POLITECNICO DI MILANO
Corso di Laurea Magistrale in Ingegneria Informatica
Scuola di Ingegneria Industriale e dell'Informazione
Dipartimento di Elettronica, Informatica e Bioingegneria

GENERAZIONE AUTOMATICA DI SPIEGAZIONI IN UN SISTEMA PER IL CONTROLLO DELL'ACCESSO AI DATI BASATO SU OWL

Relatore: Prof. Marco Colombetti
Correlatore: Ing. Fabio Marfia

Tesi di Laurea di: Matteo Tornielli

Anno Accademico 2014-2015

Ringraziamenti

I miei primi ringraziamenti vanno a coloro che hanno reso questa tesi possibile: il mio relatore il professor Marco Colombetti, e il dottor Fabio Marfia, che mi hanno seguito in questi ultimi mesi del mio percorso offrendomi la loro conoscenza e dimostrandomi una grandissima disponibilità, lavorare con voi è stato un vero piacere.
Un enorme grazie va alla mia famiglia: a mio papà Federico per aver finanziato economicamente i miei studi e per avermi sempre dato i migliori consigli che potessi ricevere, tecnici e di vita, saresti un grande ingegnere, e questo vuole essere un sincero complimento! A mia mamma Claudia, la migliore ascoltatrice che abbia mai incontrato, per essere sempre pronta ad sentire le mie difficoltà, consigliarmi e contagiarmi con la sua tranquillità e serenità. A mia sorella Marta, la mia prima compagna di giochi e marachelle, per aver sempre preso le mie difese e per aver sempre creduto in me.
Un ringraziamento speciale va alla mia ragazza Deborah, per i suoi consigli pratici sulla tesi e sul mondo del lavoro. Per essermi stata vicina nei momenti difficili, soprattutto quando ancora ci conoscevamo da poco tempo. Per avermi sempre dimostrato affetto, stima e fiducia incondizionati. Sei una persona bella e unica, averti al mio fianco rende ogni difficoltà più facile da superare.
Ringrazio anche i miei quattro nonni: Eliseo, Elena, Gianfranco e Mirella per tutti gli insegnamenti e i valori che mi hanno tramandato, e per la loro supervisione ora che non ci sono più.
Vorrei poi ringraziare i miei compagni di studi, in ordine alfabetico: Alex, Andrea, Fabio, Filippo e Sergio, il vostro supporto, la vostra compagnia e soprattutto la vostra amicizia hanno reso questi anni belli e indimenticabili.
Un ultimo grosso grazie va a tutti i miei parenti e tutti gli amici che oggi hanno deciso di esserci, che hanno deciso di dedicare un po' del loro tempo per festeggiare questo traguardo veramente molto importante per me. Grazie a tutti davvero di cuore!

Indice

Sommario 1

1 Introduzione 3

2 Stato dell'arte 5
2.1 XACML 5
2.1.1 L'architettura 5
2.1.2 Le policy 7
2.2 PoSecCo 9
2.3 Altri Policy-Access Framework 11

3 Requisiti e specifiche tecniche 13
3.1 Analisi dei requisiti 13
3.1.1 Stakeholders 13
3.1.2 Entità 15
3.1.1 Requisiti 16
3.2 Architettura 17
3.2.1 Componenti principali 17
3.2.2 Flussi di esecuzione 21

4 Framework 25
4.1 Traduzione delle policy 25
4.1.1 Il file delle policy XACML 26
4.1.2 Modelli di rule 26
4.1.3 Composizione di rule 35
4.1.4 Algoritmo di traduzione 38
4.2 Explanation 41
4.2.1 Casi iniziali 41
4.2.2 Elaborazione 42
4.2.3 Revisione 48

5 Performance 51
5.1 Valutazione delle performance 51
5.1.1 Traduzione di policy 52
5.1.2 Spiegazioni di accesso 53

6 Conclusioni **57**
6.1 Sviluppi futuri 58

Bibliografia **59**

Indice delle figure

Capitolo 2
Figura 2.1: XACML data-flow diagram 7
Figura 2.2: esempio di rule 8
Figure 2.3: meta-modello PoSecCo IT level Security 9
Figura 2.4 Classi di policy di PoSecCo IT 11

