyVariable corrispondente;
- le clausole booleane composte possono essere formate utilizzando uno o più oggetti SUPABooleanClauseComposite con uno o più oggetti SUPABooleanClauseAtomic;
- definire un componente SUPAPolicyCollection, utilizzato per aggregare un insieme di oggetti appropriato per una clausola, e completare la clausola utilizzando un SUPAPolicyOperator appropriato e un SUPAPolicyValue o SUPAPolicyVariable corrispondente;
- creare una nuova sottoclasse concreta di SUPAPolicyComponentStructure (ovvero una classe di pari livello di SUPAPolicyComponentDecorator e SUPAPolicyClause) ed utilizzare questa nuova sottoclasse in una sottoclasse concreta di SUPABooleanClause; questo approccio consente di decorare facoltativamente anche la nuova sottoclasse di SUPAPolicyComponentStructure;
- creare una nuova sottoclasse di SUPAPolicyComponentDecorator che fornisce funzionalità specifiche ECA e utilizzarla per decorare un SUPAPolicyClause;
- creare una nuova sottoclasse concreta di SUPAPolicyStructure che fornisce funzionalità specifiche dell'ECA e definire tutto o parte del suo contenuto aggregando un insieme di SUPAPolicyClauses.

Una regola SUPAECAPolicy è un tipo di SUPAPolicy. È una tupla che deve avere tre clausole, definite come segue:

- la clausola evento definisce un'espressione booleana che, se vera, avvia la valutazione della sua clausola condition (se la clausola event non è vera, non ha luogo alcuna ulteriore azione per questa Policy Rule);

- la clausola condition definisce un'espressione booleana che, se vera, consente di eseguire le azioni nella clausola action (se la clausola condition non è vera, non ha luogo alcuna ulteriore azione per questa Policy Rule);
- la clausola action contiene una serie di azioni. Un'azione può invocare un'altra SUPAECA-PolicyRule.

Ciascuna delle clausole di cui sopra può essere una semplice espressione booleana (della forma valore dell'operatore variabile o un'espressione booleana composta costituita da combinazioni booleane di clausole. Le espressioni booleane composte dovrebbero essere in CNF o DNF. Notare che ognuna delle tre precedenti clausole possono avere un insieme di oggetti SUPAPolicyMetadata che possono limitare o influenzare il modo in cui viene trattata tale clausola.

Poiché un SUPABooleanClause è una sottoclasse di un SUPAPolicyClause, può essere decorata da una o più sottoclassi concrete di SUPAPolicyComponentDecorator. Pertanto, una SUPAECAPolicyRule può essere costruita interamente da oggetti definiti in GPIM ed EPRIM.

Capitolo 4

Progettazione e design della soluzione

In questo capitolo verrà trattato nello specifico la definizione del problema al quale si vuole dare una soluzione in questa tesi, si mostreranno alcuni casi d'uso sui quali mi sono basato per ideare la soluzione e per lo sviluppo dei tool. Inoltre verrà introdotto il concetto di linguaggio astratto per controlli di sicurezza e verrà presentata la progettazione della soluzione.

4.1 Definizione del problema

L'obiettivo di I2NSF è definire una serie di interfacce software e modelli di dati per il controllo e il monitoraggio di aspetti di NSF realizzate in modo fisico e virtuale, consentendo ai clienti di specificare insiemi di regole da una o più entità di gestione. Da questo concetto la mia tesi si concentra sul problema della rappresentazione dei modelli di dati astratti di funzionalità di sicurezza di rete. Il problema che scaturisce l'interesse della mia tesi riguarda la gestione dei modelli di dati che rappresentano funzionalità di sicurezza di rete. I modelli di dati basati su Capability Information Model (CapIM) sono sufficientemente espressivi per descrivere NSF reali. Infatti partendo dal modello informativo realizzato nel framework di base definito da I2NSF in Figura 3.3, la tesi propone la definizione del modello dei dati ed un ampliamento della capacità espressiva di tale modello per poter far fronte al problema della generazione di linguaggi astratti di NSF e la successiva traduzione delle policy. Si presenta la necessità di definire un modello di dati che rappresenti, in modo generico, le security capability per servizi di sicurezza di rete (capacità di sicurezza), sufficientemente espressivo per poter rappresentare le potenzialità dei dispositivi che offrono funzionalità di sicurezza, e poter confrontare tali dispositivi per, ad esempio, verificare se possono esprimere le stesse politiche di sicurezza. Il CapDM include tutte le funzionalità che possono essere utilizzate per assegnare capacità ad una NSF seguendo il paradigma del decorator pattern.

Oltre quanto fin ora detto sorge il problema della generazione di linguaggi astratti per controlli di sicurezza per permettere la definizione di policy astratte per NSF con servizi di rete. Si evince che ogni NSF ha un linguaggio generico per la definizione di policy di sicurezza. Il linguaggio generico di una NSF è comunque specifico per ogni NSF in funzione delle capacità di sicurezza di rete a lei assegnate. Dato che il mantenimento di linguaggi astratti associati a ciascuna NSF è difficile, ingombrante e richiede tempo è sorto il problema di dover definire un procedimento automatico per derivare il linguaggio di configurazione astratto di una NSF partendo dalla descrizione delle sue capacità di sicurezza di rete. Si è riscontrato il problema di dover associare un meccanismo di trasformazione automatico in base alle istanze del CapDM. Tale trasformazione automatica necessita di un supporto per il perfezionamento automatico delle configurazioni astratte scritte nel modello linguistico specifico della NSF in impostazioni di configurazione concrete applicabili dalla NSF. Conseguentemente servono criteri o regole per definire la semantica e la sintassi che permettano la traduzione delle policy espresse nel linguaggio generico della NSF nel linguaggio di basso livello relativo alla NSF stessa.

4.2 Generazione di linguaggi astratti per controlli di sicurezza

Con l'espressione linguaggio astratto per controlli di sicurezza si intende la definizione di un linguaggio di alto livello, generico, che permetta di esprimere concetti di sicurezza concreti con una sintassi comune.

Un linguaggio astratto per controlli di sicurezza dovrebbe soddisfare i seguenti requisiti:

- Astrazione: il linguaggio deve contenere configurazioni di sicurezza astratte, indipendenti da un determinato fornitore o rappresentazione e archiviazione specifiche del prodotto. Il motivo di questo requisito è che la semantica di configurazione è indipendente dalla rappresentazione effettiva. Ovvero, le stesse impostazioni di configurazione possono essere rappresentate e applicate in diversi controlli di sicurezza;
- Diversità: deve supportare la descrizione delle configurazioni per una varietà di funzioni di sicurezza. Il metamodello di configurazione deve inoltre supportare la configurazione di tali capacità di sicurezza, che seguono criteri e concetti diversi e sono applicati a diversi tipi di controlli di sicurezza;
- Flessibilità ed estensibilità: deve essere sufficientemente flessibile ed estensibile per supportare l'introduzione di nuovi controlli di sicurezza;
- Continuità: deve garantire la continuità della catena di politiche, permettendo la traduzione delle impostazioni di controllo di sicurezza. Ciò è utile per tenere traccia di quale criterio è effettivamente applicato nella policy.

Un linguaggio generico per controlli di sicurezza viene definito per astrarre i linguaggi di configurazione con un formato indipendente dal fornitore e dal sistema di controllo, organizzato in base alle capacità; ad esempio un firewall di un determinato fornitore può implementare la funzionalità di filtraggio dei pacchetti nel suo set di capacità, mentre un altro fornitore può implementare anche la capacità di ispezione dei pacchetti nel suo firewall. Sfortunatamente, definire questa astrazione non è banale perché ogni controllo di sicurezza ha una sintassi specifica. Pertanto la mappatura può essere ingestibile in una sintassi generica e il linguaggio generico deve essere organizzato in base alle funzionalità di sicurezza. Una funzionalità di sicurezza è una funzionalità di base offerta da un controllo di sicurezza. Pertanto, un linguaggio generico per controlli di sicurezza è organizzato secondo un modello generale che definisce i concetti di alto livello (politiche, regole, condizioni, azioni, ecc.) ed un insieme di sottomodelli per acquisire i concetti specifici della semantica come attributi, tipi di condizioni, metodi, ecc.. D'altra parte, un linguaggio di configurazione specifico del controllo dipende dai formati e dalle funzioni disponibili nel controllo di sicurezza effettivo: ogni funzionalità di sicurezza definisce il proprio linguaggio di configurazione secondo le sue capacità di sicurezza.

La generazione di linguaggi astratti permette di uniformare il metodo col quale si possono esprimere le policy di sicurezza di rete.

Per definire il formato generico per esprimere il linguaggio astratto della NSF sono stati valutate diverse tecnologie espressive quali XML e YANG. La scelta effettuata in questa proposta di soluzione è stata di utilizzare il linguaggio XML, essendovi numerosi vantaggi.

Il primo vantaggio è che il linguaggio XML è predisposto per la definizione di linguaggi di markup definibili come XMLSchema (XSD), questo permette la generazione di linguaggi in funzione delle capacità che sono state attribuite ad una specifica NSF. Utilizzando il framework open source modelio per la rappresentazione grafica del Capability Information Model e del Capability Data Model, il linguaggio XML risulta compatibile immediatamente dato che nel framework modelio è presente una funzionalità che permette di esportare il modello nel formato XMI adattabile in modo semplice ed immediato al formato XML. Una proprietà fondamentale per la generazione di linguaggi astratti per controlli di sicurezza è la possibilità di definire in modo dinamico le caratteristiche semantiche del linguaggio stesso, utilizzare il linguaggio XML permette di definire, mediante appositi tool, una struttura fissa dipendente dalle caratteristiche semantiche del linguaggio stesso. Inoltre il linguaggio XML è largamente conosciuto nell'ambito informatico.

4.3 Casi d'uso

La definizione di casi d'uso permette di individuare più facilmente l'idea del problema e di iniziare a pensare alle soluzioni. Il caso d'uso più comune che si esamina è il seguente:

- Sorge la necessità di inserire una policy di sicurezza nella rete. Non si conosce il linguaggio specifico di basso livello con cui i vari servizi di sicurezza di rete possono esprimere tale policy, ma si è in grado di riconoscere quali sono le capacità di sicurezza necessarie ed assegnare loro i valori richiesti. La tesi propone una soluzione che permetta di offrire la possibilità di utilizzare un linguaggio generico per esprimere la policy necessaria e quindi di avere uno strumento che la traduca nel linguaggio specifico della NSF scelta o dell'insieme delle NSF necessarie per applicare tale policy.

I casi d'uso seguenti vengono presentati nel contesto di I2NSF. In questo contesto è presente un controller che comprende le policy di alto livello e provvede a cercare tra le istanze di NSF. Il controller infatti ha la capacità di interrogare le NSF per scoprire quale di queste possono gestire le funzionalità di sicurezza che compongo la policy di alto livello. Inoltre si suppone la presenza di un NSFStore per mantenere NSF conosciute e disponibili ma senza istanze.

- Avendo una policy di alto livello che richiede l'applicazione di alcuni requisiti di sicurezza si identificano le funzionalità necessarie per applicare i requisiti di alto livello. Successivamente si può agire con due differenti strategie:
  1. viene cercata nella rete NFV attualmente distribuita una NSF che possegga tali capacità, viene generata la nuova politica per tale NSF e dunque viene tradotta nella configurazione effettiva con il linguaggio specifico della NSF;
  2. si effettua una ricerca nel NSFStore per identificare eventuali NSF che posseggano tali capacità di sicurezza, si decide dove posizionare tale NSF, si genera la politica per tale NSF selezionata secondo il loro linguaggio astratto e si traduce la politica nella configurazione effettiva con il linguaggio specifico della NSF;
- Avendo una rete NFV configurata, c'è una NSF che è istanza di iptables. Il controller si accorge che le prestazioni di iptables non sono sufficienti per far rispettare gli standard richiesti, quindi è necessario cambiarlo con una istanza di NSF più performante. iptables è configurato secondo delle determinate policy. Il controller cerca nel NSFStore se ci sono NSF diverse da iptables che possano applicare la stessa politica di iptables. Trovata la nuova NSF si deve generare il criterio per la nuova NSF utilizzando il linguaggio astratto di quest'ultima ed inviarne la configurazione. La presenza di un linguaggio generico per esprimere le policy permette l'interscambio di policy espresse nel linguaggio di NSF che possiedano tutte le capacità di sicurezza necessarie per la determinata policy. Conseguentemente, utilizzando la nuova NSF, viene tradotta la policy nella configurazione con il linguaggio specifico della NSF scelta.

4.4 Progettazione e design della soluzione

In questo paragrafo viene presentata la progettazione ed il design della soluzione ai problemi precedentemente esposti. Inoltre verrà esposta l'architettura che è stata ideata e prodotta.

Per prima cosa viene affrontato il problema della realizzazione di un modello informativo delle capacità di sicurezza (Capability Information Model, CapIM) che possa essere sufficientemente espressivo per descrivere NSF reali, il CapIM verrà approfondito nella Sezione 5.1. Per poter descrivere NSF generiche il lavoro di tesi propone una visione di NSF astratta alla quale vengono assegnate capacità di sicurezza a seconda della funzionalità di sicurezza desiderata. Per ottenere questo risultato è stato applicato il paradigma del decorator pattern, che permette di assegnare caratteristiche in modo dinamico. Per esprimere al meglio la rappresentazione del CapIM, tale modello viene rappresentato secondo il paradigma orientato agli oggetti utilizzando il linguaggio

Unified Modeling Language UML. Il linguaggio UML permette la raffigurazione di classi generiche utilizzando il diagramma delle classi che mette in mostra sia l'esistenza delle relative classi sia le relazioni presenti tra di esse.

Per poter descrivere le caratteristiche da assegnare ad una NSF il lavoro di tesi propone una rappresentazione delle funzionalità di sicurezza dei servizi di rete. Questa raffigurazione genera un modello di dati delle capacità di sicurezza ovvero il Capability Data Model (CapDM), che verrà approfondito nella Sezione 5.2. Le stesse considerazioni fatte per il CapIM hanno portato alla scelta che per la realizzazione del CapDM si utilizzasse lo stesso metodo rappresentativo.

Il CapDM proposto è un diagramma delle classi dove viene esposta la rappresentazione della gerarchia di capacità di sicurezza seguendo determinate relazioni di parentela. Lo scopo del CapDM è di rappresentare in modo generico ogni singola parte che compone una funzionalità di sicurezza, chiamata capacità di sicurezza (security capability), specificandone in modo generico i dettagli implementativi della stessa. Il CapDM proposto raggruppa le capacità di sicurezza in insiemi definiti in base alle caratteristiche delle capacità stesse.

La gerarchia utilizzata per la rappresentazione delle capacità di sicurezza parte dalla divisione nella 6-tupla definita da I2NSF, vista nella Sezione 3.2, composta da:

- evento;
- condizione;
- azione;
- azione predefinita;
- strategia di risoluzione;
- criterio di valutazione.

Nel medesimo modello sono stati rappresentati anche gli attributi per ogni capacità di sicurezza per poterne definire una rappresentazione unica e generica, e quindi per permette la creazione di un linguaggio generico comune. Gli attributi sono stati specificati come opzionali per permettere di formare un elenco di capacità di sicurezza generiche durante la fase di assegnazione delle stesse ad una determinata NSF. I requisiti delle rappresentazioni del CapDM e del CapIM sono l'alta astrazione e la genericità, per questo motivo nessun modello presentato è condizionato da specifiche realizzazioni o influenzato da particolari produttori di funzionalità di sicurezza. Questo consente di supportare teoricamente diverse implementazioni di NSF alle quali sono assegnate mediante decorator pattern le capacità di sicurezza generiche. Per la rappresentazione grafica del CapIM e del CapDM sono stati presi in considerazione molteplici ambienti grafici per la realizzazione del diagramma delle classi, la scelta finale è stata quella di utilizzare l'ambiente di modellazione estendibile ed open source modelio. Questo ambiente permette la gestione di diagrammi utilizzando il linguaggio UML in modo semplice ed intuitivo. Modelio è una applicazione autonoma basata su Eclipse RCP e questo lo rende uno strumento compatibile con l'ambiente di programmazione Eclipse.

Successivamente per affrontare il problema della rappresentazione delle policy generiche la tesi propone uno strumento per la creazione del linguaggio generico della NSF. Il linguaggio generico della NSF è l'insieme delle rappresentazioni generiche delle sue capacità di sicurezza, al cui interno si trova la semantica utile per esprimere le eventuali policy astratte. Il linguaggio della NSF proposto viene realizzato seguendo l'elenco delle specifiche capacità di sicurezza che le sono state assegnate.

Per esprimere al meglio e rendere generico il linguaggio della NSF occorre utilizzare un metalinguaggio che permetta la definizione di ulteriori linguaggi. Valutando i possibili metalinguaggi il linguaggio XML è stato reputato la scelta migliore, perchè consente la definizione di linguaggi di markup, ovvero un linguaggio basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento. Inoltre permette la generazione di un linguaggio che è sia leggibile dall'uomo che leggibile dalla macchina, indipendente

dalle tecnologie circostanti e con alta capacità di espressione. Di conseguenza uno degli strumenti proposti permette la generazione di un file contenente il linguaggio astratto della NSF in formato XSD. Questo permette la generazione di politiche basate su grammatica senza contesto.

Lo strumento proposto per la generazione del linguaggio si basa sulle rappresentazioni del CapIM e del CapDM, infatti gestisce la generazione del linguaggio in funzione della descrizione delle capacità di sicurezza assegnate e di eventuali vincoli specifici definiti alle capacità, utilizzando un approccio model driven.

Infine è stato affrontato il problema della traduzione delle policy dal linguaggio generico della NSF al linguaggio specifico di basso livello della stessa. Per proporre una soluzione a questo problema è stato necessario uno studio delle tipologie dei file di configurazione specifici di servizi di sicurezza esistenti, come si vedrà nella Sezione 6.2, che analizza il caso specifico di iptables. Dallo studio delle strutture dei file di configurazione sono stati decisi i parametri che possono permettere al tool di personalizzare la struttura di uscita del file contenente la policy di basso livello. Lo strumento proposto utilizza alcune informazioni contenute nel CapIM, infatti, la traduzione dal linguaggio astratto della NSF al linguaggio di basso livello deve tenere conto della semantica e della sintassi proprie dell'istanza della NSF per la quale si vuole generare il file di configurazione.

La scelta dell'ambiente di programmazione è arrivata di conseguenza, in quanto essendo modelio fortemente legato ad eclipse, è stata presa la decisione di sviluppare i tool del progetto della tesi utilizzando un linguaggio gestibile sia da modelio che da eclipse. Per questo motivo i tool proposti sono realizzati utilizzando il linguaggio JAVA. I vantaggi della scelta del linguaggio di programmazione JAVA sono anche relativi all'alta compatibilità nella gestione dei file in formato XML, infatti esistono API che permettono la gestione ottimizzata di questi ultimi, quali JAXB e JAXP.

Un ulteriore motivo che ha portato alla scelta di queste tecnologie consiste nel fatto che nell'ambiente di sviluppo modelio è integrata la funzionalità di esportazione del modello UML in formato XMI. Il primo tool che è stato necessario realizzare, è un tool di conversione da linguaggio XMI a linguaggio XML. Mediante questo tool è stata realizzata la rappresentazione del CapDM in formato XMLSchema.

Inoltre è stato realizzato un tool funzionale che permette la validazione di un file XML rispetto ad uno schema di riferimento.

L'architettura comprendente degli elementi ed alcuni dei passaggi più significativi sono rappresentati in Figura 4.1.

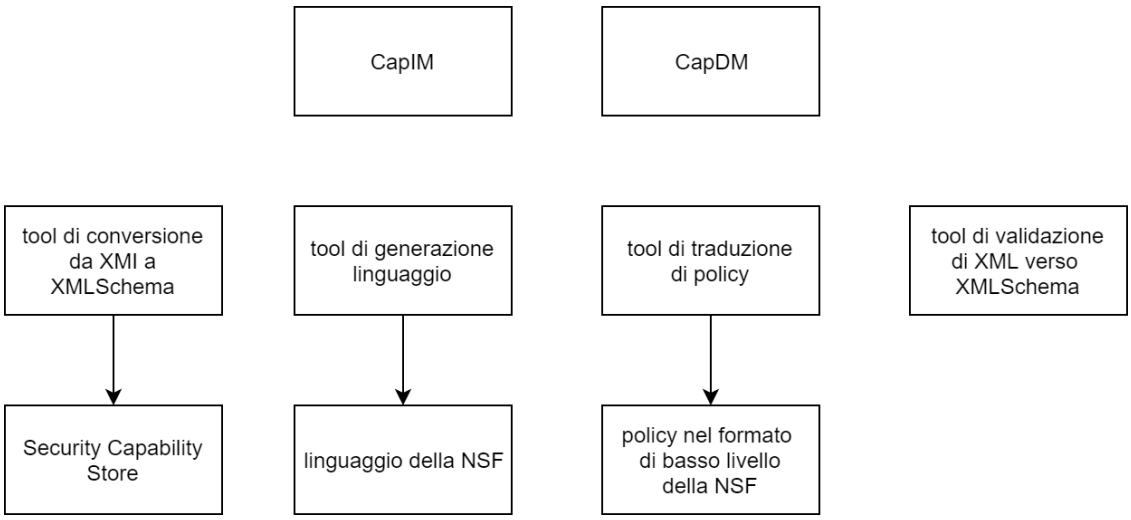

Figura 4.1. Architettura della soluzione.