Capitolo 3
Figura 3.1 Diagramma Use-case 14
Figura 3.2: Diagramma delle entità del sistema 15
Figura 3.3: l'architettura del framework 18
Figura 3.4: Struttura del PDP 19
Figura 3.5: struttura del PAP 20
Figura 3.6: struttura del Context Handler 21
Figura 3.7: sequence diagram per richiesta di autorizzazione 22
Figura 3.8: sequence diagram per richiesta di spiegazione 23
Figura 3.9: sequence diagram per la modifica delle politiche 24

Capitolo 4
Figura 4.1: Esempio XACML di rule 1 (IBAC) 28
Figura 4.2: Esempio XACML di rule 2 (RBAC) 20
Figura 4.3: Esempio XACML di rule 3 (ABAC semplice) 31
Figura 4.4: Esempio XACML di rule 4 (ABAC composta) 33
Figure 4.5: la relazione triangolo della rule 5 35
Figura 4.6: Esempio XACML congiunzione OR 36
Figura 4.7: Esempio XACML congiunzione AND 37
Figura 4.8: Algoritmo traduzione delle policy 39
Figura 4.9: Class Diagram del programma Java di traduzione 40
Figura 4.10: Casi delle explanations 42
Figura 4.11: Algoritmo elaborazione Explanations 43
Figura 4.12: explanations per rule di tipo 1 44
Figura 4.13: explanations per rule di tipo 2 45
Figura 4.14: explanations per rule di tipo 3 46
Figura 4.15: explanations per rule di tipo 4 47
Figura 4.16: explanations per rule di tipo 5 48

Capitolo 5

Figura 5.1: Tempo di creaz. policy con numero crescente di individui 52
Figura 5.2: Tempo di creaz. policy con numero crescente di policy . . 53
Figura 5.3: Tempo per Explanation con numero crescente di individui 54
Figura 5.4: Tempo per Explanation con numero crescente di policy . . 55

Sommario

Nel corso dell'ultimo decennio le policy sono diventate una soluzione molto diffusa per regolare l'accesso ai dati e gestirne la riservatezza e la sicurezza per grandi sistemi multiagente. Il massiccio aumento dei dati, soprattutto nel mondo Web, ha richiesto la definizione di regole sempre più complesse per l'accesso alle risorse, e le politiche di accesso sembrano fornire la soluzione migliore a questo problema. Un framework basato sulle politiche di accesso introduce molteplici vantaggi nella gestione dei controlli di accesso come la modifica delle regole a run-time, e porta benefici in termini di espressività, scalabilità, efficacia, flessibilità, estensibilità, sensibilità al contesto. Inoltre, un sistema di questo tipo sarebbe anche in grado di fornire all'utente delle spiegazioni per le decisioni di accesso, in una realtà composta da regole sempre più complesse sarebbe utile all'utente, infatti, sapere per quale motivo la sua richiesta di accesso non è stata approvata, al fine di rendere, se possibile, le sue credenziali di accesso idonee alla richiesta effettuata.
Lo scopo di questa tesi è quindi quello di definire un framework unificato per esprimere, far rispettare regole di accesso e fornire delle spiegazioni user-friendly per le decisioni prese, combinando gli standard dell'architettura e del linguaggio XACML con i benefici delle ontologie OWL e delle tecnologie di reasoning. Sono stati definiti una completa architettura centralizzata e i prototipi di alcuni dei suoi moduli, oltre ad algoritmi completi per tradurre automaticamente politiche di tipo XACML nei corrispondenti assiomi OWL e per generare e fornire all'utente spiegazioni dei ragionamenti effettuati.