Di seguito una breve spiegazione delle entità rappresentate in figura 4.1:

- CapIM: il Capability Information Model è il modello di riferimento per lo sviluppo della soluzione, esso rappresenta le entità principali che sono coinvolte nello studio della tesi. Questo modello viene realizzato con l'ausilio dello strumento scelto modelio per la modellazione delle classi nel relativo diagramma delle classi;
- CapDM: il Capability Data Model è il modello di riferimento dove sono rappresentate gerarchicamente le capacità di sicurezza, realizzato con l'ausilio dello strumento scelto modelio per la modellazione delle classi nel relativo diagramma delle classi;
- tool di conversione XMI a XMLSchema: per poter utilizzare e validare un insieme di capacità di sicurezza da assegnare ad una NSF è necessario ottenere un elenco di capacità di sicurezza valide. Questo tool permette la generazione di un insieme di capacità valide nel formato XSD sulla base delle capacità modellate nel CapDM;
- Security Capability Store: nella soluzione proposta questo elemento rappresenta l'insieme delle capacità di sicurezza che possono essere assegnate ad una NSF generica;
- tool di generazione del linguaggio: questo tool permette la generazione del linguaggio astratto della NSF;
- linguaggio della NSF: entità contenente le caratteristiche semantiche che permettono la generazione di politiche di sicurezza;
- tool di traduzione della policy: questo strumento permette la traduzione di una policy generata con il linguaggio della NSF nel linguaggio di basso livello della NSF;
- tool di validazione di XML verso XMLSchema: tool che permette la validazione di un file XML rispetto al suo XMLSchema;

La realizzazione di ogni tool è pensata ed adattata per un utilizzo da linea di comando, da questo si ottiene il vantaggio di poterne automatizzare l'utilizzo.

La Figura 4.2 aggiunge i dettagli delle relazioni tra gli elementi rappresentati nell'architettura della soluzione in Figura 4.1, aggiungendo anche le relazioni necessarie per i tool sviluppati al loro utilizzo.

La Figura 4.2 viene presentata seguendo un flusso logico delle azioni nel tempo che si possono compiere.

- Il CapDM realizzato con modelio permette la realizzazione, mediante un tool predisposto da modelio stesso, di esportare il modello di riferimento in formato XMI.
- Questo file viene utilizzato in input dal tool di conversione da XMI ad XMLSchima, permettendo di generare un file XSD, contenente tutte le capacità di sicurezza rappresentate nel CapDM, nella Figura chiamato Security Capability Store.
- Quest'ultimo file permette l'assegnazione delle capacità di sicurezza ad una NSF generica, ottenendo così un file XML contenente la lista di capacità di sicurezza per la determinata NSF.
- Tale file deve essere una istanza valida rispetto al file XSD che rappresenta il Security Capability Store, per ottenerne la certezza è stato sviluppato un tool di validazione tra XMLSchema e file XML.
- Il tool di generazione del linguaggio della NSF riceve in input un file XML contenente la lista delle capacità di sicurezza valido e il file XSD che rappresenta il Security Capability Store, elaborando questi due file viene generato il linguaggio generico per controlli di sicurezza della NSF.
- Utilizzando il linguaggio generico della NSF si possono istanziare la policy per la NSF scelta. Il file che rappresenta la policy deve essere un file XML valido rispetto al linguaggio della NSF.

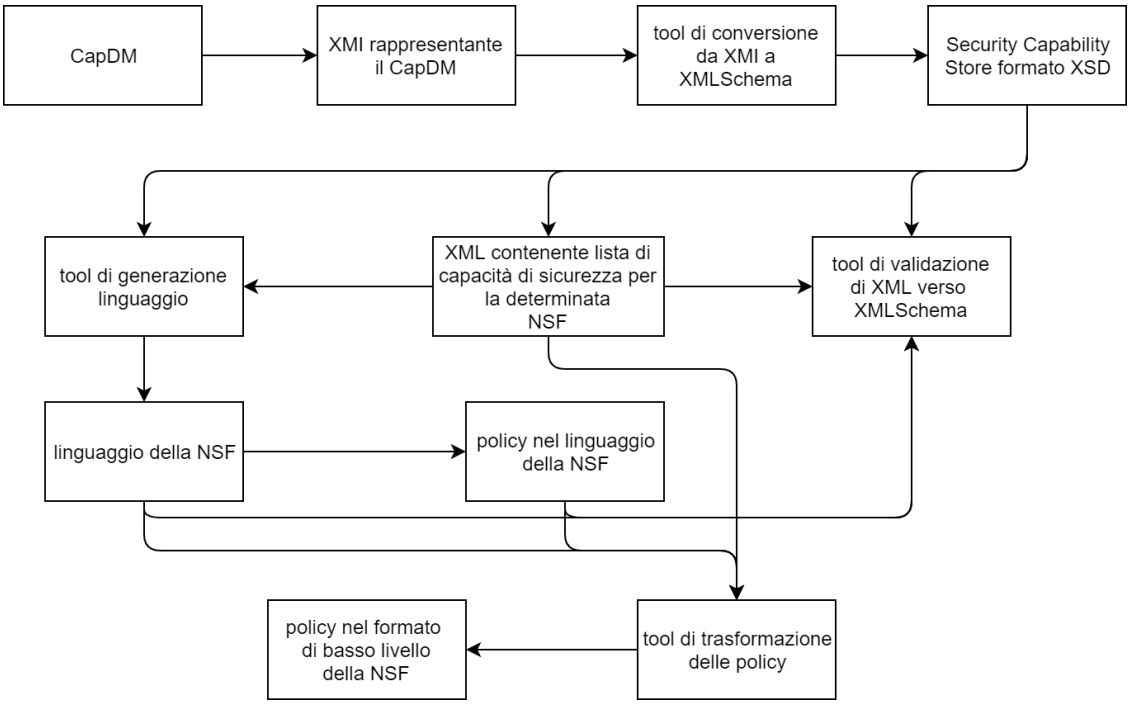

Figura 4.2. Relazioni tra gli elementi dell'architettura.

- Il tool di trasformazione delle policy riceve in input la policy nel linguaggio della NSF, il linguaggio della NSF e l'istanza contenente l'elenco delle capacità di sicurezza della NSF, nel cui file sono presenti i criteri di traduzione di ogni capacità. Questo tool genera come otuput un file TXT contenente la policy tradotta.

Capitolo 5

Modello delle capacità di sicurezza

In questo capitolo si mostrerà la modellazione delle capacità di sicurezza e della struttura della soluzione relative al modello informativo ed al modello dei dati.

5.1 Information Model

Un modello informativo permette di rappresentare e modellare oggetti gestiti a livello concettuale, indipendente da qualsiasi circostanza e tecnologia al contorno. I modelli informativi si concentrano sulla rappresentazione delle entità e funzionalità astratte di un ambiente; mettono in evidenza le relazioni tra gli oggetti interessati e le dinamiche relative alle entità. Possono avere differenti gradi di dettaglio delle astrazioni definite in funzione delle esigenze dei progettisti e nascondono i dettagli implementativi e della realizzazione delle entità rappresentate.

I2NSF definisce nel framework di base il modello informativo per una NSF come in Figura 3.3, dove vengono definiti i seguenti oggetti:

- NSF, entità che rappresenta una qualsiasi NSF;
- SecurityCapability, entità che rappresenta una qualsiasi capacità di sicurezza;
- HasSecurityCapabilityDetail, entità che rappresenta i dettagli della relazione tra una SecurityCapability e la relativa NSF.

In tale modello informativo sono rappresentate le seguenti relazioni:

- aggregazione HasSecurityCapability tra NSF e SecurityCapability, relazione che indica l'appartenenza di zero o più capacità di sicurezza a zero o più NSF;
- classe di associazione HasSecurityCapabilityDetail, relazione utilizzata per rappresentare l'entità HasSecurityCapabilityDetail appartenente ad una relazione di una capacità di sicurezza ad una NSF.

Resi noti i problemi sorti ed esposti nel capitolo precedente è quindi ideata una risoluzione, sulla base del CapIM definito da I2NSF, la tesi propone il CapIM rappresentato in Figura 5.1.

Nella figura 5.1 sono definiti i seguenti oggetti:

- NSF, entità che rappresenta una qualsiasi NSF;
- SecurityCapability, entità che rappresenta una qualsiasi capacità semplice di sicurezza;
- HasSecurityCapabilityDetail, entità che rappresenta i dettagli della relazione tra una capacità di sicurezza e la relativa NSF;

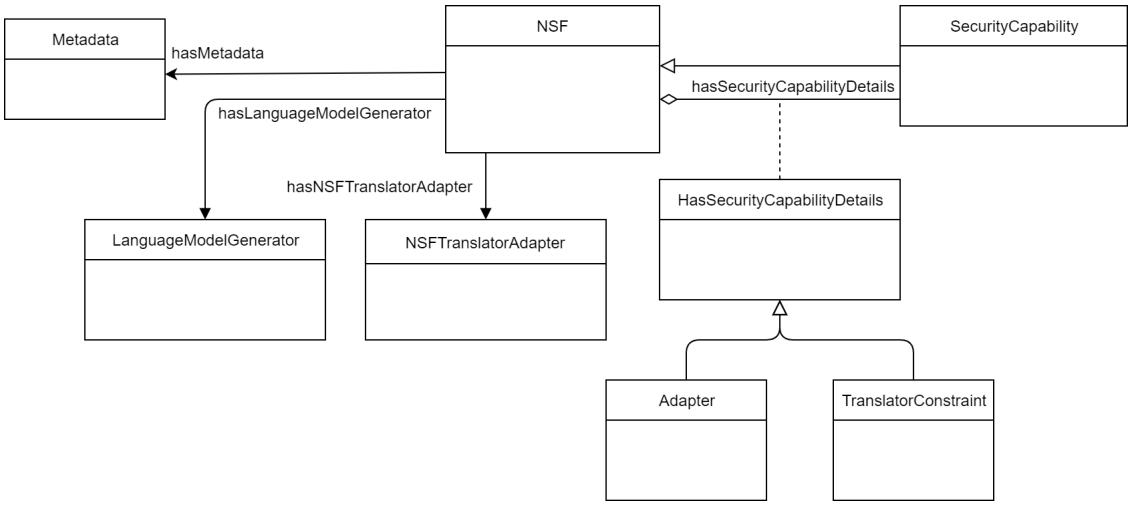

Figura 5.1. CapIM.

- LanguageModelGenerator, entità che rappresenta la generazione del linguaggio della NSF istanziata;
- NSFTranslatorAdapter, entità che rappresenta la traduzione della policy nel linguaggio di basso livello della NSF istanziata;
- Metadata, entità che rappresenta eventuali metadati della NSF, ad esempio l'elenco dei nomi dei protocolli, e relativo numero associato, conosciuti dalla NSF;
- LanguageItemAdapter, entità che eredita da HasSecurityCapabilityDetail e quindi rappresenta dettagli specifici dati dalla relazione di una capacità di sicurezza verso una NSF riguardanti la creazione del linguaggio della NSF;
- TranslatorConstraint, entità che eredita da HasSecurityCapabilityDetail e quindi rappresenta dettagli specifici dati dalla relazione di una capacità di sicurezza verso una NSF, riguardanti la traduzione delle policy dal linguaggio generico della NSF al linguaggio di basso livello della stessa.

Nel modello informativo proposto sono rappresentate le seguenti relazioni:

- relazione hasMetadata, relazione che permette alla NSF di possedere metadati, con cardinalità indefinita per permettere la relazione di più metadati corrispondenti a più NSF;
- relazione hasLanguageModelGenerator, relazione generica verso il tool che permette la generazione di un linguaggio astratto della NSF;
- relazione hasNSFTranslatorAdapter, relazione generica verso il tool che permette la trasformazione di policy espresse nel linguaggio generico della NSF producendo un file di configurazione adattato alle caratteristiche della NSF;
- aggregazione hasSecurityCapability tra NSF e SecurityCapability, relazione che indica l'appartenenza di una istanza SecurityCapability ad una istanza di NSF;
- classe di associazione HasSecurityCapabilityDetail, per rappresentare l'entità HasSecurityCapabilityDetail e di conseguenza di LanguageItemAdapter e di TranslatorConstraint, appartenenti ad una relazione di una SecurityCapability ad una NSF;
- eredità tra NSF e SecurityCapability, relazione che permette, assieme all'aggregazione HasSecurityCapability, di applicare il decorator pattern.

Il modello informativo presentato nella tesi è incentrato nella gestione di NSF, infatti rappresenta come elemento principale la NSF, come viene messa in relazione con le capacità di sicurezza e come interagisce all'assegnazione di policy. Il modello permette di descrivere una NSF e di assegnarne funzionalità di sicurezza dinamicamente mediante decorator pattern. Tra una NSF ed una capacità di sicurezza esiste una relazione definita come classe di associazione, nella quale si possono definire le caratteristiche necessarie per la gestione della generazione del linguaggio della NSF e per l'adattamento al linguaggio di basso livello dell'istanza della NSF. Una NSF può essere considerata come una qualsiasi funzionalità di sicurezza alla quale vengono assegnate capacità di sicurezza che permettono di descrivere la sua funzionalità; nell'ambito della tesi le capacità di sicurezza sono descritte nel Capability Data Model proposto.

5.2 Data Model

Rispetto ai modelli informativi, i modelli di dati definiscono gli oggetti gestiti a un livello inferiore di astrazione. Sebbene i modelli informativi e i modelli dei dati abbiano scopi diversi, non è sempre facile decidere quali dettagli appartengano al primo e quali appartengano al secondo. Allo stesso modo, a volte è difficile determinare se un'astrazione appartiene ad un modello informativo o ad un modello di dati. Sono destinati agli implementatori ed includono dettagli specifici relativi all'implementazione, ad esempio regole che spiegano come mappare oggetti gestiti su costrutti di livello inferiore oppure definire la struttura stessa delle varie entità. Poiché i modelli informativi possono essere implementati in diversi modi, è possibile derivare più modelli di dati da un singolo modello informativo.

Il modello dei dati proposto nel lavoro di tesi utilizza la suddivisione delle SecurityCapability in una 6-tupla come definito da I2NSF:

- evento: capacità di riconoscere un evento, il verificarsi di un fatto;
- condizione: capacità di valutare se i presupposti sono verificati;
- azione: capacità di effettuare un comando;
- azione predefinita: azione utilizzata di default;
- strategia di risoluzione: metodologia secondo la quale vengono valutate le regole rispetto al verificarsi di un evento;
- criterio di valutazione: metodologia secondo la quale vengono avverate le parti della regola.

La differenza rispetto alla definizione di I2NSF che si nota nella Figura 5.2 è la presenza di una relazione di aggregazione tra la classe corrispondente all'azione predefinita (classe DefaultActionCapability) e la classe relativa alle azioni (classe ActionCapability). Questa relazione permette di definire una sola volta la capacità di compiere una azione e di poter utilizzare la stessa definizione per l'azione predefinita. Quindi l'insieme delle azioni predefinite può essere uguale all'insieme delle azioni oppure un sottoinsieme di esso.

Il CapDM proposto sviluppa ampiamente le classi di condizione e di azione. A partire dalla 6-tupla ed utilizzando l'ereditarietà UML sono state definite ulteriori gerarchie di sottoclassi che permettono di differenziare l'ambito di cui si occupa la capacità di sicurezza che si sta considerando.

Per rendere il concetto più chiaro viene riportato di seguito un esempio pratico: pensando ad una NSF che rappresenta un packet filter, essa avrà la capacità di riconoscere determinati campi di un pacchetto. Ad esempio alla NSF viene assegnata la capacità di riconoscere il tipo di protocollo su cui sta viaggiando un determinato pacchetto.

Nel CapDM proposto questa capacità è stata mappata come "IpProtocolTypeCapability" come in Figura 5.3. Quest'ultima fa parte di un gruppo più ampio di capacità di sicurezza che è stato chiamato "Ipv4Capability" per definire l'ambito di appartenenza della stessa. La

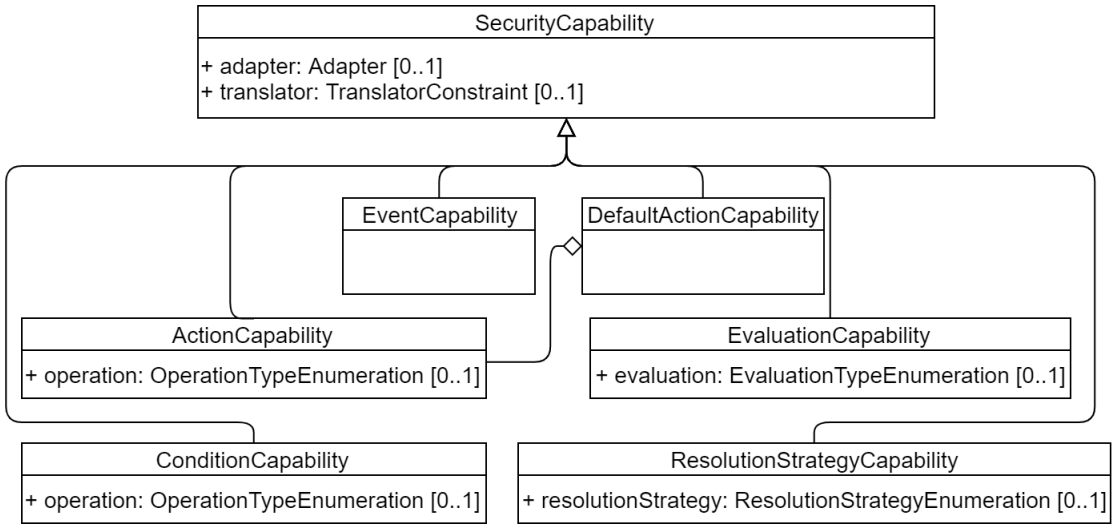

Figura 5.2. 6-tupla.

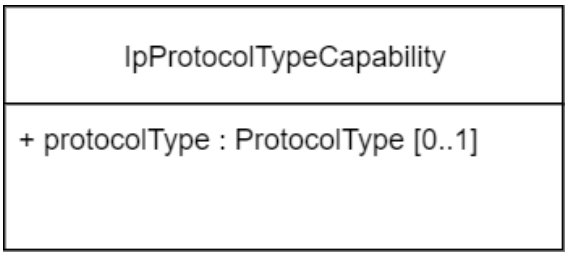

Figura 5.3. IpProtocolTypeCapability.

classe “Ipv4Capability” eredita direttamente dalla classe “ConditionCapability” appartenente alla 6-tupla precedentemente descritta.

Nella definizione delle “SecurityCapability” di tipo condizione e di tipo azione sono stati rappresentati eventuali attributi. La definizione degli attributi nel CapDM permette la standardizzazione dei nomi e dei tipi degli attributi stessi. Come si può notare dalla Figura 5.3, la capacità di sicurezza “IpProtocolTypeCapability” è stata definita con l’attributo “protocolType”. Quest’ultimo attributo è di tipo “ProtocolType”, che a sua volta è stato definito in un diagramma delle classi utilizzato appositamente per la definizione dei tipi non standard, quindi non tipi come “string” o “integer”. La Figura 5.4 mostra la definizione del tipo dell’attributo preso in esempio.

Dalla definizione del tipo preso in esempio si può notare la presenza di due attributi utilizzati per esprimere il tipo di protocollo, questo è necessario per permettere la dichiarazione di un protocollo sia utilizzando il nome sia utilizzando il numero corrispondente. Il valore relativo al campo “protocolTypeName” è stato anch’esso standardizzato utilizzando una enumerazione contenente i valori in formato stringa dei principali protocolli, come mostra la Figura 5.5 che riporta una porzione dell’enumerazione proposta.

Quanto sopra spiega la modalità con la quale si è giunti alla definizione delle capacità di sicurezza di rete e le capacità di sicurezza relative alle azioni rappresentate nel CapDM applicabili ad istanze di NSF.

Il CapDM proposto contiene la definizione degli eventuali attributi per ogni capacità di sicurezza. La definizione degli attributi nel CapDM comporta il vantaggio di standardizzare i nomi ed i tipi con cui ci si riferisce per generare le istanze di capacità di sicurezza; in questo modo si ottiene genericità ed omogeneità nel riferirsi ad istanze di tali capacità. Un ulteriore vantaggio viene evidenziato quando si genera il linguaggio astratto della NSF, perchè avendo già definito i nomi ed i tipi che permettono le istanze, i medesimi si possono direttamente utilizzare. Infatti,

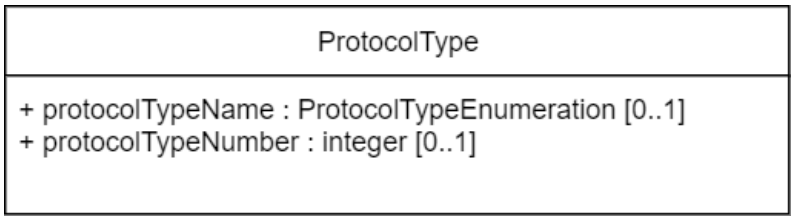

Figura 5.4. ProtocolType.

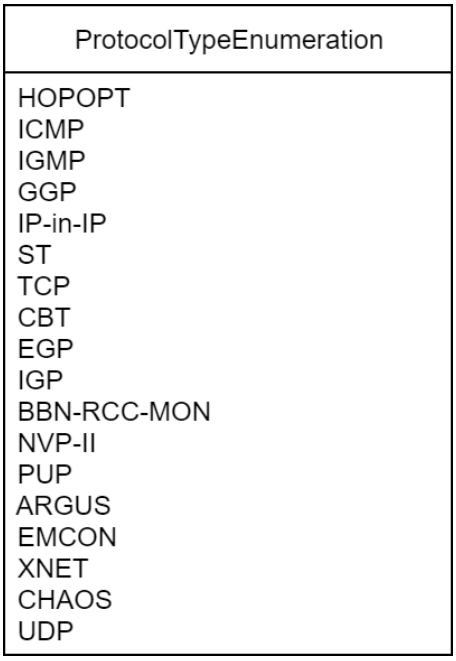

Figura 5.5. ProtocolTypeEnumeration.

istanze diverse di NSF, ma con capacità di sicurezza in comune, avranno lo stesso linguaggio astratto per quelle determinate capacità di sicurezza. La cardinalità degli attributi definiti nel CapDM presenta come valore minimo 0, perchè questo permette di non dover attribuire dei valori agli attributi durante la fase di assegnazione delle capacità di sicurezza ad una NSF.