Abstract

Policies have become nowadays a very popular solution for regulating access to data and manage confidentiality and security for large multi-agent systems. The massive increase of data, especially on the Web, requires the formulation of more complex rules for accessing resources, and policies seem to provide the best solution to such issue. A framework based on access policies introduces several advantages in the management of access control such as: changing the rules at run-time, generating benefits in terms of scalability, expressiveness, flexibility, effectiveness, extensibility and context-sensitivity. In addition, a system of such type would also be able to provide explanations for access decisions. In fact, it would be useful for a user to be aware of the causes of a request rejection, in order to correct his/her own credentials for suiting an appropriate profile for the request he/she made, if possible.
Therefore, the purpose of this thesis is to define a unified framework for expressing and enforcing policies that is able to provide a user-friendly explanation for an obtained decision. Such goal is reached by combining the standard XACML architecture with the advantages of OWL ontologies and reasoning techniques. We define a complete and centralized architecture, and some prototypes of its modules, in addition to complete algorithms for translating policies from XACML to the corresponding OWL axioms and generating automatic explanations for obtained decisions.

Capitolo 1

Introduzione

Al giorno d'oggi numerosi sistemi multiagenti fanno uso di Policy per regolare l'accesso ai dati e gestire privacy e sicurezza. L'enorme incremento dei dati, specialmente sul web, richiede la definizione di sempre più complesse regole di accesso, e le politiche di accesso sembrano proporre la soluzione migliore a questo tipo di problema. Come negli anni '60, quando alcuni ingeneri decisero di creare meccanismi logici di stoccaggio e gestione dei dati indipendenti dalle applicazioni: le prime Basi di Dati, ora comincia a diffondersi l'idea di creare sistemi di Gestione delle Politiche di accesso indipendenti dalle applicazioni che ne fanno uso.
Le politiche di accesso non sono altro che proposizioni formali secondo le quali è possibile specificare se un agente è o non è autorizzato a campiere una determinate azioni.
Un approccio basato sulle politiche introduce importanti benefici al controllo sugli accessi: in primo luogo supporta il cambio dinamico di politiche a run-time, sopraffacendo l'uso obsoleto di definizioni di configurazioni statiche fatte prima dell'avvio del sistema. Le politiche possono essere definite indipendentemente dal dominio dell'applicazione, le regole di accesso possono essere cambiate senza dover modificare il dominio e sono anche agente-indipendenti, non è necessario che gli agenti sappiano della loro esistenza.
Un altro grossissimo vantaggio di un approccio basato sulle politiche di accesso è il loro elevatissimo potere espressivo: esse consentono il controllo di accesso ai dati non basandosi solo sull'identità o sul ruolo del soggetto, ma anche sui suoi attributi e sulle relazioni che collegano il soggetto ad altri agenti o alle risorse stesse del dominio dell'applicazione.
In conclusione un Framework per il Controllo di Accessi basato su politiche porterebbe vantaggi enormi in termini di efficienza, flessibilità, estensibilità, scalabilità, riusabilità e sensibilità al contesto.
L'uso sempre maggiore di politiche per gestire i controlli di accesso per scenari di complessi sistemi distribuiti ha favorito lo sviluppo di framework centralizzati ma composti da differenti moduli. Questa scelta oltre a facilitare funzionalità come modifica e gestione delle politiche a run-time, porta anche benefici in

termini di concorrenzialità, controlli di consistenza e ottimizzazione del processo di query. Assicura inoltre una maggiore scalabilità e fornisce la possibilità di produrre spiegazioni a proposito delle decisioni di accesso prese.
Negli ultimi anni molti framework basati su XACML sono stati sviluppati, alcuni di loro facendo uso della Description Logic (DL), ma non c'è traccia nell'attuale letteratura né di un framework che combini XACML con OWL né di un sistema in grado di generare spiegazioni automatiche. Fino ad ora infatti, anche in grossi e complessi sistemi di autorizzazione, agli utenti viene concesso o negato l'accesso alle risorse senza alcuna possibilità di sapere in maniere chiara quali siano le più importanti cause che hanno portato a una determinata risposta di sistema.
Lo scopo di questa tesi è perciò quello colmare questo vuoto e definire un framework unificato per esprimere e far rispettare le politiche, combinando l'architettura e il linguaggio di politiche XACML con i benefici delle ontologie OWL e delle tecnologie di reasoning, come destritto nell'articolo "An OWL-Based XACML Policy Framework" [14]. In grado inoltre di generare automaticamente delle spiegazioni di accesso e restituirle in modo chiaro e conciso agli utenti.
Il documento è organizzato come segue. Il corrente stato dell'arte riguardo al controllo sugli accessi basato su politiche è presentato nel Capitolo 2, includendo anche una descrizione dettagliata dello standard XACML, sui cui è basato il nostro sistema. I requisiti funzionali e i dettagli architetturali del sistema proposto sono descritti nel Capitolo 3. Una descrizione completa di come il progetto è stato implementato è fornita nel Capitolo 4. Questo include la descrizione di un dominio di esempio, la spiegazione di come le politiche XACML siano convertite in assiomi di DL e di di come il sistema generi automaticamente spiegazioni per le decisioni prese. Le performance del sistema, e gli esperimenti condotti per misusarle, sono analizzate nel Capitolo 5. Infine sono presentate alcune conclusioni nel Capitolo 6, insieme a descrizioni di possibili sviluppi futuri.

Capitolo 2

Stato dell'arte

Le policy sono ormai una soluzione molto diffusa per gestire l'accesso alle risorse al fine di garantirne la sicurezza, esse infatti permettono o negano agli utenti l'accesso ai dati seguendo specifiche regole. Negli anni sono stati sviluppati numerosi framework che cercano di offrire una soluzione al problemi che derivano dai controlli di accesso ai dati. Ciò che li accomuna è il tentativo di sviluppare un sistema centralizzato in grado di gestire diverse funzioni, come: la modifica delle politiche di accesso, l'integrazione delle stesse e il processo decisionale.
I framework possono essere suddivisi in tre categorie: quelli bastati su XML, quelli che si basano su linguaggi logici tipo Prolog e Datalog e quelli che fanno uso delle Description Logics (DL).

2.1 XACML

La maggior parte dei framework usati per il controllo delle politiche di accesso sono basati su linguaggi di markup come XML. Il più importante di questi è XACML, è proprio per questo che il lavoro svolto in questa tesi parte da politiche scritte con questo linguaggio.

2.1.1 L'architettura

eXtensible Access Control Markup Language (XACML) è uno standard pubblicato da OASIS (Organization for the Advancement of Structured Information Standards) che definisce un linguaggio per la definizione di politiche di controllo degli accessi in formato XML, e illustra come valutare le richieste di autorizzazione. Rappresenta lo standard attuale tra i linguaggi di controllo di accesso tramite politiche, permette agli amministratori, attraverso l'implementazione di un layer di sicurezza di raggruppare e gestire le diverse regole.

Il modello XACML, come descritto nel suo documento di specificazione [1] è formato dai seguenti componenti architetturali:

- *Policy Enforcement Point (PEP):* il componente del sistema che esegue il controllo di accesso, inoltrando richieste di decisione al PDP e facendo rispettare la il risultato di autorizzazione ottenuto.
- *Policy Decision Point (PDP):* il componente del sistema che valuta l'applicabilità di una politica e restituisce la decisione sull'autorizzazione.
- *Policy Information Point (PIP):* il componente del sistema che fornisce i valori degli attributi degli attori coinvolti nella richiesta di accesso.
- *Policy Administration Point (PAP):* il componente del sistema che crea una o un insieme di politiche.
- *Context Handler:* il componente del sistema che coordina la comunicazione fra PEP, PDP e PIP. Converte richieste di decisioni dal formato nativo alla forma canonica di XACML, e decisioni di autorizzazione dalla forma canonica di XACML al formato nativo della risposta.