5.3 Classe LanguageConstraint

Per affrontare il problema della generazione di un linguaggio astratto della NSF, la tesi propone uno strumento che sfrutta il meccanismo di Model-driven transformation. Questa trasformazione permette di passare dalla generica rappresentazione di capacità di sicurezza, definite nel CapDM, alla specifica definizione del linguaggio astratto della NSF, alla quale sono state assegnate determinate capacità di sicurezza con la possibilità, nel caso, di applicare dettagli per la personalizzazione. Infatti, la classe "LanguageConstraint" viene proposta come sottoclasse della classe di associazione definita da I2NSF "HasSecurityCapabilityDetail"; questo permette di rendere esplicita la capacità a cui appartiene e la NSF di contesto. La struttura che governa la generazione del linguaggio della NSF proposta dalla tesi è rappresentata in Figura 5.6. A seconda dei valori specificati nella classe "LanguageConstraint" durante l'assegnazione delle capacità di sicurezza, lo strumento di generazione del linguaggio interpreta tali valori e genera il linguaggio di conseguenza. Questo permette di personalizzare alcuni aspetti della semantica della NSF e permette di definire la semantica non standard al momento dell'assegnazione delle capacità di sicurezza alla NSF.

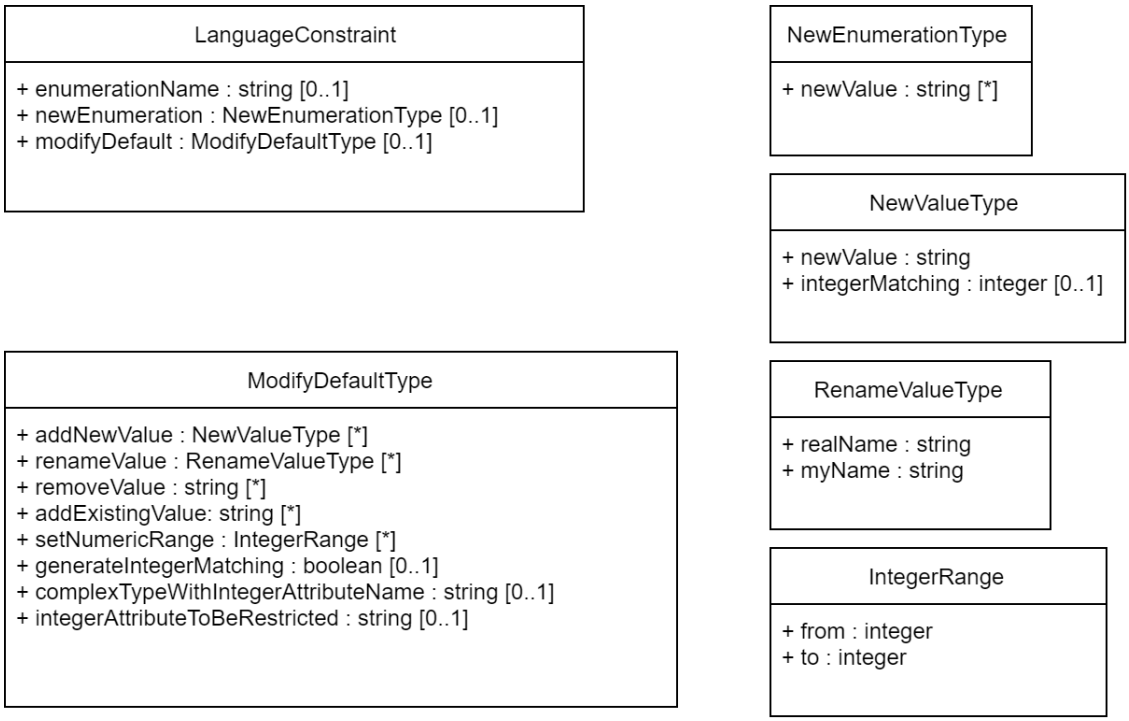

Figura 5.6. Language.

Nello specifico, l'obiettivo della classe "LanguageConstraint" è di permettere la personalizzazione dei valori assegnabili a determinati parametri, come possono essere parametri interi o valori appartenenti ad enumerazioni.

La classe proposta "LanguageConstraint" è composta dalla seguente struttura:

- enumerationName: eventuale nome dell'enumerazione da modificare.
- newEnumeration: eventuale parametro che permette la generazione di una enumerazione propria; è composto da:
  - newValue: valore che deve essere assegnato alla nuova enumerazione, ha cardinalità 0..n per poter generare una lista di nuovi valori;
- modifyDefault: parametro che permette di modificare i valori di default di un attributo, permette anche la restrizione di valori interi; è composto da:
  - addNewValue: parametro che permette di inserire un nuovo valore all'enumerazione di default. Questo elemento è composto ulteriormente da due parametri, di cui uno opzionale, per permettere, eventualmente, il matching tra valore stringa e valore numerico;
  - renameValue: parametro che permette di rinominare un valore esistente nell'enumerazione di default, questo elemento è composto ulteriormente da due parametri per permettere la nuova nomenclatura rispetto alla nomenclatura predefinita;
  - removeValue: parametro che permette di rimuovere valori esistenti nell'enumerazione predefinita e di considerare validi tutti gli altri. Questo parametro non può essere utilizzato assieme ad "addExistingValue";
  - addExistingValue: specificando questo elemento si dichiara di volere nella propria enumerazione il valore passato che è già presente nell'enumerazione predefinita. Questo parametro non può essere utilizzato assieme ad "removeValue";
  - setNumericRange: parametro che permette di indicare il range di valori interi che possono essere accettati nell'istanziazione della capacità quando si definisce una regola di una policy;

– generateIntegerMatching: parametro che, se vero, genera la corrispondenza tra valore stringa dell'enumerazione e valore intero del parametro corrispondente;

– complexTypeWithIntegerAttributeName: parametro che permette di indicare il nome del tipo complesso dichiarato nel CapDM al quale sarà imposta la modifica;

– integerAttrobuteToBeRestricted: parametro che permette di indicare la eventuale lista di parametri appartenenti a "complexTypeWithIntegerAttributeName" sui quali applicare la restrizione.

5.4 Classe Adapter

Per affrontare il problema della traduzione delle policy dal linguaggio generico della NSF nel linguaggio di basso livello della stessa, la tesi propone uno strumento che interpreta una sintassi espressa dalla classe "Adapter" al momento dell'assegnazione delle capacità di sicurezza alla NSF.

Infatti, lo strumento per la traduzione proposto conosce il linguaggio astratto della NSF, l'istanza della policy da tradurre e le regole di traduzione nella classe "Adapter"; una volta elaborate genera la policy tradotta nel linguaggio specifico dell'istanza della NSF.

La classe "Adapter" viene proposta come sottoclasse della classe di associazione definita da I2NSF "HasSecurityCapabilityDetail"; questo permette di rendere esplicita la capacità di sicurezza a cui appartiene e la NSF di contesto. La struttura che governa la traduzione nel linguaggio specifico della NSF proposta dalla tesi è rappresentata in Figura 5.7. A seconda dei valori specificati nella classe "Adapter" durante l'assegnazione delle capacità di sicurezza, lo strumento di generazione del linguaggio interpreta tali valori e traduce la policy di conseguenza. Alcuni parametri della classe "Adapter" possono influenzare la struttura del formato di traduzione di ogni capacità di sicurezza, questo può condizionare il metodo di utilizzo del file prodotto dalla traduzione.

Nello specifico, l'obiettivo della classe "Adapter" è di permettere la personalizzazione della semantica e della sintassi riguardante il linguaggio di basso livello della NSF. Questa classe è basata sulla definizione del CapDM e permette di personalizzare le caratteristiche utilizzate per la determinata capacità di sicurezza alla quale è stata assegnata, nel contesto della NSF. Situazioni comuni possono essere la definizione dei nomi dei comandi o dei separatori tra attributi.

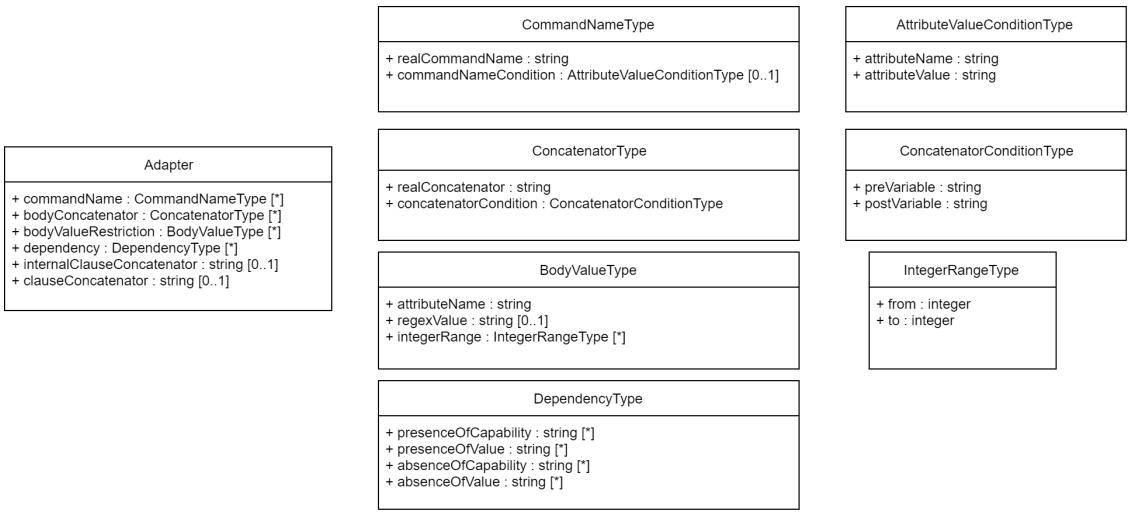

Figura 5.7. Adapter.

La classe proposta "Adapter" è composta dalla seguente struttura, i nomi degli attributi a cui di seguito si farà riferimento sono quelli definiti nel CapDM:

- commandName: parametro che permette di specificare il nome del comando nel linguaggio di basso livello della NSF. Dato che possono esserci più comandi relativi ad una capacità di sicurezza questo parametro è composto da:
  - realCommandName: parametro che contiene il nome effettivo da utilizzare quando viene rispettata l'eventuale "commandNameCondition";
  - commandNameCondition: parametro che permette di formare condizioni basate sul valore di attributi specificati. Questo parametro è composto da:
    - attributeName: nome dell'attributo da considerare per valutare se utilizzare il "realCommandName";
    - attributeValue: valore dell'attributo che deve essere confrontato per valutare se utilizzare il "realCommandName";
- bodyConcatenator: parametro che permette di specificare il separatore dei valori che possono essere attributi al relativo comando. Dato che possono esserci più separatori relativi agli attributi esprimibili questo parametro é composto da:
  - realConcatenator: valore del separatore da utilizzare nel caso in cui la "concatenatorCondition" viene verificata;
  - concatenatorCondition: parametro che permette di formare condizioni basate sul valore di attributi specificati. Questo parametro è composto da:
    - preVariable: nome dell'attributo da considerare il cui valore si troverà prima del concatenatore relativo;
    - postVariable: nome dell'attributo da considerare il cui valore si troverà dopo il concatenatore relativo;
- bodyValueRestriction: parametro che permette di restringere il campo di traduzione di parametri. Dato che possono esserci più tipi di restrizioni questo parametro é composto da:
  - attributeName: nome dell'attributo al quale si vuole dare delle restrizioni di traduzione;
  - regexValue: parametro che permette la dichiarazione di una regex per restringere il campo dei valori traducibili;
  - integerRange: parametro che permette la restrizione dei valori interi che si possono tradurre. Questo parametro è composto da:
    - from: valore di inizio del range;
    - to: valore di fine del range;
- dependency: parametro che permette di esprimere delle dipendenze della capacità di sicurezza a cui è assegnato rispetto ad altre capacità di sicurezza. Dato che possono esserci più tipologie di dipendenze questo parametro é composto da:
  - presenceOfCapability: impone la presenza di una determinata capacità di sicurezza;
  - presenceOfValue: impone la presenza di una determinato valore;
  - absenceOfCapability: impone l'assenza di una determinata capacità di sicurezza;
  - absenceOfValue: impone l'assenza di una determinato valore;
- internalClauseConcatenator: parametro che permette la definizione di un valore che separa il nome del comando ed il valore ad esso attribuito;
- clauseConcatenator: parametro che permette la definizione di un valore che separa il costrutto tradotto della capacità di sicurezza a cui viene assegnato dalla successiva.

Capitolo 6

Analisi e validazione su un caso concreto: iptables

In questo capitolo verrà esposto il workflow proposto dalla tesi per la gestione di NSF in ambito di sicurezza delle reti. Successivamente si analizza un caso concreto, iptables, mostrando concretamente l'applicazione del workflow.

6.1 Workflow

Il workflow di riferimento, mostrato in Figura 6.1, si basa sul modello informativo ed il modello dei dati descritti nel capitolo 5.

Avendo definito il modello dei dati delle capacità di sicurezza (CapDM), contenente sufficienti capacità di sicurezza, si può istanziare una NSF. Le fasi descritte dal workflow in Figura 6.1 si sviluppano come descritto di seguito:

1. *istanza di una NSF*: decisione di voler generare una nuova NSF e studio delle capacità di sicurezza necessarie o appartenenti a tale NSF;
2. *assegnazione di capacità di sicurezza*: in questa fase si assegnano le capacità di sicurezza alla NSF, utilizzando i nomi definiti nel CapDM. Dato che la relazione tra NSF e Security-Capabilty del CapIM è una aggregazione, per l'assegnazione è sufficiente generare un elenco delle capacità di sicurezza necessarie alla NSF;
3. *definizione di Adapter e Language per ogni capacità di sicurezza*: in questa fase si deve essere a conoscenza delle caratteristiche di basso livello della NSF. Infatti, nella classe "Language" andranno descritti i criteri di definizione del linguaggio propri della NSF, mentre nella classe "Adapter" saranno descritti i criteri necessari per la traduzione nel linguaggio di basso livello della NSF, entrambi relativamente alla specifica capacità di sicurezza a cui sono assegnati;
4. *generazione del linguaggio della NSF*: in questa fase si fa uso dello strumento proposto nell'ambito della tesi che permette la generazione del linguaggio della specifica NSF;
5. *creazione della policy nel linguaggio della NSF*: in questa fase, essendo a conoscenza del linguaggio generico della NSF, si può procedere alla definizione delle policy utilizzando tale linguaggio;
6. *traduzione della policy nel linguaggio di basso livello della NSF*: in questa fase si fa uso dello strumento proposto nell'ambito della tesi che permette la traduzione della policy espressa nel linguaggio astratto della NSF nella policy espressa nel linguaggio di basso livello e nel formato necessario alla NSF;
7. *attuazione della policy di sicurezza*: in questa fase si utilizza il file di output della fase precedente per poter usufruire direttamente della policy tradotta nel linguaggio di basso livello della NSF.

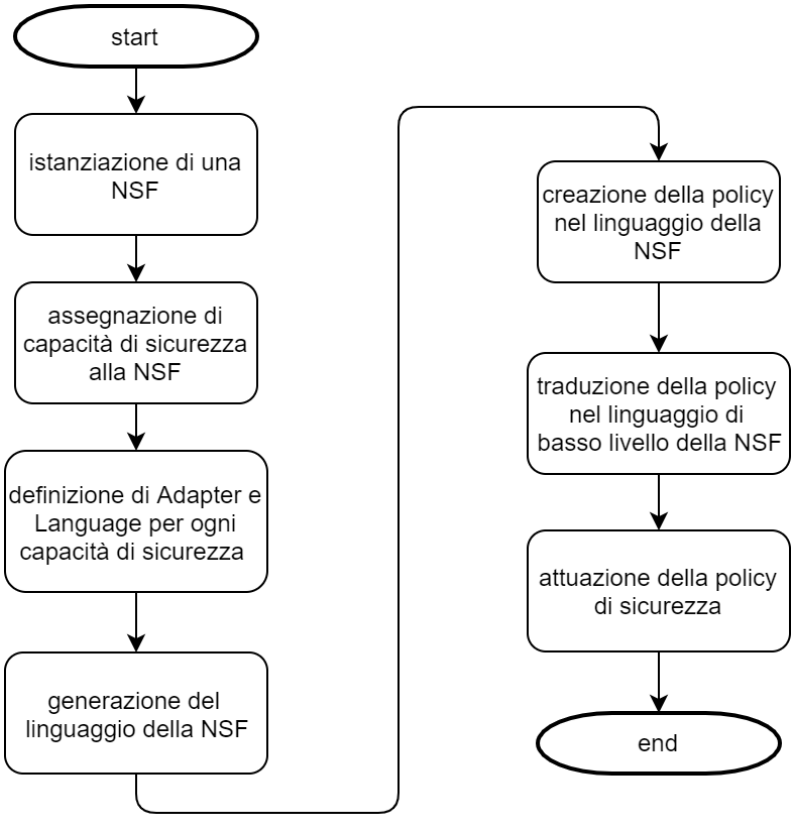

Figura 6.1. Workflow.

6.2 Analisi firewall iptables

In questa sezione viene analizzato il workflow precedentemente descritto con un caso concreto di funzionalità di sicurezza di rete NSF: firewall con funzionalità di packet filtering.

Questo esempio è basato sul componente interno del Kernel di Linux chiamato Iptables/Netfilter.

Col termine Netfilter si riferisce ad un framework all'interno del kernel Linux che può essere utilizzato per agganciare le funzioni nello stack di rete. Netfilter è un componente back-end che funziona solo nel kernel ed è responsabile dell'elaborazione del traffico. Netfilter espone determinati hook, ossia delle interfacce su cui è possibile collegare funzioni per elaborare il traffico nel kernel.

Iptables è un firewall che utilizza Netfilter per agganciare funzioni progettate per eseguire operazioni su pacchetti, come il filtraggio, nello stack di rete. Si può pensare a Netfilter come ad un framework in cui iptables crea funzionalità di firewall. Iptables viene usato per impostare, mantenere ed ispezionare le tabelle delle regole di filtro dei pacchetti IP nel kernel Linux. Nell'ambiente iptables sono preimpostate cinque tabelle. Ogni tabella contiene un numero di catene integrate e può anche contenere catene definite dall'utente. Ogni catena è un elenco di regole che possono corrispondere a un insieme di pacchetti. Ogni regola specifica cosa fare con un pacchetto che corrisponde ai requisiti definiti dalla regola stessa. Iptables è l'applicazione front-end nello spazio utente utilizzabile da riga di comando, attraverso la quale si viene configurato il set di regole di filtro pacchetti.

Iptables è un sistema firewall molto conosciuto e normalmente preinstallato sui sistemi operativi Unix. Esso è molto conosciuto e sufficientemente complesso da poter analizzare un ampio spettro di capacità di sicurezza definite nel CapDM.

In questo paragrafo vengono riportati nel dettaglio gli esempi più significativi dello sviluppo del caso concreto analizzato senza riportare completamente tutti i passaggi, i dettagli ed i file di configurazione realmente necessari per l'implementazione effettuata, per non appesantire la lettura dell'elaborato.

Ogni sezione seguente corrisponde ad una fase del workflow in Figura 6.1. Il presupposto su cui si basano le seguenti fasi è di aver generato il file contenente il CapDM e le classi "Adapter" e "LanguageConstraint" in formato XMLSchema, ad esempio, utilizzando il tool apposito con il comando:

```
java -jar XMItoXSDconverter.jar CapabilityDataModel.xmi
```

Questo comando genera il file "converted.xsd" che contiene tutte le capacità di sicurezza mappate nel CapDM e le classi "Adapter" e "LanguageConstraint".

6.2.1 Studio delle capacità

Nella prima fase sono state studiate nel dettaglio le funzionalità offerte da iptables. Dato che iptables possiede alcune caratteristiche specifiche, è stata aggiunta nel modello una sezione dedicata ad iptables che rappresentasse in modo generico le particolari caratteristiche da esso possedute. Alcune capacità di sicurezza specifiche mappate nel CapDM dopo lo studio di iptables sono riportate in Figura 6.2.

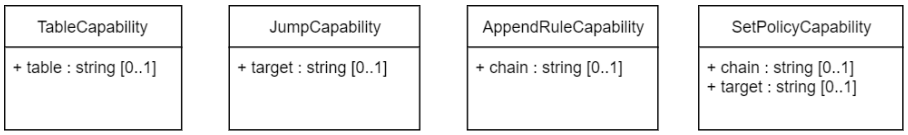

Figura 6.2. Classe specifiche di Iptables.

Queste nuove capacità di sicurezza sono state categorizzate sotto alla sezione "azione" della 6-tupla precedentemente descritta. Inoltre. sono state rese generiche e qualsiasi NSF può utilizzarle se le vengono assegnate.

6.2.2 Assegnazione delle capacità

Dopo aver studiato le capacità di sicurezza offerte dal firewall iptables, si può procedere alla stesura della lista di esse in un formato generico compatibile con la traduzione del CapDM. Dato che nell'architettura è stato definito l'utilizzo del linguaggio XML come linguaggio generico di supporto, in Figura 6.3 viene mostrata l'assegnazione di alcune capacità di sicurezza interne al file XML di configurazione di iptables.

Le capacità di sicurezza scelte per mostrare l'esempio sono:

- IpDestinationAddressCapability: questa capacità di sicurezza generica di tipo condizione permette di identificare l'indirizzo ip di destinazione di un pacchetto;
- IpProtocolTypeCapability: questa capacità di sicurezza generica di tipo condizione permette di identificare il tipo di protocollo che gestisce la trasmissione di un pacchetto;
- SourcePortCapability: questa capacità di sicurezza generica di tipo condizione permette di identificare la porta sorgente dalla quale proviene un pacchetto;
- JumpCapability: questa capacità di sicurezza specifica di iptables, ma resa generica, di tipo azione, permette di dichiarare l'azione da svolgere quando si avverano le condizioni;
- AppendRuleCapability: questa capacità di sicurezza specifica di iptables, ma resa generica, permette di dichiarare l'azione da svolgere alla regola stessa al momento della dichiarazione;

```
<securityCapability xsi:type="p:IpDestinationAddressCapability"/>
<securityCapability xsi:type="p:IpProtocolTypeCapability"/>
<securityCapability xsi:type="p:SourcePortCapability"/>
<securityCapability xsi:type="p:JumpCapability"/>
<securityCapability xsi:type="p:AppendRuleCapability"/>
<securityCapability xsi:type="p:TableCapability"/>
```

Figura 6.3. Assegnazione capacità di sicurezza di esempio.