Il data-flow model della Figura 2.1 mostra tutti gli attori coinvolti nel framework basato su XACML. Il PAP permette di inserire e modificare una o un insieme di politiche scritte in XACML e le rende poi disponibili per il PDP. I restanti tre attori invece coordinati dal Context Handler sono quelli coinvolti nel processo decisionale. La richiesta di un'autorizzazione arriva al PEP che la inoltra al Context Handler, che a sua volta la passa al PDP insieme a eventuali attributi che quest'ultimo gli ha richiesto e che il Context ha collezionato dal PIP. Il PDP ha quindi la richiesta, gli attributi e le politiche è quindi in grado di decidere se concedere o no l'autorizzazione. Una volta che ha stabilito l'esito della richiesta passa il risultato al Context Handler che a suo volta lo inoltra al PEP. Il flusso termina qui, l'ultimo compito del PEP è infatti quello di far rispettare la decisione presa.

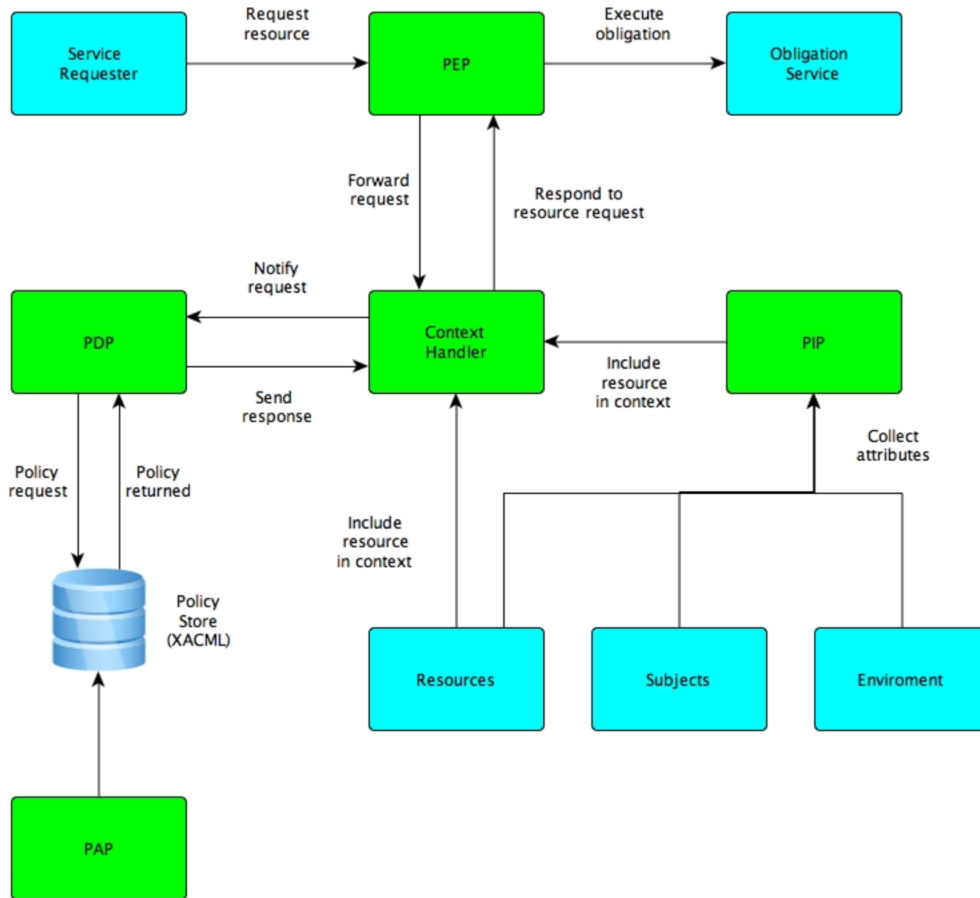

Figura 2.1: XACML data-flow diagram.