- TableCapability: questa capacità di sicurezza specifica di iptables, ma resa generica, permette di identificare la tabella che deve gestire la regola a cui viene assegnata.

Successivamente il file di configurazione delle capacità di sicurezza da assegnare ad una NSF può essere validato mediante il tool specifico di validazione. Dato che la trascrizione del file di validazione per la verifica delle capacità di sicurezza di iptables è stato fatto a mano, la validazione evita errori di trascrizione o errori dovuti a capacità di sicurezza inesistenti nel modello. In questa fase le capacità di sicurezza sono utilizzate in modo astratto e solo in funzione del nome, infatti non è necessario definire o specificare gli attributi di ogni capacità di sicurezza dato che sono definiti nel CapDM.

6.2.3 Definizione Language e Adapter per le capacità di sicurezza assegnate

Dopo aver studiato le capacità di sicurezza offerte dal firewall iptables è stata appresa la sua sintassi e la sua semantica, questo ha permesso di istanziare le classi Language e Adapter necessarie ai tool per la generazione del linguaggio e la traduzione delle policy.

Le classi Language ed Adapter sono rappresentate da campi interni ad ogni capacità di sicurezza, questo permette di essere specifici per quella determinata capacità. Dato che le capacità sono raggruppate ed assegnate ad una particolare NSF, queste classi permettono di specificare dettagli relativi alla capacità di sicurezza nel sistema della NSF specifica. La classe Language permette di personalizzare la generazione del linguaggio della NSF relativamente alla singola capacità di sicurezza. La classe Adapter, invece, permette di specificare i dettagli di traduzione e di dipendenze della singola capacità di sicurezza nell'ambito della NSF. La definizione dei dettagli della classe Adapter influenza la semantica e la struttura effettiva della traduzione delle capacità di sicurezza. Quindi per la definizione delle caratteristiche di un Adapter bisogna tenere conto anche dello scopo e dell'uso che si farà della traduzione della policy.

I seguenti esempi sono riferiti ad alcune delle capacità di sicurezza precedentemente elencate.

Nel contesto di iptables il comando necessario per aggiungere una nuova regola viene espresso come "-A *catena specifiche-della-regola*". Per permettere questa traduzione la classe Adapter verrà espressa come mostrato in Figura 6.4. L'elemento principale di tale Figura è la definizione del campo "realCommandName", dove viene espressa la semantica specifica della NSF per dichiarare la capacità di sicurezza alla quale la classe Adapter è assegnata. Oltre alla dichiarazione del comando è stata espressa la dipendenza rispetto alla capacità di sicurezza "TableCapability"; in fase di traduzione questa restrizione rende obbligatoria la presenza di tale capacità di sicurezza quando viene utilizzata la "AppendRuleCapability". I nomi utilizzati per esprimere le dipendenze sono quelli dichiarati dal CapDM.

Per analizzare un caso di Adapter più complesso si può fare riferimento alla capacità di sicurezza "IpDestinationAddressCapability".

Nel caso di iptables tale comando permette di esprimere regole verso gli indirizzi IP di destinazione di un pacchetto, questa condizione viene espressa come "[!] -d *address[/mask]*". Per poter mappare questa tipologia di traduzione viene realizzato l'Adapter in Figura 6.5. Da questa definizione si possono notare la presenza di due "realCommandName", che rappresentano le

```
<securityCapability xsi:type="p:AppendRuleCapability">
  <adapter>
    <commandName>
      <realCommandName>-A</realCommandName>
    </commandName>
    <dependency>
      <presenceOfCapability>TableCapability</presenceOfCapability>
    </dependency>
  </adapter>
</securityCapability>
```

Figura 6.4. AppendRuleCapability.

traduzioni della capacità di sicurezza negata o non negata. In questo modo il tool è in grado di utilizzare entrambe le traduzioni ed utilizzerà quella giusta in funzione del valore del parametro "operation". Inoltre una ulteriore differenza rispetto alla Figura 6.4, riguarda la definizione del separatore, infatti il comando iptables permette l'utilizzo opzionale della variabile "mask", la quale è separata dalla variabile "address" da un simbolo specifico. In questo modo si possono esprimere i separatori tra attributi. Per permettere l'identificazione di quale siano gli attributi da separare sono stati specificati dei costrutti appositi, nei quali vengono inseriti i valori dei parametri a cui fare riferimento. Tali parametri vengono espressi con la nomenclatura definita nel CapDM.

```
<securityCapability xsi:type="p:IpDestinationAddressCapability">
  <adapter>
    <commandName>
      <realCommandName>-d</realCommandName>
    </commandName>
    <commandName>
      <realCommandName>! -d</realCommandName>
      <commandNameCondition>
        <attributeName>operation</attributeName>
        <attributeValue>NOT_EQUAL_TO</attributeValue>
      </commandNameCondition>
    </commandName>
    <bodyConcatenator>
      <realConcatenator>/</realConcatenator>
      <concatenatorCondition>
        <preVariable>address</preVariable>
        <postVariable>mask</postVariable>
      </concatenatorCondition>
    </bodyConcatenator>
    <dependency>
      <presenceOfCapability>AppendRuleCapability</presenceOfCapability>
    </dependency>
  </adapter>
</securityCapability>
```

Figura 6.5. IpDestinationAddressCapability.

Per analizzare un esempio della classe Language viene riportata la capacità di sicurezza di "IpProtocolTypeCapability". In questo caso iptables definisce un insieme di protocolli che sono esprimibili utilizzando il nome stesso del protocollo, in versione maiuscola o minuscola. Inoltre, il protocollo, può essere espresso come valore numerico ma i valori numerici possibili sono limitati. Per esprimere queste caratteristiche si definisce la classe Language come in Figura 6.6.

Nel CapDM sono stati espressi dei valori predefiniti per le enumerazioni, questo esempio rende nota la possibilità di poter manipolare tali valori. Per lo scopo dell'esempio si fa riferimento ai valori di predefiniti dell'enumerazione "ProtocolTypeEnumeration", Figura 5.5.

Analizzando la Figura 6.6 si rendono note le seguenti possibilità di personalizzazione date dalla classe Language:

- dichiarare a quale enumerazione fare riferimento;
- dichiarare nuovi valori per l'enumerazione definendo il numero che corrisponde alla stringa. Questo tipo di definizione permette di generare un matching, tra valore intero e valore stringa dell'enumerazione. La dichiarazione del numero intero è opzionale, nel caso del protocollo però avere il matching può risultare utile;
- rinominare determinati valori permettendo di conoscere la corrispondenza del valore predefinito con quello nuovo da utilizzare. La dichiarazione del valore predefinito permette, ad esempio, di mantenere traccia del riferimento numerico del nuovo valore;
- assegnare dei valori esistenti nell'enumerazione preimpostata all'enumerazione specifica della NSF. Questo parametro può essere sostituito dall'elenco dei valori che si vogliono escludere, per praticità nell'esempio è stato utilizzato il metodo per rendere nota la conferma dei valori preimpostati;
- restringere gli insiemi numerici di valori interi, definendo l'insieme preciso dei numeri che si potranno assegnare nel linguaggio della NSF. Assieme a questa specifica devono essere esplicitati quali parametri dovranno utilizzare questa restrizione.

Il file completo viene convalidato utilizzando il tool apposito nei confronti del file XSD contenente il CapDM. Tale file verrà utilizzato per la generazione del linguaggio della NSF.

6.2.4 Generazione linguaggio

Nella fase di generazione del linguaggio viene utilizzato lo strumento proposto nel progetto di tesi creato appositamente per lo scopo. Per poter utilizzare lo strumento di generazione del linguaggio sono necessari il file generato nel passaggio precedente ed il file che contiene la rappresentazione in formato XMLSchema del CapDM. Ad esempio, nel caso considerato, si può utilizzare il comando seguente:

```
java -jar generateLanguage.jar converted.xsd iptables.xml
```

dove i parametri passati corrispondono a:

- converted.xsd - rappresenta il percorso al file che contiene l'elenco completo delle capacità di sicurezza mappate nel CapDM e che contiene la definizione delle classi "Adapter" e "LanguageConstraint";
- iptables.xml - rappresenta il percorso al file creato nella Sezione 6.2.2 e Sezione 6.2.3.

Tale comando genera il linguaggio generico della NSF con nome standard "language.xsd".

Il linguaggio generato dal tool relativo alla capacità di sicurezza "IpProtocolTypeCapability" viene spiegato nelle seguenti figure:

- la Figura 6.7 mostra la struttura della capacità di sicurezza "IpProtocolTypeCapability" rappresentata con funzionalità di condizione nel linguaggio della NSF, in questo caso iptables. La rappresentazione scelta permette di istanziare la classe di tipo "IpProtocolTypeCapability" per porre condizioni sul tipo di protocollo desiderato utilizzando l'apposito elemento chiamato "protocolType". I dettagli degli attributi "name", della porzione di codice in Figura, sono direttamente dipendenti dalla rappresentazione della capacità di sicurezza relativa nel CapDM;

```
<securityCapability xsi:type="p:IpProtocolTypeCapability">
          .
          .
<language>
  <enumerationName>ProtocolTypeEnumeration</enumerationName>
  <modifyDefault>
    <addNewValue>
      <newValue>hopopt</newValue>
      <integerMatching>0</integerMatching>
    </addNewValue>
    <addNewValue>
      <newValue>icmp</newValue>
      <integerMatching>1</integerMatching>
    </addNewValue>
    <addNewValue>
      <newValue>igmp</newValue>
      <integerMatching>2</integerMatching>
    </addNewValue>
          .
          .
          .
    <renameValue>
      <realName>IP-in-IP</realName>
      <myName>IP-ENCAP</myName>
    </renameValue>
    <renameValue>
      <realName>ESP</realName>
      <myName>IPSEC-ESP</myName>
    </renameValue>
    <renameValue>
      <realName>AH</realName>
      <myName>IPSEC-AH</myName>
    </renameValue>
          .
          .
          .
    <addExistingValue>HOPOPT</addExistingValue>
    <addExistingValue>ICMP</addExistingValue>
    <addExistingValue>IGMP</addExistingValue>
          .
          .
    <setNumericRange>
      <from>0</from>
      <to>255</to>
    </setNumericRange>
    <complexType>ProtocolType</complexType>
    <integerAttribute>protocolTypeNumber</integerAttribute>
  </modifyDefault>
</language>
          .
          .
</securityCapability xsi:type="p:IpProtocolTypeCapability">
```

Figura 6.6. Assegnazione classe Language alla capacità di sicurezza di esempio.

```
<xs:complexType name="IpProtocolTypeCapability">
  <xs:complexContent>
    <xs:extension base="Ipv4Capability">
      <xs:sequence>
        <xs:element maxOccurs="1" minOccurs="0" name="protocolType"
            type="ProtocolType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Figura 6.7. Linguaggio per indicare un protocollo.

- la Figura 6.8 mostra la struttura del tipo dell'elemento di nome "protocolType" relativo alla Figura 6.7. La porzione di codice in Figura 6.8 rappresenta gli elementi da cui è formato il tipo generico "ProtocolType"; gli elementi che lo compongono sono direttamente dipendenti dalla rappresentazione del tipo generico nel CapDM. Invece i tipi relativi agli elementi presenti nell'esempio sono dipendenti dalla classe Language definita per tale capacità di sicurezza al momento dell'assegnazione della stessa alla NSF;

```
<xs:complexType name="ProtocolType">
  <xs:sequence>
    <xs:element maxOccurs="1" minOccurs="0" name="protocolTypeName"
        type="ProtocolTypeEnumeration"/>
    <xs:element maxOccurs="1" minOccurs="0" name="protocolTypeNumber"
        type="protocolTypeNumberRestriction"/>
  </xs:sequence>
</xs:complexType>
```

Figura 6.8. Linguaggio del tipo di protocollo.

- la Figura 6.9 mostra la struttura del tipo "ProtocolTypeEnumeration". Questa enumerazione può avere i valori preimpostati nella sezione dei tipi generici del CapDM, oppure può contenere valori personalizzati definiti come in Figura 6.6. In questo esempio sono stati riportati gli stessi attributi che sono stati specificati in Figura 6.6. In questo modo vengono resi noti al linguaggio della NSF tutti i possibili valori che potrà assumere ogni attributo di tipo "ProtocolTypeEnumeration" nel momento di una eventuale istanziazione;

- la Figura 6.10 mostra la restrizione di ogni attributo di tipo "protocolTypeNumberRestriction". Questo tipo di classe, dato che può non essere standard, viene generata in modo dinamico solo sulla base della definizione della classe Language in fase di assegnazione delle capacità di sicurezza. Infatti, nonostante l'esempio riporti esattamente il valore numerico minimo e massimo realmente assegnabile ad un protocollo, una NSF generica può decidere di utilizzare solo uno o più determinati insiemi di valori numerici.

Lo strumento di generazione del linguaggio fornisce in output un file in formato XMLSchema che rappresenta il linguaggio della NSF. Il linguaggio è considerato generico perchè viene generato principalmente dalla modellazione delle capacità di sicurezza nel CapDM. Infatti viene generato un linguaggio sufficientemente generico da essere compatibile con i linguaggi di altre NSF nelle parti di linguaggio riguardanti le capacità di sicurezza assegnate in comune. Ma è considerato il linguaggio specifico della NSF perchè è caratterizzato dalla scelta delle capacità di sicurezza che conosce e da eventuali personalizzazioni ulteriori.

```
<xs:simpleType name="ProtocolTypeEnumeration">
  <xs:restriction base="xs:string">
    .
    .
  <xs:enumeration value="HOPOPT"/>
  <xs:enumeration value="ICMP"/>
  <xs:enumeration value="IGMP"/>
  <xs:enumeration value="IPSEC-ESP"/>
  <xs:enumeration value="IPSEC-AH"/>
  <xs:enumeration value="hopopt"/>
  <xs:enumeration value="icmp"/>
  <xs:enumeration value="igmp"/>
    .
    .
  </xs:restriction>
</xs:simpleType>
```

Figura 6.9. Valori dell'enumerazione del protocollo.

```
<xs:simpleType name="protocolTypeNumberRestriction">
  <xs:union>
    <xs:simpleType>
      <xs:restriction base="xs:integer">
        <xs:minInclusive value="0"/>
        <xs:maxInclusive value="255"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:union>
</xs:simpleType>
```

Figura 6.10. Valori numerici possibili del protocollo.

Inoltre lo strumento di generazione del linguaggio genera eventualmente file di testo contenenti metadati riguardo le associazioni di valori numerici rispetto a valori alfabetici delle enumerazioni che necessitano tale abbinamento.

6.2.5 Creazione policy

Nella fase di creazione delle policy l'informazione principale da tenere presente è il linguaggio della NSF alla quale si vuole assegnare la policy. La Figura 6.11 mostra una regola valida di una eventuale policy nel linguaggio della NSF di esempio. Considerando il caso di iptables la regola proposta si suddivide nelle seguenti parti:

- TableCapability: utilizzando questa capacità di sicurezza si può definire quale tabella, specifica di iptables, dovrà gestire la regola, la tabella di riferimento viene definita nell'attributo "table". In questo caso ci si riferisce alla tabella "filter";
- AppendRuleCapability: utilizzando questa capacità di sicurezza si può definire quale catena dovrà gestire la regola, la catena di riferimento viene definita nell'attributo "chain". In questo caso ci si riferisce alla catena "FORWARD";
- IpDestinationAddressCapability: utilizzando questa capacità di sicurezza si può definire su quale indirizzo IP di destinazione si vuole fornire una condizione, l'indirizzo a cui si fa

riferimento si può definire nell'attributo "address" ed eventualmente la sua maschera nell'attributo "mask". In questo caso è stato definito l'indirizzo ip "192.168.1.1" con maschera "255.255.255.0";

- IpProtocolTypeCapability: utilizzando questa capacità di sicurezza si può definire su quale protocollo si vuole fornire una condizione, il protocollo a cui si fa riferimento viene definito nell'attributo "protocolTypeName" perchè viene espresso con un nome valido rispetto al linguaggio. In questo caso si crea una condizione sul protocollo "TCP";
- IpDestinationAddressCapability: utilizzando questa capacità di sicurezza si può definire su quale porta si vuole fornire una condizione. In questo caso viene specificata una condizione su un insieme di porte, limitate inferiormente dal valore espresso nella variabile "startValue", nell'esempio, la porta 80, e limitate superiormente dal valore espresso nella variabile "endValue", nell'esempio, la porta 90;
- SourcePortCapability: utilizzando questa capacità di sicurezza si può definire l'azione che si vuole effettuare identificato un pacchetto che rispetta le precedenti condizioni. In questo caso viene specificata l'azione "ACCEPT" che permette il passaggio di tale pacchetto.

```
<securityCapability xsi:type="TableCapability">
  <table>filter</table>
</securityCapability>
<securityCapability xsi:type="AppendRuleCapability">
  <chain>FORWARD</chain>
</securityCapability>
<securityCapability xsi:type="IpDestinationAddressCapability">
  <ipAddress>
    <address>192.168.1.1</address>
    <mask>255.255.255.0</mask>
  </ipAddress>
</securityCapability>
<securityCapability xsi:type="IpProtocolTypeCapability">
  <protocolType>
    <protocolTypeName>TCP</protocolTypeName>
  </protocolType>
</securityCapability>
<securityCapability xsi:type="SourcePortCapability">
  <port>
    <startValue>80</startValue>
    <endValue>90</endValue>
  </port>
</securityCapability>
<securityCapability xsi:type="JumpCapability">
  <target>ACCEPT</target>
</securityCapability>
```

Figura 6.11. Istanza di regola nel linguaggio della NSF.

In questa fase le capacità di sicurezza hanno il ruolo della loro categoria di riferimento, ad esempio IpProtocolTypeCapability rappresenta una condizione sul protocollo, JumpCapability rappresenta una azione da effettuare quando sono verificate le condizioni.

6.2.6 Traduzione delle policy

La fase di traduzione della policy è strettamente dipendente da come si vuole utilizzare il file di output dello strumento di traduzione della policy. Oltre alla classe Adapter lo strumento di

traduzione è stato pensato per permettere tipi di output di formati diversi, i dettagli dell'utilizzo dello strumento verranno analizzati più approfonditamente nella Sezione A.3.

Dato che possono esserci differenti formati disponibili mediante i quali una NSF può ricevere le policy o le regole da attuare, è necessario lo studio dei vari formati anche allo scopo di decidere quale convenga di più utilizzare. In funzione di questo possono essere definiti diversi tipi di classi Adapter e si può utilizzare il tool di traduzione, in modi specifici, per poter generare strutture di traduzioni differenti.

Nell'esempio di iptables si possono identificare due tipi principali di formati per l'output in funzione di come, successivamente, si vuole attuare la policy/regola nei confronti di iptables. I successivi formati di output sono stati ottenuti modificando la classe Adapter della capacità di sicurezza "TableCapability" ed utilizzando determinati parametri da passare allo strumento di traduzione.

- il primo formato identificato, Figura 6.12, permette di generare un file che contiene per ogni riga una singola regola autonoma. Questo file si può utilizzare per verificare la possibilità di inserire ogni regola direttamente da linea di comando.

```
iptables -t filter -A FORWARD -d 192.168.1.1/255.255.255.0 -p TCP --sport
    80:90 -j ACCEPT
```

Figura 6.12. Istanza di regola nel linguaggio della NSF per configurazioni da linea di comando.

  In questo caso la capacità di sicurezza relativa a "TableCapability" presenta come valore di traduzione del comando il termine "-t" e non vengono specificati ulteriori parametri; inoltre al tool di traduzione è stata passata la parola chiave "iptables " come uno dei parametri specifici opzionali resi disponibili dal tool stesso. Questo parametro permette di generare un output contenente tale parole chiave come elemento iniziale per ogni regola tradotta.

  Per ottenere questo tipo di formato utilizzando le informazioni dell'esempio è stato utilizzato il tool di traduzione con il comando seguente:

  ```
  java -jar translatePolicyRule.jar language.xsd iptables.xml
      iptablesRuleInstance.xml "+siptables "
  ```

  dove i parametri passati corrispondono a:

  - language.xsd - rappresenta il percorso al file che contiene il linguaggio della NSF;
  - iptables.xml - rappresenta il percorso al file creato nella Sezione 6.2.2 e nella Sezione 6.2.3;
  - iptablesRuleInstance.xml - rappresenta il percorso al file creato nella fase spiegata nella Sezione 6.2.5;
  - "+siptables␣" - permette di inserire all'inizio di ogni regola la parola chiave "iptables␣", le virgolette sono state messe apposta per inserire anche il carattere spazio.

  Tale comando genera un file di testo con nome standard "policy.txt" con la regola tradotta come in Figura 6.12.

- il secondo formato identificato, Figura 6.13, permette di generare un file contenente le regole in modo che iptables possa aggiungere alle sue policy con un comando specifico le regole contenute nel file.

  In questo caso la calsse Adapter della capacità di sicurezza relativa a "TableCapability" presenta come valore di traduzione del comando il termine "*" e viene specificata la congiunzione con la capacità di sicurezza successiva con il carattere relativo per lo scopo di andare a capo; inoltre al tool di traduzione è stata passata la parola chiave "COMMIT" come uno dei parametri specifici opzionali resi disponibili dal tool stesso. Questo parametro permette al tool di concludere ogni regola che traduce con il valore passato.

```
*filter
-A FORWARD -d 192.168.1.1/255.255.255.0 -p TCP --sport 80:90 -j ACCEPT
COMMIT
```

Figura 6.13. Istanza di regola nel linguaggio della NSF per configurazioni da file.

Per ottenere questo tipo di formato utilizzando le informazioni dell'esempio è stato utilizzato il tool di traduzione con il comando seguente:

```
java -jar translatePolicyRule.jar language.xsd iptables.xml
    iptablesRuleInstance.xml +eCOMMIT +cre
```

dove i parametri passati corrispondono a:

- language.xsd - rappresenta il percorso al file che contiene il linguaggio della NSF;
- iptables.xml - rappresenta il percorso al file creato nella Sezione 6.2.2 e nella Sezione 6.2.3;
- iptablesRuleInstance.xml - rappresenta il percorso al file creato nella fase spiegata nella Sezione 6.2.5;
- +eCOMMIT - permette di inserire alla fine di ogni regola la parola chiave "COMMIT";
- +cre - permette di inserire un "a capo" prima dell'inserimento della parola chiave "COMMIT".

Tale comando genera un file di testo con nome standard "policy.txt" con la regola tradotta come in Figura 6.13.

La funzionalità delle regole in Figura 6.12 ed in Figura 6.13 è equivalente, le differenze principali corrispondono all'evidente struttura con la quale vengono espresse ed il metodo di utilizzo dei relativi file interfacciandosi con iptables stesso. Utilizzare il primo formato per la conversione di una policy non permette l'inserimento automatico della stessa nel linguaggio di basso livello della NSF. Invece utilizzare il secondo formato permette la gestione ed il caricamento delle policy in modo automatico. Infatti, il file così generato, può essere considerato come un file di configurazione per policy da assegnare ad iptables utilizzando il seguente comando in figura 6.14.

```
iptables-restore < nome-del-file
```

Figura 6.14. Comando per il caricamento automatico delle policy presenti in un file.

6.2.7 Attuazione concreta delle policy

Per poter verificare l'effettiva compatibilità con iptables è stata istanziata una macchina virtuale con sistema operativo Ubuntu. I sistemi operativi Linux disponibili gratuitamente come ubuntu si basano sulle idee che il software dovrebbe esser disponibile gratuitamente, gli strumenti software dovrebbero essere utilizzabili dalle persone nella loro lingua locale e che le persone dovrebbero avere la libertà di personalizzare ed alterare il loro software nel modo che ritengono opportuno. Le distribuzioni basate su Linux permettono l'istallazione agevole di software come iptables.

Utilizzando il sistema operativo ubuntu si ha come firewall preinstallato proprio iptables. Inizialmente il firewall risulta senza regole. Per poter interagire con il firewall iptables sono necessari i privilegi di amministratore, ad esempio ottenibili preponendo la parola specifica "sudo" al comando desiderato.

Con il comando 3 in Figura 6.16 si ottengono i privilegi di amministratore grazie alla parola specifica "sudo", ed utilizzando il comando "-L" si ottiene la lista delle regole impostate alla tabella "filter", sottintesa dato che non è stata indicata nessuna tabella.

Per aggiungere una regola in modo manuale si può utilizzare il formato realizzato nel file secondo il procedimento descritto per ottenere la Figura 6.12. Con questo formato si può copiare ogni riga contenuta dal file ed incollarla nella linea di comando per imporre tale regole al firewall iptables, come il comando 4 in Figura 6.16.

Utilizzando la sequenza di azioni appena descritta si ottiene come risultato l'aggiunta di una regola nel firewall iptables, come mostrato in Figura 6.15. Come si può notare nella catena "FORWARD" della tabella "filter" è stata inserita una regola, esattamente quella richiesta confermando il comando 4 rappresentato in Figura 6.16.

```
Chain FORWARD (policy DROP)
target     prot opt  source           destination
ACCEPT     tcp  --   anywhere         192.168.1.1/24     tcp spts:http:90
```

Figura 6.15. Stato finale iptables, tabella filter.

Utilizzare la sequenza di azioni appena descritta non è pratico pensando a sistemi o ambienti automatizzati. Per permettere un procedimento automatico si può procedere utilizzando un file di configurazione ottenuto mediante il processo descritto per ottenere la rappresentazione della regola come in Figura 6.13. Utilizzando questo formato di file, iptables mette a disposizione un comando che permette il caricamento di policy direttamente da un file di testo. Infatti utilizzando il comando in Figura 6.14 vengono caricate le regole contenute in tale file nelle regole applicate da iptables, eliminando quelle precedentemente presenti nel firewall iptables. Considerando di avere la regola definita come in Figura 6.13 contenuta nel file di nome "configurazione.txt" si può utilizzare il comando 1 in Figura 6.16 per caricare direttamente le regole presenti in tale file.

Dato che potrebbe essere utile mantenere le eventuali regole già impostate nel firewall iptables si può utilizzare il comando 2 in Figura 6.16 per permettere di inserire le regole contenute nel file prodotto mediante il tool. Questo comando però mantiene anche regole già presenti nel firewall iptables, quindi potrebbe generare inserimenti di regole duplicate.

```
//comando 1:
iptables-restore < configurazione.txt

//comando 2:
iptables-restore -n < configurazione.txt

//comando 3:
sudo iptables -L

//comando 4:
iptables -t filter -A FORWARD -d 192.168.1.1/255.255.255.0 -p TCP --sport
    80:90 -j ACCEPT
```

Figura 6.16. Aggiunta di una delle regole nel file.

Capitolo 7

Conclusioni

Lo studio affrontato ha esaminato il modello informativo definito dal gruppo di lavoro Interface To Network Security Functions e ne ha definito un modello di dati analizzando determinate capacità di sicurezza per i servizi di rete. Unitamente a ciò è stato ampliato il modello informativo di partenza per permettere la gestione di politiche di sicurezza per servizi di rete, rendendo tale modello compatibile ad un approccio policy-based management, ovvero un approccio di amministrazione della rete tramite gestione basata su politiche.

Lo studio effettuato ha realizzato un metodo per rappresentare, in modo generico, le funzionalità offerte dai servizi di sicurezza. Utilizzando questo metodo per la rappresentazione generica delle capacità dei dispositivi, è stato possibile svolgere diverse attività essenziali sia nell'ambito della ricerca che nel campo dell'amministrazione della sicurezza di rete. Ad esempio, si è reso possibile confrontare in maniera formale e precisa dispositivi di sicurezza diversi per verificare se hanno caratteristiche in comune per, ad esempio, poter applicare le stesse politiche di sicurezza. Nel corso dello studio si è evidenziato il problema che ogni elemento che applica funzionalità di sicurezza utilizza un proprio linguaggio, è sorta quindi la necessità di trovare un linguaggio astratto per controlli di sicurezza, che, ad esempio, permetta di esprimere politiche di sicurezza con una sintassi comune. Il criterio di definizione del linguaggio è basato sulle capacità di sicurezza assegnate alla specifica NSF, in questo modo si ottiene un linguaggio astratto e generico ma specifico per ogni NSF. Tale linguaggio permette di specificare politiche di sicurezza con una sintassi generica, ma in questo modo si è riscontrato il problema di dover tradurre la policy nel linguaggio concreto dell'attuatore della funzionalità di sicurezza, per questo motivo si è dovuta trovare una metodologia valida a far si che si eviti di dover generare un traduttore apposito per ogni linguaggio di basso livello.

L'aspetto pratico della tesi è stato quello di realizzare uno strumento per la creazione del linguaggio astratto di una specifica NSF utilizzando un approccio Model-driven, e la realizzazione di uno strumento di traduzione generico che permetta la traduzione di una policy espressa nel linguaggio generico di una NSF nel rispettivo linguaggio specifico. Questo approccio permette di gestire le policy e le capacità di sicurezza in modo astratto.

Partendo dall'ambiente specificato da I2NSF, sono stati approfonditi gli argomenti relativi alle funzionalità di sicurezza NSF e quindi il paradigma Network Function Virtualization NFV e Software Defined Networking SDN.

Utilizzando tecniche e linguaggi altamente conosciuti quali UML e XML, ed utilizzando pattern specifici, è stato possibile ottenere una rappresentazione delle capacità di sicurezza generica, facilmente comprensibile e largamente utilizzabile. Inoltre la possibilità di esprimere politiche di sicurezza a livello generico utilizzando un linguaggio astratto, definito appositamente per una NSF, tenendo conto delle sue funzionalità di sicurezza, permette una definizione standard di politiche di sicurezza mediante un linguaggio espressivo come il linguaggio XML. La difficoltà principale è sorta nell'esprimere adeguatamente i parametri delle classi di associazione riguardanti la classe Translator e la classe Adapter, per mezzo delle quali i tool proposti adattano la generazione del linguaggio astratto della NSF e la conseguente traduzione nel linguaggio specifico di basso livello della NSF. La principale difficoltà viene riscontrata solo durante l'assegnazione delle capacità di

sicurezza della NSF, una volta definita la configurazione delle classi Translator ed Adapter i tool proposti si adeguano agli input, questo grazie ad un approccio Model-driven. Un altro vantaggio portato dalla definizione del linguaggio astratto dipendente dalle capacità di sicurezza è la trasportabilità delle policy definite in tale linguaggio, infatti la traduzione di policy nel linguaggio concreto del dispositivo è dipendente solo dalle caratteristiche della classe Adapter, quindi per due NSF alle quali viene assegnata la stessa policy, quindi che hanno almeno l'insieme di funzionalità di sicurezza necessarie alla policy in comune, il linguaggio astratto permette l'interscambio della policy stessa, dato che essa viene definita allo stesso modo nel linguaggio generico di entrambe le NSF ma dalla quale si ottiene una traduzione differente.

Utilizzando il CapDM proposto si è ottenuta una descrizione delle capacità di sicurezza sufficientemente ampia per coprire i casi di lavoro del firewall iptables. Infatti, quest'ultimo è stato utilizzato per verificare l'applicabilità della soluzione e la validità degli strumenti proposti. La soluzione proposta si può applicare teoricamente ad ogni NSF che possa mappare le proprie capacità di sicurezza in quelle espresse nel CapDM. Inoltre l'utilizzo dell'approccio Model-driven offre la possibilità di aggiungere, modificare, rimuovere capacità di sicurezza dal CapDM permettendo la diretta applicabilità di queste ultime per caratterizzare le istanze di NSF, senza dover modificare il codice degli strumenti ma solo il modello. Nel caso in cui si aggiunga una nuova capacità di sicurezza al CapDM, per poterla utilizzare è sufficiente assegnare quest'ultima alla NSF e definire le classi che permettano la traduzione e la generazione del linguaggio relativo alla nuova capacità, così facendo gli strumenti proposti funzionano senza necessità di modificare il codice.

Data l'attuale realizzazione dei tool e del CapDM si può pensare ad eventuali sviluppi ulteriori, ad esempio aumentare il numero delle capacità di sicurezza descritte dal CapDM per comprendere ulteriori casi, tenendo conto che l'aggiunta di capacità di sicurezza nel modello non causa modifiche al codice dei tool proposti. Oppure permettere al tool di traduzione la possibilità di aggiungere al linguaggio porzioni di esso dovute all'aggiunta di nuove assegnazioni di capacità alla NSF, inoltre si può pensare a migliorare le performance dei tool dato che principalmente il tool di generazione del linguaggio non è ottimizzato. Il lavoro proposto può essere ampliato per permettere di esprimere le politiche in un linguaggio di più alto livello e creare un meccanismo di traduzione delle policy di alto livello nelle policy rappresentate nel linguaggio generico della NSF. Oppure si può inserire il lavoro proposto in un contesto di gestione centralizzata dei controlli di rete. Si può permettere ad un controller centrale di gestire l'inserimento di nuove capacità di sicurezza nel CapDM per permettere un metodo di inserimento o un metodo di modifica standardizzato; tale controller può gestire le possibili NSF conosciute e le istanze attualmente attive nella rete che controlla; Il controller può inoltre gestire i tool proposti per permettere un controllo centralizzato dell'utilizzo di essi per verificare eventuali permessi di utilizzo e la corretta espressione dei comandi da adoperare.

Appendice A

Manuale utente

In questa sezione viene presentato come un utente possa utilizzare i tool proposti. L'unico prerequisito fondamentale per l'utilizzo di tali tool è avere JAVA installato con versione minima 1.8, ed eventualmente le NSF concrete a cui assegnare le configurazioni generate. Essendo eseguibili in formato JAR non è necessaria istallazione di ulteriori programmi. I tool vengono utilizzati mediante linea di comando utilizzando la sintassi: "java -jar nome_jar". Oppure i file JAR possono essere inclusi in progetti JAVA per poter utilizzare le relative funzionalità.

A.1 Tool di trasformazione da XMI ad XSD

Questo tool permette la trasformazione di un file XMI in un file XMLSchema seguendo le caratteristiche con cui modelio trasforma un diagramma delle classi in file XMI. Nell'architettura dell'ambito della tesi, questo tool viene utilizzato per permettere la generazione di un file XMLSchema contenente tutte le capacità di sicurezza e relativi dettagli, rappresentati nel diagramma delle classi del modello dei dati delle capacità di sicurezza (CapDM).

A.1.1 Utilizzo da linea di comando

Per invocare il tool utilizzando la linea di comando si usa la seguente sintassi:

```
java -jar XMItoXSDconverter.jar parametro1 [parametro2]
```

Dove i parametri hanno la seguente funzione:

- parametro1: percorso del file XMI da convertire;
- parametro2: percorso dove generare il file di output. Questo parametro è opzionale, se non viene espresso si crea il file di output nella directory attuale dell'eseguibile con nome del file determinato in modo standard.

L'esecuzione del comando può avere esito positivo o negativo.

Quando l'esecuzione ha esito positivo viene generato un file XMLSchema nella posizione rappresentata dal parametro2 se presente e ne viene confermato l'esito con una stampa a video. Durante la generazione del file di output vengono settati i campi "xmlns" e "targetNamespace" del file stesso utilizzando i valori passati dal parametro2, di conseguenza per riferirsi a tale XMLSchema bisogna utilizzare il "targetNamespace" ed il percorso indicati.

Quando l'esecuzione ha esito negativo il comando genera una risposta contenente i dettagli sul relativo tipo di errore.

A.1.2 Utilizzo come libreria JAVA

Per l'utilizzo del tool come libreria JAVA in un progetto si può istanziare un oggetto di classe "Converter" e chiamare la funzione "convertXMI".

Le definizioni della funzione "convertXMI" sono le seguenti:

- `public boolean convertXMI(String pXMI)`

  permette la conversione di un file XMI in un file XMLSchema con nome definito di default "converted.xsd" nella directory del progetto.

  parametri:

  – pXMI - percorso del file XMI da convertire.

  valore di ritorno:

  – valore booleano che indica l'esito positivo o negativo dell'esecuzione della funzione.

- `public boolean convertXMI(String pXMI, String outPath)`

  permette la conversione di un file XMI in un file XMLSchema.

  parametri:

  – pXMI - percorso del file XMI da convertire;
  – outPath - percorso dove generare il file di output.

  valore di ritorno:

  – valore booleano che indica l'esito positivo o negativo dell'esecuzione della funzione.

A.2 Tool di generazione del linguaggio della NSF

Questo tool permette la generazione di un file XMLSchema contenente il linguaggio della NSF. Il file di output viene generato in funzione delle capacità di sicurezza assegnate alla NSF utilizzando un file in formato XML valido rispetto al file XMLSchema contenente tutte le capacità di sicurezza assegnabili. Nell'architettura dell'ambito della tesi, questo tool viene utilizzato ogni volta che si vuole generare il linguaggio di una NSF alla quale sono state assegnate, rimosse o aggiunte capacità di sicurezza al suo file di assegnazione delle capacità. Questo tool può fare uso di file con funzionalità di metadati. La cartella contenente tali metadati deve chiamarsi "metadati" e deve essere nella stessa cartella che contiene l'eseguibile.

A.2.1 Utilizzo da linea di comando

Per invocare il tool utilizzando la linea di comando si usa la seguente sintassi:

```
java -jar languageGenerator.jar parametro1 parametro2 [parametro3]
```

Dove i parametri hanno la seguente funzione:

- parametro1: percorso del file XSD di riferimento, contenente l'elenco completo delle capacità di sicurezza esistenti (ad esempio il file generato dal comando XMItoXSDconverter.jar);
- parametro2: percorso del file XML contenente l'istanza delle capacità di sicurezza volute per una determinata NSF, con relative classi Translator ed Adapter istanziate;
- parametro3: percorso dove generare il file di output. Questo parametro è opzionale, se non viene espresso si crea il file di output nella directory attuale dell'eseguibile con nome del file determinato in modo standard.

L'esecuzione del comando può avere esito positivo o negativo.

Quando l'esecuzione ha esito positivo viene generato un file XMLSchema contenente il linguaggio della NSF, nella posizione rappresentata dal parametro relativo e viene confermato l'esito con una stampa a video. Durante la generazione del file viene utilizzato il file XML passato come parametro2 per avere le informazioni delle eventuali classi Translator definite per specializzare il linguaggio della NSF. Questo tool può eventualmente generare anche alcuni file di metadati. All'interno di tali file si possono trovare informazioni riguardo alla relazione tra valore numerico e valore alfabetico di determinate enumerazioni. Gli eventuali file relativi ai metadati si troveranno nella stessa posizione del file XMLSchema di output.

Quando l'esecuzione ha esito negativo il comando genera una risposta contenente i dettagli sul relativo tipo di errore.

A.2.2 Utilizzo come libreria JAVA

Per l'utilizzo del tool come libreria JAVA in un progetto si può istanziare un oggetto di classe "LanguageModelGenerator" e chiamare la funzione "generateLanguage".

Le definizioni del costruttore della classe "LanguageModelGenerator" sono i seguenti:

- `public LanguageModelGenerator(String xsd, String xml)`

  permette di istanziare un oggetto di tipo "LanguageModelGenerator"

  parametri:

  - xsd - percorso del file XMLSchema contenente le'elenco completo delle capacità di sicurezza;
  - xml - percorso del file XML contenente l'istanza delle capacità di sicurezza assegnate ad una determinata NSF, con relative classi Translator ed Adapter istanziate.

- `public LanguageModelGenerator(String xsd, String xml, String outputName)`

  permette la conversione di un file XMI in un file XMLSchema.

  parametri:

  - xsd - percorso del file XMLSchema contenente le'elenco completo delle capacità di sicurezza;
  - xml - percorso del file XML contenente l'istanza delle capacità di sicurezza assegnate ad una determinata NSF, con relative classi Translator ed Adapter istanziate;
  - outputName - percorso e nome dell'output da generare.

La definizione della funzione "generateLanguage" è la seguente:

- `public boolean generateLanguage()`

  attua la conversione effettiva dei file coi quali l'istanza è stata generata.

  valore di ritorno:

  - valore booleano che indica l'esito positivo o negativo dell'esecuzione della funzione.

A.3 Tool di traduzione nel linguaggio di basso livello della NSF

Questo tool permette la generazione di un file di testo contenente la policy espressa nel linguaggio di basso livello della NSF. Il file di output viene generato in funzione delle capacità di sicurezza presenti nella policy di sicurezza espressa nel linguaggio generico della NSF. Nell'architettura dell'ambito della tesi, questo tool viene utilizzato ogni volta che si vuole tradurre una policy per la data NSF.

A.3.1 Utilizzo da linea di comando

Per invocare il tool utilizzando la linea di comando si usa la seguente sintassi:

```
java -jar translatorAdapter.jar parametro1 parametro2 parametro3 [parametro4]
    [parametro5] [parametro6] [parametro7] [parametro8] [parametro9]
```

Dove i parametri hanno la seguente funzione:

- parametro1: percorso del file XSD contenente il linguaggio generico della NSF scelta;
- parametro2: percorso del file XML contenente l'istanza delle capacità di sicurezza assegnate alla NSF con relative classi Translator e Adapter;
- parametro3: percorso del file XML contenente l'istanza della policy a tradurre;
- parametro4: percorso del file dove si vuole l'output. Questo parametro è opzionale, se non viene espresso si crea il file di output nella directory attuale dell'eseguibile con nome del file determinato in modo standard.
- parametro5: parametro opzionale che permette di aggiungere all'output una parte statica alfanumerica all'inizio di ogni regola, questo parametro deve iniziare con la sequenza di caratteri "+s";
- parametro6: parametro opzionale che permette di aggiungere all'output una parte statica alfanumerica alla fine di ogni regola, questo parametro deve iniziare con la sequenza di caratteri "+e";
- parametro7: parametro opzionale per forzare la generazione delle regole valide anche nel caso in cui alcune regole della policy non rispettino determinati criteri di dipendenza, si esprime come "-f";
- parametro8: parametro opzionale che permette di andare a capo dopo il parametro iniziale di ogni regola (cioè il parametro5), questo parametro deve iniziare con la sequenza di caratteri "+crs";
- parametro9: parametro opzionale che permette di andare a capo prima del parametro finale di ogni regola (cioè il parametro6), questo parametro deve iniziare con la sequenza di caratteri "+cre."

L'ordine di inserimento dei parametri è importante solo per i primi tre, i parametri opzionali possono essere inseriti in qualsiasi ordine.

L'esecuzione del comando può avere esito positivo o negativo.

Quando l'esecuzione ha esito positivo viene generato un file di testo contenente la traduzione nel linguaggio di basso livello della NSF della policy espressa nel linguaggio generico della NSF. Il file viene generato nella posizione come espresso dal parametro4. Il formato del contenuto del file di output è fortemente influenzato dalla presenza o assenza dei parametri opzionali passati al comando.

Quando l'esecuzione ha esito negativo il comando genera una risposta contenente i dettagli sul relativo tipo di errore.

A.3.2 Utilizzo come libreria JAVA

Per l'utilizzo del tool come libreria JAVA in un progetto si può istanziare un oggetto di classe "NSFTranslatorAdapter" e chiamare la funzione "translate".

La definizione della funzione "translate" è la seguente:

- `public void translate(String xsd, String xmlAdapter, String xmlPolicy, String outputName, String startString, String endString, String forced, boolean scr, boolean ecr)`

parametri:

  - xsd: percorso del file XSD contenente il linguaggio generico della NSF scelta;
  - xmlAdapter: percorso del file XML contenente l'istanza delle capacità di sicurezza assegnate alla NSF con relative classi Translator e Adapter;
  - xmlPolicy: percorso del file XML contenente l'istanza della policy a tradurre;
  - outputName: percorso del file dove si vuole l'output. Questo parametro è opzionale, se non viene espresso si crea il file di output nella directory attuale dell'eseguibile con nome del file determinato in modo standard.
  - startString: parametro opzionale che permette di aggiungere all'output una parte statica alfanumerica all'inizio di ogni regola;
  - endString: parametro opzionale che permette di aggiungere all'output una parte statica alfanumerica alla fine di ogni regola;
  - forced: parametro opzionale per forzare la generazione delle regole valide anche nel caso in cui alcune regole della policy non rispettino determinati criteri di dipendenza;
  - scr: parametro opzionale che permette di andare a capo dopo il parametro iniziale di ogni regola (cioè startString);
  - ecr: parametro opzionale che permette di andare a capo prima del parametro finale di ogni regola (cioè endString).

valore di ritorno:

  - valore booleano che indica l'esito positivo o negativo dell'esecuzione della funzione.

A.4 Tool di validazione

Questo tool permette la validazione di un file XML rispetto ad un file XMLSchema entrambi passati come parametri al tool. Nell'architettura dell'ambito della tesi questo tool viene utilizzato ogni qualvolta si debba istanziare un file XML necessario ad uno dei tool proposti.

A.4.1 Utilizzo da linea di comando

Per invocare il tool utilizzando la linea di comando si usa la seguente sintassi:

```
java -jar validate.jar parametro1 parametro2
```

Dove i parametri hanno la seguente funzione:

- parametro1: percorso del file XML Schema di riferimento;
- parametro2: percorso del file XML da convalidare.

L'esecuzione del comando può avere esito positivo o negativo.

Quando l'esecuzione ha esito positivo viene restituita la stringa "true".

Quando l'esecuzione ha esito negativo il comando genera una risposta contenente i dettagli sul relativo tipo di errore.

A.4.2 Utilizzo come libreria java

Per l'utilizzo del tool come libreria JAVA in un progetto si può istanziare un oggetto di classe "Validation" e chiamare la funzione "validate".

La definizione della funzione "validate" è la seguente:

- `public boolean validate(String xsd, String xml)`

  parametri:

  - xsd: percorso del file XML Schema di riferimento;
  - xml: percorso del file XML da convalidare.

  valore di ritorno:

  - valore booleano che indica l'esito positivo o negativo dell'esecuzione della funzione.

Appendice B

Manuale programmatore

In questa sezione vengono riportate le specifiche del manuale programmatore relative al modello delle capacità e ad ogni tool, in particolare verranno spiegate dettagliatamente le architetture dei tool e le varie funzioni che compongono il codice di ogni tool. La caratteristica che accomuna i tool è che sono scritti tutti in JAVA. Per la gestione dei file XML sono state utilizzate le API offerte da Java API for XML Processing (JAXP). Le librerie aggiunte appartengono al progetto "Apache Xerces Project", scaricabili direttamente dal sito relativo al progetto[1], il pacchetto specifico utilizzato è "xerces-2_12_0" e le librerie presenti nei progetti sono:

- resolver.jar;
- serializer.jar;
- xercesImpl.jar;
- xml-apis.jar.

B.1 Gestione del CapDM utilizzando modelio

Il progetto proposto per la creazione del CapDM si sviluppa in un insieme di diagrammi delle classi relazionati tra di loro. Partendo dalla definizione dell'elemento più generico, ovvero la classe "SecurityCapability", è stata generata la prima gerarchia, ossia la divisione nei compiti definiti dalla 6-tupla come in Figura 5.2. Utilizzando la relazione di generalizzazione sono state generate ulteriori classi di suddivisione, fino ad arrivare ad elementi "foglia", i quali determinano una capacità di sicurezza concreta.

Le relazioni tra i molteplici diagrammi delle classi sono rappresentati dalla Figura B.1, dove viene rappresentato anche il fatto che il Capability Information Model (CapIM, Figura 5.1) utilizza le capacità di sicurezza definite nel Capability Data Model (CapDM). Dalla definizione della capacità di sicurezza generica viene definita la 6-tupla con la relazione di genericità, utilizzando sempre questa relazione vengono definiti i gruppi di appartenenza delle capacità di sicurezza ed eventualmente ulteriori sottogruppi, fino a definire la capacità foglia concreta in modo generico. Questa capacità può utilizzare i tipi che sono definiti nel diagramma delle classi definito apposta per i tipi.

B.1.1 Aggiungere una nuova capacità

Per aggiungere una nuova capacità si può procedere con i seguenti passaggi:

[1] http://xerces.apache.org/mirrors.cgi

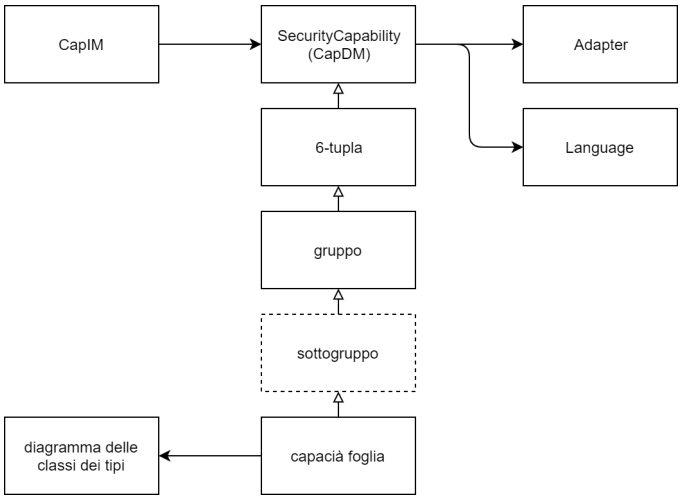

Figura B.1. Relazioni tra i diagrammi delle classi.

- identificare a quale gruppo appartiene tale capacità;
- identificare il diagramma delle classi utilizzato per tale gruppo di capacità;
- creare una nuova classe nel diagramma delle classi del gruppo identificato;
- definire la nuova classe con nome generico;
- definire eventuali dettagli relativi agli attributi necessari per manipolare la nuova capacità.

Se gli attributi definiti per la nuova classe non sono di tipo generico, ad esempio di tipo stringa o di tipo intero, si può definire un nuovo tipo generico procedendo con i seguenti passaggi:

- identificare come comporre il nuovo tipo;
- creare una nuova classe nel diagramma delle classi dei tipi;
- definire la nuova classe con nome generico;
- definire i dettagli dei nuovi attributi necessari per manipolare la nuova capacità;
- assegnare al parametro della nuova capacità creato precedentemente il tipo della classe appena definita.

In questo modo si possono definire dei tipi standard che ogni nuova capacità di sicurezza potrà utilizzare.

B.2 Tool di trasformazione da XMI ad XSD

Questo tool permette la trasformazione di un file XMI in un file XMLSchema seguendo le caratteristiche con cui modelio trasforma un diagramma delle classi in file XMI. Nell'architettura dell'ambito della tesi, questo tool viene utilizzato per permettere la generazione di un file XMLSchema contenente tutte le capacità di sicurezza e relativi dettagli, rappresentati nel diagramma delle classi del modello dei dati delle capacità di sicurezza (CapDM). Il tool sviluppa l'output seguendo il workflow in Figura B.2.

I passaggi principali del funzionamento di questo tool sono i seguenti:

- lettura del file di input;

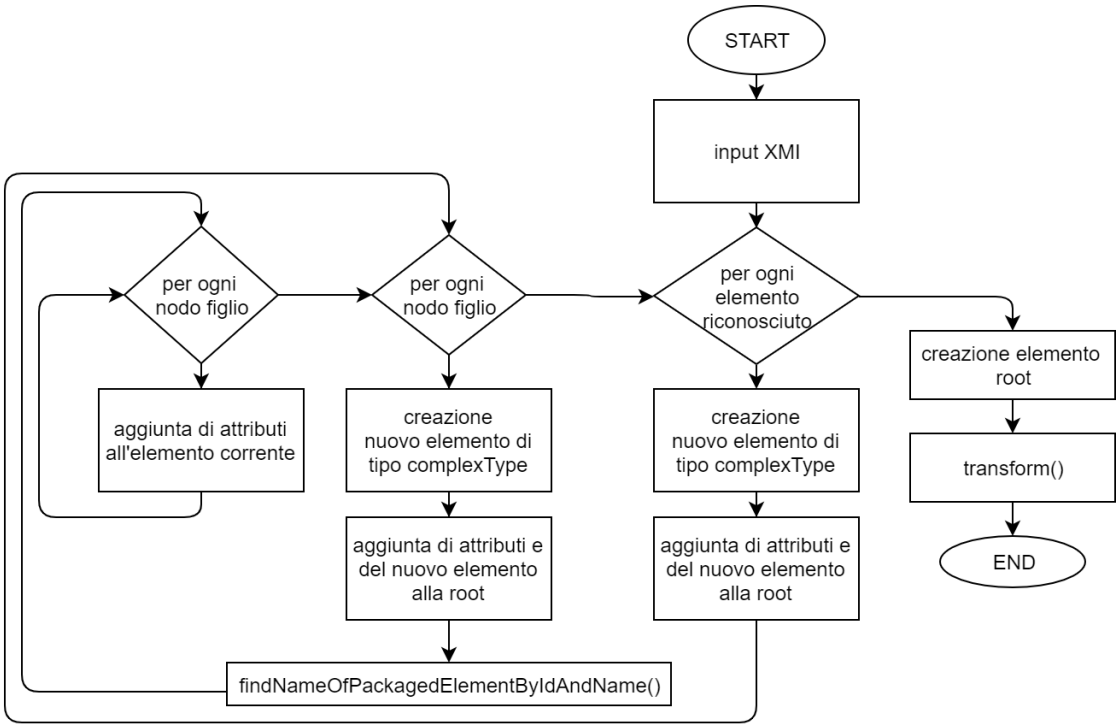

Figura B.2. Workflow del Tool di trasformazione da XMI ad XSD.

- ciclo per ogni elemento. Il punto di riferimento del tool è un tag chiamato "packagedElement" che contiene tutte le classi generate in quella cartella, il cui tipo determina il tipo di elemento che si sta analizzando;
- riconosciuto un elemento corretto si genera un elemento di tipo complexType da inserire nel file di output;
- ogni elemento generico può essere costituito da ulteriori elementi quindi si esplora ogni elemento fino a trovare gli attributi utili a generare il tipo complesso necessario al file di output;
- conclusa la traduzione di ogni elemento presente nel file di input viene generato un elemento concreto che funzioni da elemento radice del file XMLSchema;
- creata tutta la struttura la funzione "transform()" si occupa della generazione concreta del file di output.

B.2.1 Architettura del tool

L'architettura del tool proposto è strutturata in due classi principali, rappresentate nella Figura B.3, ed una classe di supporto per la creazione di oggetti da inserire nel documento; tali classi sono composte come spiegato di seguito:

- Converter: classe che si occupa della gestione del file di input e di riconoscere i dettagli utili alla creazione degli elementi per il file di output. Questa classe non definisce variabili di istanza. Questa classe è composta dalle seguenti funzioni:

  – `public Converter()`

  funzione costruttore che permette di istanziare un oggetto corrispondente alla classe "Converter";

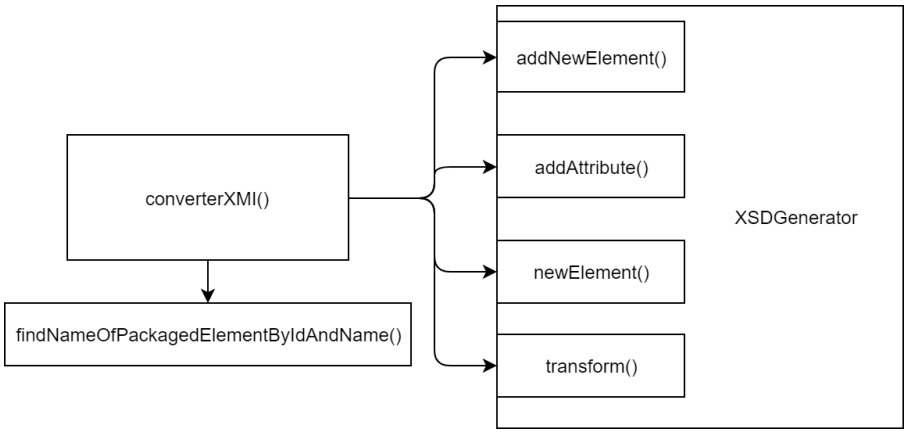

Figura B.3. Architettura del Tool di trasformazione da XMI ad XSD.

– `public boolean convertXMI();`

questa funzione si occupa della lettura del file di input, di riconoscere gli elementi utili per la generazione dell'output ed interagisce con le altre funzioni per la finalità del tool;

– `private String findNameOfPackagedElementByIdandType(NodeList nl, String id, String classe)`

questa funzione si occupa nel dettaglio di trovare il nome, in formato stringa, di un elemento appartenente alla lista dei nodi passati come parametro. Per trovare il nome corretto viene ricercato nella lista di nodi passata l'elemento che ha il valore "classe" nell'attributo "xmi:type" ed il valore "id" nell'attributo "xmi:id". Nel caso in cui la ricerca non andasse a buon fine viene ritornato "null".

- XSDgenerator: classe che si occupa della gestione degli elementi riconosciuti e convertiti con il formato XMLSchema, si occupa anche della trasformazione effettiva dell'elemento generato durante la lettura del file di input e della generazione effettiva del file di output. Possiede un elemento principale che costituisce la radice del file di output.

  Questa classe definisce le seguenti variabili di istanza:

  – `private final static String NS_PREFIX = "xs:";`

  questo campo permette di definire con quale prefisso saranno generati tutti gli elementi;

  – `private Document doc;`

  questo campo contiene il documento di output a cui si riferisce questa istanza;

  – `private Element schemaRoot;`

  questo campo contiene l'elemento radice del documento di output dell'istanza;

  – `private NameTypeElementMaker elMaker;`

  questo campo contiene l'elemento che permette di generare nuovi elementi compatibili con il documento di output.

  Questa classe è composta dalle seguenti funzioni:

  – `public XSDgenerator()`

  funzione costruttore che permette di istanziare un oggetto corrispondente alla classe "XSDgenerator" e si occupa di creare gli oggetti principali della classe;

  – `public boolean transform(String outputName)`

questa funzione si occupa della generazione dell'output relativo al nuovo file XSD utilizzando il parametro "outputName" come riferimento per il nome ed il path di destinazione del file di output;

- `public Element newElement(String name)`

  questa funzione genera un nuovo elemento, il parametro "name" viene utilizzato per dare il nome all'elemento da generare;

- `public void addAttribute(Element element, String nameAttr, String attrValue)`

  questa funzione si occupa di aggiungere all'elemento "element" un attributo di nome "nameAttr" e di valore "attrValue";

- `public void addNewElement(Element e)`

  questa funzione aggiunge un nuovo elemento alla radice dello schema da generare;

- NameTypeElementMaker: classe di supporto alla classe "XSDgenerator" che mette a disposizione due funzioni per la gestione di oggetti nel documento desiderato.

  Questa classe definisce le seguenti variabili di istanza:

  - `private String nsPrefix;`

    questo campo permette di definire con quale prefisso saranno generati tutti gli elementi;

  - `private Document doc;`

    questo campo contiene il documento di output a cui si riferisce questa istanza;

  Le funzioni che compongono la classe sono:

  - `public NameTypeElementMaker(String nsPrefix, Document doc)`

    questo costruttore permette di definire a quale documento si riferisce la classe e se esiste una stringa di prefisso;

  - `public Element createElement(String elementName)`

    questa funzione si occupa di creare un elemento nel documento assegnato alla classe in modo corretto, aggiungendo al parametro "elementName" il prefisso assegnato alla classe;

  - `public void setAttribute(Element element, String nameAttr, String attrValue)`

    questa funzione si occupa di aggiungere all'elemento "element" un attributo di nome "nameAttr" e di valore "attrValue".

B.2.2 Eventuali modifiche

Si possono aggiungere funzionalità alla funzione "convertXMI()" permettendo di riconoscere più tipologie di elementi presenti nel file XMI. Attualmente riconosce e sa manipolare gli elementi "uml:Class" e "uml:Enumeration", si possono implementare porzioni di codice che riconoscano ad esempio gli elementi "uml:Association" e "uml:AssociationClass". Ai fini dello scopo dell'elaborato tali elementi non erano necessari. Per riconoscere gli elementi necessari è stato utilizzato il costrutto condizionale "if"; per aggiungere il riconoscimento degli elementi sopra citati si possono aggiungere porzioni di codice come

```
else if
    (element.getAttribute("xmi:type").equalsIgnoreCase("uml:Association"))
    {.....}
```

```
else if
    (element.getAttribute("xmi:type").equalsIgnoreCase("uml:AssociationClass"))
    {.....}
```

ed implementarli secondo gli scopi desiderati.

B.3 Tool di generazione del linguaggio della NSF

Questo tool permette la generazione di un file XMLSchema contenente il linguaggio della NSF. Il file di output viene generato in funzione delle capacità di sicurezza assegnate alla NSF utilizzando un file in formato XML valido rispetto al file XMLSchema contenente tutte le capacità di sicurezza assegnabili. Nell'architettura dell'ambito della tesi, questo tool viene utilizzato ogni volta che si vuole generare il linguaggio di una NSF alla quale sono state assegnate, rimosse o aggiunte capacità di sicurezza al suo file di assegnazione delle capacità. Questo tool può fare uso di file con funzionalità di metadati. La cartella contenente tali metadati deve chiamarsi "metadati" e deve essere nella stessa cartella che contiene il tool, o il sorgente del tool. I metadati consistono nella relazione tra numero intero e valore stringa di eventuali enumerazioni che necessitano questa relazione, e sono strutturati in modo che per ogni riga ci sia prima il valore numerico e, separato da uno spazio, il valore alfabetico. I file dei metadati proposti riguardano i protocolli e le porte di servizio.

Il tool sviluppa l'output seguendo il workflow in Figura B.4.

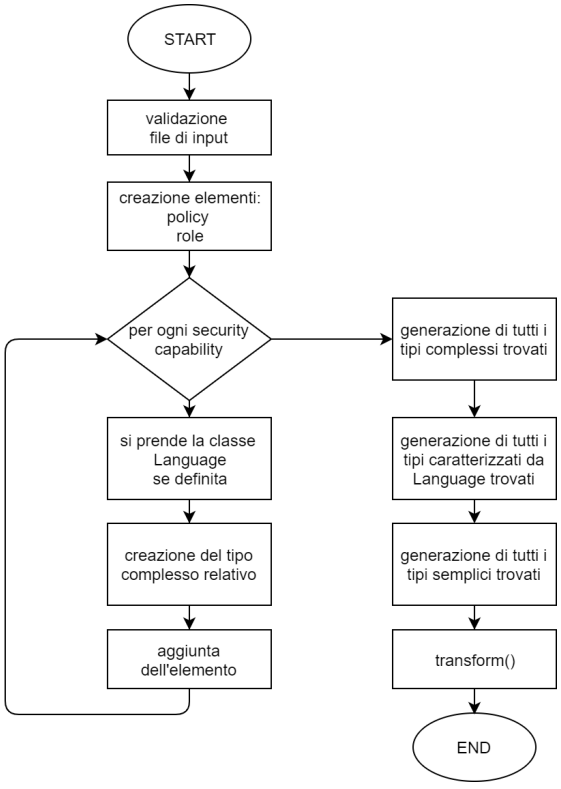

Figura B.4. Workflow del Tool di generazione del linguaggio della NSF.

I passaggi principali del funzionamento di questo tool sono i seguenti:

- validazione dei file in input;
- generazione degli elementi policy e role secondo i criteri che serviranno per istanziare delle policy;

- ciclo per ogni capacità di sicurezza presente nel file di configurazione della NSF. Il punto di riferimento del tool è il tag chiamato “securityCapability” che permette di riconoscere ogni capacità di sicurezza definita;
- da ogni elemento “securityCapability” viene preso, se presente, l’elemento con tag chiave “language”;
- da ogni elemento “securityCapability” viene preso il tipo di capacità di sicurezza definiti nell’attributo “xsi:type” e viene inserito in una lista di stringhe;
- conclusa l’esplorazione delle capacità di sicurezza definite vengono generati i tipi complessi che sono stati incontrati;
- vengono generati gli eventuali tipi caratterizzati dalla classe “Language”;
- vengono generati i tipi semplici necessari per le capacità di sicurezza assegnate;
- viene utilizzata la funzione “transform()” che permette la generazione dell’output.

B.3.1 Architettura del tool

L’architettura del tool proposto è strutturata in due classi principali, ed è rappresentata in Figura B.5 ed in Figura B.6. L’architettura è divisa in più classi per permettere la gestione dell’interpretazione del file in input, da parte della prima classe “LanguageModelGenerator”, per gestire la struttura e tutti gli elementi che serviranno per generare l’output stesso, nella seconda classe “XSDgenerator” ed una ulteriore classe per la gestione della creazione degli elementi effettivi appartenenti al documento di output. Inoltre le immagini separate permettono di dividere le funzioni coinvolte durante la generazione del linguaggio generico, Figura B.5, e le funzioni utilizzate durante la gestione della classe “LanguageConstraint” di una capacità di sicurezza, Figura B.6.

L’architettura del tool è caratterizzata da una funzione principale “geneareLanguage()” che utilizza tutte le altre funzioni. L’architettura si sviluppa nelle seguente modo:

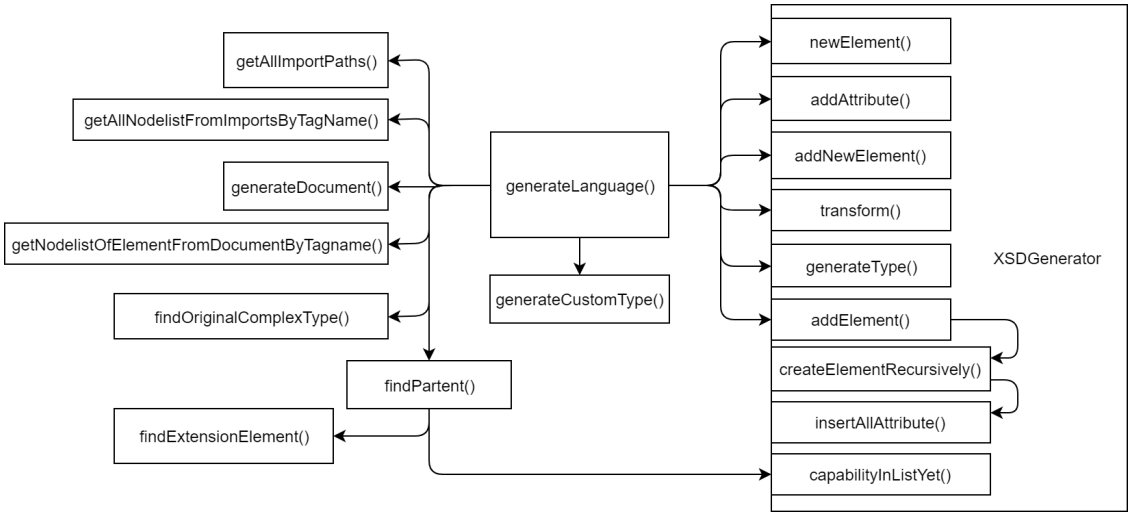

Figura B.5. Architettura del Tool di generazione del linguaggio della NSF, prima parte.

- la classe “LanguageModelGenerator” si occupa della lettura del file di input e della gestione delle strutture ricevute.

  Questa classe definisce le seguenti variabili di istanza:

  - `private String xsd;`

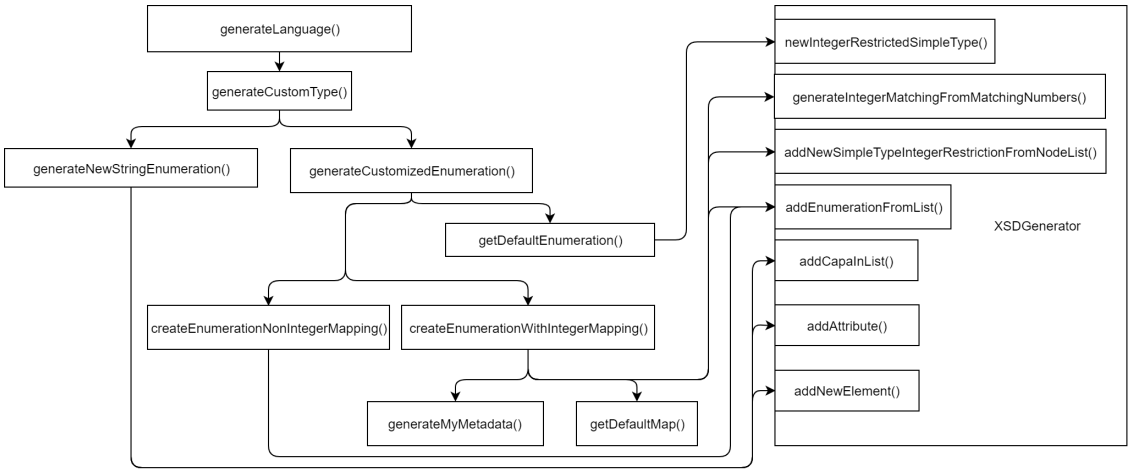

Figura B.6. Architettura del Tool di generazione del linguaggio della NSF, seconda parte.

campo che contiene il percorso del file XMLSchema di riferimento, file che contiene la rappresentazione XMLSchema del CapDM;

- `private String xml;`

campo che contiene il percorso del file XML di riferimento, file che contiene l'elenco delle capacità di sicurezza assegnate ed eventuali classi "Language" ed "Adapter";

- `private String outputName;`

campo che contiene il percorso dove verrà generato il file di output e quindi il nome del file stesso.

- `private List<String> imports;`

campo che contiene la lista dei percorsi ai file che compongono il CapDM;

- `private List<NodeList> complexTypeNodeLists;`

campo che contiene la lista dei nodi complessi appartenente a tutti i file che compongono il CapDM;

- `private List<NodeList> simpleTypeNodeLists;`

campo che contiene la lista dei nodi semplici appartenente a tutti i file che compongono il CapDM;

- `private XSDgenerator gen;`

campo che contiene l'oggetto di istanza della classe "XSDgenerator";

- `private List<String> metadataPath;`

campo che contiene la lista dei percorsi riferiti ai file dei metadati generati durante la generazione del linguaggio.

Questa classe è composta dalle seguenti funzioni:

- `public LanguageModelGenerator(String xsd, String xml, String outputName)`

funzione costruttore permette di istanziare oggetti della classe "LanguageModelGenerator", istanziando le variabili della classe stessa tra cui quelle riferite ai parametri passati col costruttore;

- `public boolean generateLanguage()`

  questa funzione si occupa di generare il linguaggio della NSF. Si occupa della generazione degli elementi necessari per la gestione della policy, quindi crea l'elemento "policy" ed il tipo complesso "rule", inoltre si occupa di leggere tutte le capacità di sicurezza e relativi "language" nel file di input e di trasformarli in tipi complessi da inserire nel file di output.

- `private List<String> getAllImportPaths()`

  questa funzione genera un lista contenente tutti i path degli elementi "xs:import" presenti nel file di input. Questa funzione è necessaria nel caso in cui il CapDM tradotto come file XMLSchema dovesse essere diviso in molteplici file XSD, in questo modo è sufficiente che il file XSD principale contenga i dettagli dei percorsi necessari per raggiungere tutti i file XSD ulteriori;

- `private List<NodeList> getAllNodelistFromImportsByTagName(List<String> imports, String tag)`

  questa funzione genera una lista di elementi nodeList contenente la lista dei nodi contenenti elementi con quel determinato "tag" per ogni documento nella lista degli "imports";

- `private static NodeList getNodelistOfElementFromDocumentByTagname (Document d, String tagname)`

  questa funzione ritorna la lista dei nodi contenenti gli elementi con quel determinato "tagname" appartenenti al documento "d";

- `private static Document generateDocument(String path)`

  questa funzione genera un oggetto di tipo "Document" in base al percorso specificato nella variabile "path";

- `private static Element findOriginalComplexType (NodeList complextype, String capa)`

  questa funzione cerca un elemento che abbia come attributo "name" il valore contenuto in "capa" appartenente ad una lista di nodi "complextype" e ritorna tale elemento di tipo "Element" oppure "null" se non è stato trovato;

- `private Element findParent (Element complextype)`

  questa funzione risale, in modo ricorsivo, la catena degli elementi da cui deriva l'elemento "complextype", fino a raggiungere l'elemento generico "SecurityCapability". Ad ogni iterazione viene aggiunto l'elemento corrente agli elementi necessari da utilizzare per il linguaggio, eventuali elementi già presenti non vengono ulteriormente inseriti;

- `private Element findExtensionElement (Element e)`

  questa funzione cerca se nei vari nodi interni all'elemento "e" passato esiste un elemento "xs:extension", se esiste allora ritorna tale elemento, altrimenti ritorna "null".

- inoltre la classe "LanguageModelGenerator" si occupa della gestione dei dettagli definiti negli elementi di tipo "Language" del file di input e della gestione delle strutture ricevute. Tenendo conto della Figura 5.6 la classe "LanguageModelGenerator" è gestita dalle seguenti funzioni:

  - `private boolean generateCustomType(Element element, NodeList n)`

    questa funzione permette la generazione del linguaggio della NSF basata sui dettagli esposti nella classe "LanguageConstraint". Viene chiamata solo nel caso in cui siano stati riconosciuti elementi di tag "language". Riconosce gli elementi contenuti nella classe "LanguageConstraint" e li gestisce utilizzando le apposite funzioni;

- `private boolean generateCustomizedEnumeration(String enumerationName, Element modifyDefaultEnumeration, NodeList modeledNL)`

questa funzione genera gli elementi che rispettano i dettagli definiti negli elementi "language" incontrati. Questa funzione si basa strettamente sulla struttura della classe "LanguageConstraint". Le operazioni effettuate da tale funzione dipendono dal contenuto dell'elemento "language" che si sta analizzando. Inoltre in questa funzione sono definiti i metodi di utilizzo di eventuali metadati come nel caso dei tipi di protocollo o i tipi di porte. Se si vuole modificare il comportamento del tool nei confronti della classe "LanguageConstraint" questa è la funzione da cui si deve iniziare;

- `private boolean createEnumerationWithIntegerMapping(String enumerationName, NodeList addNewValueNodeList, NodeList renameValueNodeList, NodeList removeValueNodeList, NodeList addExistingValueNodeList, Element defaultEnumeration, NodeList setNumericRangeNodeList, NodeList generateIntegerMatching, String metadata, List<String> capabilityAndAttributesToBeChanged)`

questa funzione si occupa di creare l'enumerazione con i parametri definiti nell'elemento "language", l'eventuale tipo intero e si occupa di modificare il tipo dell'attributo interessato se presente. Inoltre si occupa della generazione di eventuali metadati dovuti alla relazione tra intero e valore stringa dell'enumerazione;

- `private void generateMyMetadata(Map<String, Integer> nameValueMap, String metadata)`

questa funzione si occupa realmente della generazione del file di output relativo ai metadati. I metadati vengono ordinati in ordine numerico crescente;

- `private static Map<String, Integer> sortByValue(Map<String, Integer> unsortMap)`

questa funzione si occupa di ordinare una mappa secondo i suoi valori interi;

- `private Map<String, Integer> getDefaultMap(String metadata)`

questa funzione genera una mappa di default costruita utilizzando i valori contenuti nel file dei metadati passato come parametro;

- `private boolean createEnumerationNonIntegerMapping(String name, NodeList addNewValueNodeList, NodeList renameValueNodeList, NodeList removeValueNodeList, NodeList addExistingValueNodeList, Element defaultEnumeration)`

questa funzione genera il tipo enumerazione senza la relazione tra intero e valore stringa. Inizialmente genera una lista di valori stringa che viene passata ad un'altra funzione appartenente alla classe "XSDgenerator" che si occuperà di creare l'effettivo oggetto adatto all'output;

- `private Element getDefaultEnumeration(String name, NodeList modeledNL)`

questa funzione si occupa di cercare in una lista di nodi "modeledNL" l'elemento con attributo "name" equivalente al contenuto del parametro della funzione "name";

- `private boolean generateNewStringEnumeration(String name, NodeList valueNL)`

questa funzione genera una nuova enumerazione di nome il contenuto del parametro "name", se non è ancora nella lista degli elementi già generati. Tale enumerazione viene generata esclusivamente sulla base dei valori contenuti in ogni nodo della lista "valueNL";

- la classe "XSDgenerator" si occupa della gestione degli elementi riconosciuti e convertiti con il formato XMLSchema, si occupa anche della trasformazione effettiva dell'elemento generato durante la lettura del file di input e della generazione effettiva del file di output. Possiede un elemento principale che costituisce la radice del file di output.

  Questa classe definisce le seguenti variabili di istanza:

  - `private final static String NS_PREFIX = "xs:";`

    questo campo permette di definire con quale prefisso saranno generati tutti gli elementi;

  - `private Document doc;`

    questo campo contiene il documento di output a cui si riferisce questa istanza;

  - `private Element schemaRoot;`

    questo campo contiene l'elemento radice del documento di output dell'istanza;

  - `private NameTypeElementMaker elMaker;`

    questo campo contiene l'elemento che permette di generare nuovi elementi compatibili con il documento di output.

  - `private TreeSet<String> capability;`

    questo campo contiene l'elenco dei nomi delle capacità di sicurezza incontrati;

  - `private List<String> type;`

    questo campo contiene l'elenco dei nomi dei tipi complessi o semplici incontrati;

  Questa classe è composta dalle seguenti funzioni:

  - `public XSDgenerator()`

    questo costruttore permette di istanziare un oggetto corrispondente alla classe "XSDgenerator" e si occupa di creare gli oggetti principali della classe;

  - `public boolean transform(String outputName)`

    questa funzione si occupa della generazione dell'output relativo al nuovo file XSD utilizzando il parametro "outputName" come riferimento per il nome ed il path di destinazione del file di output;

  - `public Element newElement(String name)`

    questa funzione genera un nuovo elemento, il parametro "name" viene utilizzato per dare il nome all'elemento da generare;

  - `public void addAttribute(Element element, String nameAttr, String attrValue)`

    questa funzione si occupa di aggiungere all'elemento "element" un attributo di nome "nameAttr" e di valore "attrValue";

  - `public void addNewElement(Element e)`

    questa funzione aggiunge un nuovo elemento alla radice dello schema da generare;

  - `public void addElement(Element e)`

    questa funzione aggiunge un nuovo elemento alla radice dello schema da generare utilizzando una copia dell'elemento passato come parametro;

  - `private Element createElementRecursively(Element e)`

questa funzione genera ricorsivamente una copia dell'elemento passato come parametro, quindi di tutte le sue caratteristiche interne. Questa funzione evita di generare le classi o attributi relativi ad elementi "adapter" o "language", non necessari nel linguaggio della NSF;

- `private void insertAllAttributes(Element fromElement, Element toElement)`

questa funzione aggiunge tutti gli attributi di un elemento, "fromElement", in un altro elemento "toElement". Inoltre se viene riconosciuto un attributo "type" allora viene aggiunto alla lista dei tipi incontrati, lista necessaria per generare i complex type del linguaggio della NSF;

- `public void addCapaInList (String s)`

questa funzione inserisce il nome passato come parametro in una lista di stringhe senza duplicati;

- `public boolean capabilityInListYet (String s)`

questa funzione valuta se il valore passato come parametro è già presente nella lista. Questa funzione è utilizzata assieme alla lista delle capacità che sono già state incontrate durante la traduzione;

- `public void generateType(NodeList typeComplex)`

questa funzione utilizza i nodi appartenenti alla lista dei nodi passata come parametro per generare un nuovo elemento da poter aggiungere alla radice che genererà l'output. Questa funzione utilizza "addElement()" per la generazione di ogni elemento riconosciuto;

- `public void addEnumerationFromList (String name, List<String> valueList)`

questa funzione crea l'elemento enumerazione con il formato XMLSchema, utilizzando come nome dell'enumerazione il parametro "name" ed inserendo tutti i valori della lista passata come parametro "valueList";

- `public void addNewSimpleTypeIntegerRestrictionFromNodeList(List<String> capabilityAndAttributesToBeChanged, NodeList setNumericRangeNodeList)`

questa funzione genera un nuovo elemento utilizzando i valori presenti nella lista di nodi "setNumericRangeNodeList". Il nome viene deciso con il primo valore della lista "capabilityAndAttributesToBeChanged". Questa lista è composta da un primo valore che indica il nome della classe contenente gli attributi ai quali si vuole cambiare tipo, gli ulteriori valori della lista sono i nomi degli attributi ai quali si vuole cambiare il tipo, tali attributi devono appartenere alla classe il cui nome è in posizione 0 della lista;

- `private void changeElementTypeInExistingComplexType(List<String> capabilityAndAttributesToBeChanged, String newTypeName)`

questa funzione si occupa di cercare l'elemento il cui nome è nel primo elemento del parametro "capabilityAndAttributesToBeChanged". Trovato tale elemento la funzione si occupa di modificare il tipo degli attributi i cui nomi sono i successivi elementi nella lista inserendo il valore del parametro "newTypeName".

- `public void generateIntegerMatchingFromMatchingNumbers(List<String> capabilityAndAttributesToBeChanged, Map<String, Integer> nameValueMapSortedByIntegerValue)`

questa funzione genera una nuova restrizione di tipo intero utilizzando i valori contenuti nella mappa "nameValueMapSortedByIntegerValue", il nome del nuovo tipo viene generato con il valore in prima posizione della lista "capabilityAndAttributesToBeChanged" aggiungendo la stringa "IntegerRestriction". Per completare la generazione questa funzione chiama la funzione "changeElementTypeInExistingComplexType" per modificare il tipo degli attributi interessati;

- `public void newIntegerRestrictedSimpleType(NodeList setNumericRangeNodeList, List<String> capabilityAndAttributesToBeChanged)`

  questa funzione genera una nuova restrizione di tipo intero utilizzando i valori contenuti nella lista "capabilityAndAttributesToBeChanged", il nome del nuovo tipo viene generato con il valore in prima posizione della lista "capabilityAndAttributesToBeChanged" aggiungendo la stringa "IntegerRestriction". Per completare la generazione questa funzione chiama la funzione "changeElementTypeInExistingComplexType" per modificare il tipo degli attributi interessati;

- la classe "NameTypeElementMaker" è una classe di supporto per la generazione di elementi in uno specifico documento. Questa classe viene utilizzata esclusivamente dalle funzioni appartenenti alla classe "XSDgenerator", dato che si occupano della creazione del documento di output.

  Questa classe definisce le seguenti variabili di istanza:

  - `private String nsPrefix;`

    questo campo permette di definire con quale prefisso saranno generati tutti gli elementi;

  - `private Document doc;`

    questo campo contiene il documento di output a cui si riferisce questa istanza;

  Questa classe formata dalle seguenti funzioni:

  - `public NameTypeElementMaker(String nsPrefix, Document doc)`

    questo costruttore permette di definire a quel documento si riferisce la classe e se esiste una stringa di prefisso;

  - `public Element createElement(String elementName)`

    questa funzione si occupa di creare un elemento nel documento assegnato alla classe in modo corretto, aggiungendo al parametro "elementName" il prefisso assegnato alla classe;

  - `public void setAttribute(Element element, String nameAttr, String attrValue)`

    questa funzione si occupa di aggiungere all'elemento "element" un attributo di nome "nameAttr" e di valore "attrValue". In questo caso però viene controllato che i nomi degli attributi non contengano riferimenti ad altri XMLSchema, in caso viene utilizzato solo il nome reale dell'attributo.

B.3.2 Eventuali modifiche

Se si volessero aggiungere funzionalità alla classe "LanguageConstraint" vengono coinvolte le funzioni da "generateCustomType()" seguendo l'immagine dell'architettura B.5. Per gestire un nuovo parametro della classe "LanguageConstraint" va aggiunto il costrutto "getElementsByTagName()" relativo e quindi va aggiunto il procedimento di risoluzione del nuovo parametro aggiungendo la funzione relativa.

B.4 Tool di traduzione nel linguaggio di basso livello della NSF

Questo tool permette la generazione di un file di testo contenente la policy espressa nel linguaggio di basso livello della NSF. Il file di output viene generato in funzione delle capacità di sicurezza presenti nella policy di sicurezza espressa nel linguaggio generico della NSF. Nell'architettura dell'ambito della tesi, questo tool viene utilizzato ogni volta che si vuole tradurre una policy per la data NSF. Il tool sviluppa l'output seguendo il workflow in Figura B.7.

I passaggi principali del funzionamento di questo tool sono i seguenti:

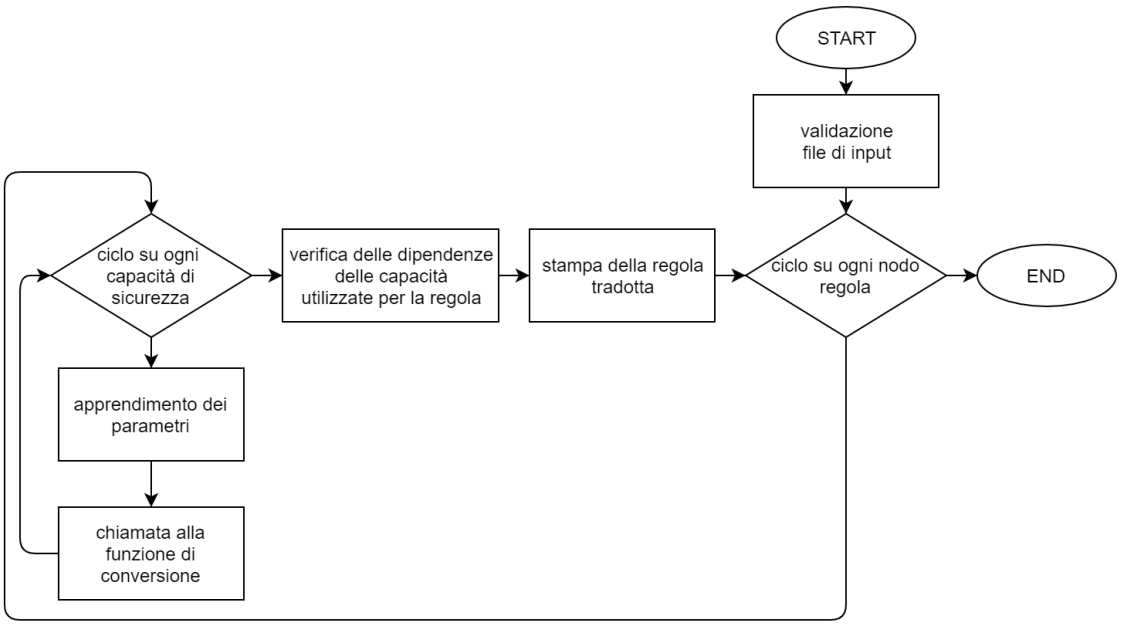

Figura B.7. Workflow del Tool di traduzione nel linguaggio di basso livello della NSF.

- validazione dei file in input;
- ciclo per ogni regola nel file contenente la policy;
- ciclo per ogni capacità di sicurezza nella regola che si sta considerando;
- dal file di assegnazione delle capacità viene preso l'elemento con tag chiave "adapter" relativo alla capacità di sicurezza attualmente considerata;
- utilizzando i valori contenuti nell'istanza della capacità di sicurezza attualmente considerata ed i valori contenuti nella classe "adapter" relativa, viene generata la traduzione della porzione di regola di sicurezza;
- concluse le capacità di sicurezza relative ad una regola vengono controllate le dipendenze di ogni capacità di sicurezza utilizzate per la regola;
- se la regola risulta corretta viene scritta nel file di output;

B.4.1 Architettura del tool

L'architettura del tool proposto è rappresentata nella Figura B.8, in questa struttura la funzione principale è "translate()" che si occupa della traduzione a livello macroscopico, ma affida la traduzione del dettaglio alla funzione "clauseConverter()", la quale si occupa nel dettaglio di ogni

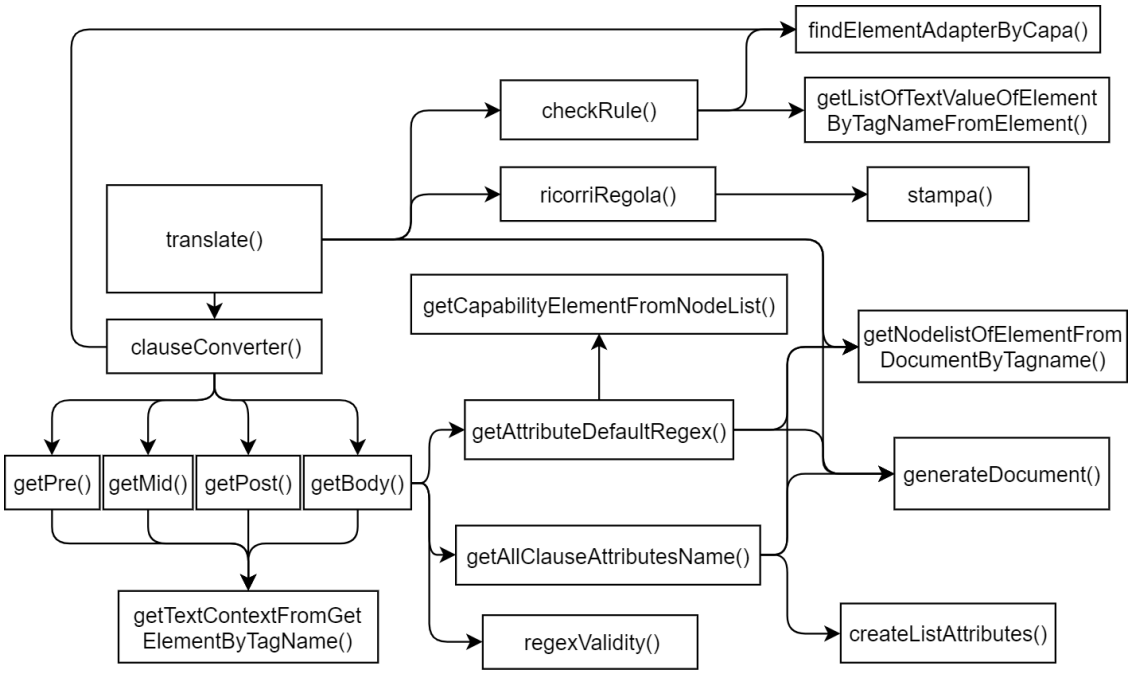

Figura B.8. Architettura del Tool di traduzione nel linguaggio di basso livello della NSF.

elemento che compone la regola. Per effettuare la traduzione la classe "NSFTranslatorAdapter" utilizza le informazioni contenute nella Figura 5.7.

Questa classe definisce le seguenti variabili di istanza:

- `private String xsdLanguage`

  campo che contiene il percorso del file XMLSchema che contiene la rappresentazione del linguaggio della NSF;

- `private String xmlAdapter;`

  campo che contiene il percorso del file XML che contiene l'elenco delle capacità di sicurezza assegnate alla NSF ed eventuali classi "Language" ed "Adapter";

- `private String xmlRule;`

  campo che contiene il percorso del file XML che contiene la policy, espressa nel linguaggio della NSF, da tradurre nel linguaggio di basso livello della NSFs;

- `private String outputName;`

  campo che contiene il percorso ed il nome del file di output dove verrà generata la policy di basso livello della NSF;

- `private String temporaryRule;`

  campo utilizzato per inserire le parti di regola tradotte in formato stringa. Alcuni parametri passati alla funzione translate()" agiscono direttamente su questo campo. Finite le capacità di sicurezza relative a questa regola questo campo verrà utilizzato per la verifica della correttezza della regola stessa ed in caso di risultato positivo il contenuto verrà scritto nel file di output;

- `private String temporaryCapa;`

campo che contiene il nome della capacità di sicurezza attualmente gestita;

- `private List<String> temporaryListCapaOfRule;`

campo che contiene l'elenco dei nomi delle capacità di sicurezza incontrate durante la traduzione delle regola attuale;

- `private NodeList adapterNodes;`

campo che contiene la lista dei nodi delle capacità di sicurezza che contengono le informazioni riguardo le classi "Adapter" per ogni capacità di sicurezza;

- `private Element myCapaAdapter;`

campo che contiene l'istanza della classe "Adapter" relativo alla capacità di sicurezza attualmente analizzata;

- `private String temporaryCapaAndAttributes;`

campo stringa che contiene il nome della capacità di sicurezza attualmente considerata ed una sequenza di nomi di attributi ciascuno seguito dal valore di tale attributo. Ogni componente di questo campo è separato dal successivo con il carattere spazio "␣". Ad esempio: "nomeCapacità␣attributo1␣valore1␣attributo2␣valore2".

Questa classe è composta dalle seguenti funzioni:

- `public NSFTranslatorAdapter()`

costruttore che permette di istanziare un oggetto corrispondente alla classe "NSFTranslatorAdapter";

- `public boolean translate(String xsd, String xmlAdapter, String xmlRule, String outputName, String startString, String endString, String forced, boolean scr, boolean ecr)`

questa funzione si occupa di leggere la policy in input e gestirne ogni elemento utilizzando l'istanza della classe "adapter" corretta. Si occupa della scrittura del file di output ogni volta che viene tradotta una regola utilizzando i parametri di personalizzazione che sono stati passati al momento della chiamata di tale funzione, inoltre chiama la funzione apposita per la verifica della correttezza di ogni regola. L'analisi della policy contenuta nel file di input si basa sulla definizione delle capacità di sicurezza mediante il tag "securityCapability" definito dal tool che genera il linguaggio della NSF.

- `private static Document generateDocument(String path)`

questa funzione genera un oggetto "Document" in base al percorso specificato nella variabile "path";

- `private static NodeList getNodelistOfElementFromDocumentByTagname (Document d, String tagname)`

questa funzione ritorna la lista dei nodi contenenti gli elementi con quel determinato "tagname" appartenenti al documento "d";

- `private void ricorriRegola(Element e)`

questa funzione ricorsiva esplora un elemento fino all'elemento più interno. Ad ogni iterazione della ricorsione viene chiamata la funzione "stampa()";

- `private void stampa(Element e)`

questa funzione utilizza una variabile della classe nella quale può scrivere. La funzione riceve un elemento, a seconda di che elemento può compiere delle azioni:

- se l'elemento ricevuto è un elemento contenente il nome della capacità che si sta considerando allora inserisce nella variabile di riferimento il nome di tale capacità;
- se l'elemento ricevuto rappresenta un attributo allora concatena il nome dell'attributo ed il suo valore nella variabile di riferimento;
- se l'elemento ricevuto non è di una delle due categorie precedenti allora la funzione non esegue nessuna azione;

- `private boolean checkRule()`

questa funzione verifica se sono soddisfatte le regole delle dipendenze contenute nell'istanza di ogni componente della regola. Per ottenere tale risultato la funzione utilizza le informazioni contenute nell'istanza "adapter" di ogni capacità di sicurezza che compone la regola. In particolare considera le istanze degli elementi "dependency". Per la verifica delle dipendenze sono utilizzate delle strutture contenenti le capacità utilizzate dalla regola e la regola già tradotta ma non ancora validata;

- `private Element findElementAdapterByCapa ()`

questa funzione cerca un elemento in base al nome della capacità di sicurezza che si sta considerando in questo momento, il nome è contenuto in una variabile della classe. Questa funzione ritorna null se non dovesse esserci l'istanza della classe "adapter" relativa.

- `private List<String> getListOfTextValueOfElementByTagNameFromElement (Element e, String tagName)`

questa funzione, dato un elemento "e", trova l'elemento di tag "tagName" e genera una lista di stringhe contenente tutti i valori testuali appartenenti a tali elementi;

- `private String clauseConverter()`

questa funzione si occupa della conversione del singolo elemento di una regola. Per la gestione del singolo elemento viene proposta la scomposizione di tale elemento in quattro parti:

- pre : parte che contiene la porzione di stringa che indica qual'è il comando che si sta utilizzando;
- mid : parte che contiene il concatenatore tra il pre ed il body;
- body : parte che contiene la porzione di stringa che concretamente indica l'eventuale valore del comando;
- post : parte che contiene il concatenatore tra l'attuale porzione di regola e la successiva;

Ognuna delle quali parti viene sviluppata da una funzione specifica. Questa funzione si occupa di ritornare una stringa dove vengono concatenate le 4 parti appena descritte;

- `private String getPre()`

questa funzione considera il nome della capacità di sicurezza relativo alla porzione di regola attuale e la classe "adapter" relativa a tale capacità di sicurezza. Dalle condizioni della porzione di regola viene riconosciuto quale comando, definito nella classe "adapter", utilizzare;

- `private String getMid()`

questa funzione cerca l'elemento "adapter" relativo alla concatenazione interna della porzione della regola; se nella classe "adapter" non è stato esplicitato allora si considera uno spazio come concatenatore prestabilito;

- `private String getBody()`

questa funzione considera gli attributi della capacità di sicurezza ed i loro valori relativi alla porzione di regola attuale e la classe "adapter" relativa alla capacità di sicurezza considerata. Valutando gli attributi della porzione di regola vengono riconosciuti i casi di utilizzo di eventuali concatenatori da applicare, definiti nella classe "adapter". Questa funzione si occupa anche di valutare se sono stati inseriti valori corretti, infatti può valutare tali valori applicando eventuali espressioni regolari o restrizioni numeriche esplicite;

- `private String getPost()`

questa funzione cerca l'elemento "adapter" relativo alla concatenazione tra l'attuale porzione della regola e la successiva; se nella classe "adapter" non è stato esplicitato allora si considera uno spazio come concatenatore prestabilito;

- `private String getTextContextFromGetElementByTagName(Element e, String s)`

questa funzione ritorna il valore testuale dell'elemento di nome equivalente al contenuto del parametro "s";

- `private List<String> getAllClauseAttributesName()`

questa funzione si occupa di generare una lista contenente tutti i nomi degli attributi della capacità di sicurezza che si sta attualmente considerando. La lista viene riempita utilizzando la funzione "createListAttributes()";

- `private void createListAttributes (NodeList elementNL, List<String> ls, Element e)`

questa funzione crea, in modo ricorsivo, una lista che contiene gli attributi appartenenti all'elemento "e", se l'elemento è un tipo complesso allora viene richiamata la stessa funzione fino ad ottenere tutti i nomi dei parametri;

- `private String getAttributeDefaultRegex(String attributeName)`

questa funzione cerca se la capacità di sicurezza che si sta considerando al momento, nell'attributo il cui nome corrisponde al valore del parametro "attributeName", possiede una espressione regolare preimpostata;

- `private Element getCapabilityElementFromNodeList(String capa, NodeList nl)`

questa funzione ritorna, se presente, l'elemento il cui attributo "name" ha lo stesso valore del contenuto del parametro "capa";

- `private boolean regexValidity(String value, String regex)`

questa funzione verifica la validità del valore contenuto nel parametro "value" rispetto all'espressione regolare contenuta nel parametro "regex".

B.4.2 Eventuali modifiche

Se si volessero aggiungere funzionalità alla classe "Adapter" vengono coinvolte le funzioni da "clauseConverter()" e da "checkRule()" seguendo l'immagine dell'architettura B.8. Per gestire un nuovo parametro della classe "Adapter" va aggiunto il costrutto "getElementsByTagName()" relativo e quindi va aggiunto il procedimento di risoluzione del nuovo parametro aggiungendo la funzione relativa o le porzioni di codice relativi al nuovo parametro. La funzione "clauseConverter()" può essere modificata per poter riconoscere eventuali parametri speciali ricevuti utilizzando la classe "Adapter". Ad esempio, in questa classe, è stato necessario utilizzare la seguente porzione di codice per permettere l'inserimento del carattere speciale "
n" come concatenatore della traduzione interno alla capacità stessa o tra la capacità e quella successiva;

```
post = getPost();
if(post.equals("\\n")) {
       post = System.lineSeparator();
}
```

B.5 Tool di validazione

Questo tool permette la validazione di un file XML rispetto ad un file XMLSchema entrambi passati come parametri al tool. Nell'architettura dell'ambito della tesi questo tool viene utilizzato ogni qualvolta si debba istanziare un file XML necessario ad uno dei tool proposti.

L'architettura di questo tool è composta da una singola classe.

Questa classe è composta dalle seguenti funzioni:

- `public Validation()`

costruttore che permette di istanziare un oggetto corrispondente alla classe "Validation";

- `public boolean validate(String xsd, String xml)`

questa funzione permette di validare il file definito nel percorso contenuto nel parametro "xml" rispetto al file definito nel percorso contenuto nel parametro "xsd".

Bibliografia

[1] International Organization for Standardization project, https://www.iso.org/about-us.html

[2] Object Management Group, Unified Modeling Language project, https://www.omg.org/spec/UML/

[3] Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C, November 2008, https://www.w3.org/TR/xml/

[4] Information processing – Text and office systems – Standard Generalized Markup Language (SGML), 2008, https://www.iso.org/standard/16387.html

[5] World Wide Web Consortium, Extensible Markup Language essential, https://www.w3.org/standards/xml/core

[6] World Wide Web Consortium project, https://www.w3.org/

[7] World Wide Web Consortium, XML Schema Definition Language (XSD) 1.1 Part 1: Structures, April 2012, https://www.w3.org/TR/xmlschema11-1/

[8] World Wide Web Consortium, XML Schema: Formal Description, September 2001, https://www.w3.org/TR/2001/WD-xmlschema-formal-20010925/

[9] M.Bjorklund, "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC-6020, October 2010, DOI 10.17487/RFC6020

[10] R.Enns, M.Bjorklund, M.Bjorklund, J.Schoenwaelder, A.Bierman, "Network Configuration Protocol (NETCONF)", RFC-6241, June 2011, DOI 10.17487/RFC6241

[11] M.Bjorklund, "The YANG 1.1 Data Modeling Language", RFC-7950, August 2016, DOI 10.17487/RFC7950

[12] E.Gamma, R.Helm, R.Johnson, J.Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1994, ISBN-10: 0-201-63361-2 ISBN-13: 978-0-201-63361-0

[13] C.Alexander, S.Ishikawa, M.Silverstein, "A Pattern Language", Oxford University Press, 1977, ISBN 0-19-501919-9

[14] William J.Brown, Raphael C.Malveau, Hays W.McCormick III, Thomas J.Mowbray, "AntiPatterns Refactoring Software, Architectures, and Projects in Crisis" John Wiley & Sons, 1998, ISBN 0-471-19713-0

[15] A.Pras, J.Schoenwaelder, "On the Difference between Information Models and Data Models", RFC-3444, January 2003, DOI 10.17487/RFC3444

[16] Pyang project, github, https://github.com/mbj4668/pyang

[17] Libyang project, github, https://github.com/CESNET/libyang

[18] Ydk project, github, https://github.com/CiscoDevNet/ydk-py

[19] Model Driven Architecture (MDA), Object Management Group (OMG), https://www.omg.org/mda/

[20] MDA Guide Version 1.0.1, Object Management Group (OMG), June, 2003 https://www.omg.org/news/meetings/workshops/UML_2003_Manual/00-2_MDA_Guide_v1.0.1.pdf

[21] Rachel A.Pottinger, "Merging Models Based on Given Correspondences", 2003 http://www.vldb.org/archives/website/2003/papers/S26P01.pdf

[22] Object Management Group, "OMG Meta Object Facility (MOF) Core Specification" Version 2.5.1, November 2003 https://www.omg.org/spec/MOF/2.5.1/PDF

[23] A.Kleppe, J.Warmer, W.Bast, "MDA Explained: The Model Driven Architecture: Practice and Promise", Addison-Wesley, April, 2003, ISBN-13: 978-0321194428 ISBN-10: 032119442X

[24] M.Fowler, R.Parsons, "Domain-Specific Languages", Addison-Wesley, 2011.

[25] K.Balasubramanian, A.Gokhale, G.Karsai, J.Sztipanovits, S.Neema, "Developing Applications Using Model-Driven Design Environments", IEEE Computer, Vol. 39, No. 2 February 2006, pp. 33-40, DOI 10.1109/MC.2006.54

[26] B.Selic, "The pragmatics of model-driven development", IEEE Software , Vol. 20, No. 5 Sept.-Oct. 2003, pp. 19-25, DOI 10.1109/MS.2003.1231146

[27] Jean-Marie Favre, "Towards a Basic Theory to Model Model Driven Engineering", October 2011, https://www.researchgate.net/publication/239016408_Towards_a_Basic_Theory_to_Model_Model_Driven_Engineering

[28] S.Kent, "Model Driven Engineering", International Conference on Integrated Formal Methods, Volume 2335, April 2002, DOI 10.1007/3-540-47884-1_16

[29] J.Bézivin, "On the unification power of models", Software & Systems Modeling, Vol. 4, No. 2, May 2005, DOI 10.1007/s10270-005-0079-0

[30] T.Hettel, M. Lawley, K. Raymond "Model Synchronisation: Definitions for Round-Trip Engineering", Theory and Practice of Model Transformations, July 2008, pp. 31-45, DOI 10.1007/978-3-540-69927-9_3

[31] E.Seidewitz, "What models mean", IEEE Software, Vol. 20, No. 5, September 2003, pp. 26-32, DOI 10.1109/MS.2003.1231147

[32] Robert P.Goldberg, "Survey of virtual machine research", IEEE Computer, Vol. 7, No. 6 June 1974, pp. 34-45, DOI 10.1109/MC.1974.6323581

[33] Rashid Mijumbi, Joan Serrat, J.Gorricho, Niels Bouten, Filip De Turck, Raouf Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges", IEEE Communications Surveys & Tutorials, Vol. 18, No. 1 September 2015, pp. 236-262, DOI 10.1109/COMST.2015.2477041

[34] European Telecommunications Standards Institute, "Network Functions Virtualisation (NFV); Virtualization Requirements", October 2013, https://www.etsi.org/deliver/etsi_gs/NFV/001_099/004/01.01.01_60/gs_NFV004v010101p.pdf

[35] European Telecommunications Standards Institute, "Network Functions Virtualisation (NFV); Virtual Network Function Architecture", December 2014, https://www.etsi.org/deliver/etsi_gs/NFV-SWA/001_099/001/01.01.01_60/gs_NFV-SWA001v010101p.pdf

[36] European Telecommunications Standards Institute, "Network Functions Virtualisation (NFV); Resiliency Requirements", January 2015, https://www.etsi.org/deliver/etsi_gs/NFV-REL/001_099/001/01.01.01_60/gs_NFV-REL001v010101p.pdf

[37] J.Halpern, C.Pignataro, "Service Function Chaining (SFC) Architecture", RFC-7665, November 2015, DOI 10.17487/RFC7665

[38] Diego Kreutz, Fernando M. V. Ramos, Paulo Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, Steve Uhlig, "Software-Defined Networking: A Comprehensive Survey", Proceedings of the IEEE, Vol. 103, No. 1 January 2015, pp. 14-76, DOI 10.1109/JPROC.2014.2371999

[39] The Open Networking Foundation (ONF), Online, https://www.opennetworking.org

[40] N.McKeown, T.Anderson, H.Balakrishnan, G.Parulkar, L.Peterson, J.Rexford, S.Shenker, and J.Turner, "OpenFlow: Enabling innovation in campus Networks", SIGCOMM Computer Communication Review, Vol. 38, No. 2, pp. 69-74, 2008.

[41] European Telecommunications Standards Institute, "Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV", December 2014, https://www.etsi.org/deliver/etsi_gs/NFV/001_099/003/01.02.01_60/gs_nfv003v010201p.pdf

[42] S.Hares, J.Strassner, D.Lopez, L.Xia, H.Birkholz, "Interface to Network Security Functions (I2NSF) Terminology" July 2019, https://datatracker.ietf.org/doc/draft-ietf-i2nsf-terminology/

[43] S.Hares, D.Lopez, M.Zarny, C.Jacquenet, R.Kumar, J.Jeong, "Interface to Network Security Functions (I2NSF): Problem Statement and Use Cases", RFC-8192, July 2017, DOI 10.17487/RFC8192

[44] A.Pastor, D.Lopez, K.Wang, X.Zhuang, M.Qi, M.Zarny, S.Majee, N.Leymann, L.Dunbar, M.Georgiades, "Use Cases and Requirements for an Interface to Network Security Functions", June 2015, https://tools.ietf.org/html/draft-pastor-i2nsf-merged-use-cases-00

[45] Interface to Network Security Functions (i2nsf), Online, https://datatracker.ietf.org/wg/i2nsf/about/

[46] E.Messmer, "Gartner: Cloud-based security as a service set to take off", October 2013 https://www.networkworld.com/article/2171424/gartner--cloud-based-security-as-a-service-set-to-take-off.html

[47] D.Lopez, E.Lopez, L.Dunbar, J.Strassner, R.Kumar, "Framework for Interface to Network Security Functions", RFC-8329, February 2018, DOI 10.17487/RFC8329

[48] J.Jeong, C.Chung, S.Hares, L.Xia, H.Birkholz, "I2NSF NSF Monitoring YANG Data Model", July 2019 https://www.ietf.org/id/draft-ietf-i2nsf-nsf-monitoring-data-model-01.txt

[49] J.Jeong, E.Kim, T.Ahn, R.Kumar, S.Hares, "I2NSF Consumer-Facing Interface YANG Data Model", July 2019, https://www.ietf.org/id/draft-ietf-i2nsf-consumer-facing-interface-dm-06.txt

[50] J.Kim, J.Jeong, J.Park, S.Hares, Q.Lin, "I2NSF Network Security Function-Facing Interface YANG Data Model", July 2019 https://www.ietf.org/id/draft-ietf-i2nsf-nsf-facing-interface-dm-07.txt

[51] L.Xia, J.Strassner, C.Basile, D.Lopez, "Information Model of NSFs Capabilities", April 2019, https://datatracker.ietf.org/doc/draft-ietf-i2nsf-capability/

[52] J.Strassner, J.Halpern, J.Coleman, "Generic Policy Information Model for Simplified Use of Policy Abstractions (SUPA)", March 2016, https://tools.ietf.org/html/draft-strassner-supa-generic-policy-info-model-05

[53] P.Mohagheghi, J.Aagedal, "Evaluating Quality in Model-Driven Engineering", Online document, https://www.omg.org/ocsmp/MiSE2007-QualityMDE.pdf

[54] S.Hyun, J.Jeong, T.Roh, S.Wi, J.Park, "I2NSF Registration Interface YANG Data Model", July 2019, https://datatracker.ietf.org/doc/draft-ietf-i2nsf-registration-interface-dm/