2.1.2 Le policy

Dopo l'architettura e il suo funzionamento ci concentriamo ora sugli attori principali di questo linguaggio: le policy. Possiamo identificare tre elementi importanti: **policy set**, **policy** e **rule** (regola) quest'ultima a sua volta è composta da altri elementi. Un policy set è semplicemente un insieme di politiche e una policy è un insieme di rule.
La rule è l'elemento fondamentale, è composta da un campo **effetto** che può assumere due valori *Permit (consenti)* o *Deny (nega)* e un campo target che identifica gli attori coinvolti nella regola. Ogni target è composto da un elenco di **subjects** (soggetti), un elenco di **resources** (risorse) e un elenco di **actions** (azioni). Una regola applica il suo *effetto* sul *target* specifico, cioè permette o nega al soggetto (ai soggetti) di eseguire l'azione sulla risorsa (sulle risorse).

```xml
<Rule RuleId="Rule2" Effect="Permit">
    <Target>
        <Subjects>
            <Subject>
                <SubjectMatch MatchId="urn:polimi:names:dbsp:1:function:ontology-id-equal">
                <AttributeValue DataType="urn:polimi:names:dbsp:1:data-type:ontology-id">medicalConsultant</AttributeValue>
                <SubjectAttributeDesignator AttributeId="urn:polimi:names:dbsp:1:attribute:class" MustBePresent="true"
                    DataType="urn:polimi:names:dbsp:1:data-type:ontology-id"/>
                </SubjectMatch>
            </Subject>
        </Subjects>

        <Resources>
            <Resource>
                <ResourceMatch MatchId="urn:polimi:names:dbsp:1:function:ontology-id-equal">
                <AttributeValue DataType="urn:polimi:names:dbsp:1:data-type:ontology-id">medicalRegulationDocument</AttributeValue>
                <ResourceAttributeDesignator AttributeId="urn:polimi:names:dbsp:1:attribute:class"
                    DataType="urn:polimi:names:dbsp:1:data-type:ontology-id"/>
                </ResourceMatch>
            </Resource>
        </Resources>

        <Actions>
            <Action>
                <ActionMatch MatchId="urn:polimi:names:dbsp:1:function:ontology-id-equal">
                    <AttributeValue DataType="urn:polimi:names:dbsp:1:data-type:ontology-id">write</AttributeValue>
                    <ActionAttributeDesignator DataType="urn:polimi:names:dbsp:1:data-type:ontology-id"
                AttributeId="urn:polimi:names:dbsp:1:attribute:id" />
                </ActionMatch>
            </Action>
        </Actions>
    </Target>
</Rule>
```

Figura 2.2: esempio di rule

Le rule vengono raggruppate in una policy attraverso un algoritmo combinatorio, la stessa cosa avviene per le policy combinate in policy set.
Lo scopo degli algoritmi combinatori è quello di raggruppare policy (rule) che devono essere valutate insieme per stabilire il risultato di una certa richiesta di autorizzazione. In altre parole essi definiscono una procedura per ottenere una decisione di autorizzazione dati i risultati individuali delle singole policy (rule).
Gli algoritmi combinatori sono diversi ma i più importanti sono:

- *Deny-overrides:* se una singola policy o rule, il cui effetto di negare un azione, è verificata, allora l'effetto combinato è quello di negare l'azione qualsiasi sia l'esito delle altre valutazioni.
- *Permit-overrides:* se una singola policy o rule, il cui effetto è di consentire un azione, è verificata, allora l'effetto combinato è quello di consentire l'azione qualsiasi sia l'esito delle altre valutazioni.
- *First-applicable:* l'effetto combinato è lo stesso dell'effetto di valutare il primo elemento policy set, policy o rule della lista il cui target e condizione sia applicabile alla decisione richiesta.
- *Only-one-applicable:* questo algoritmo si può applicare solo alle policy. Esso afferma che una e una sola policy (o policy set) può essere applicato in virtù dei suoi target. Se ce ne è più di uno applicabile, allora il risultato è *indeterminato*, altrimenti se nessuna policy è applicabile allora il risultato è *non applicabile*.

Grazie alla sua completa e ancora molto flessibile architettura XACML ha incontrato un grande successo diventando lo standard per le espressioni di politiche di accesso.

2.2 PoSecCo

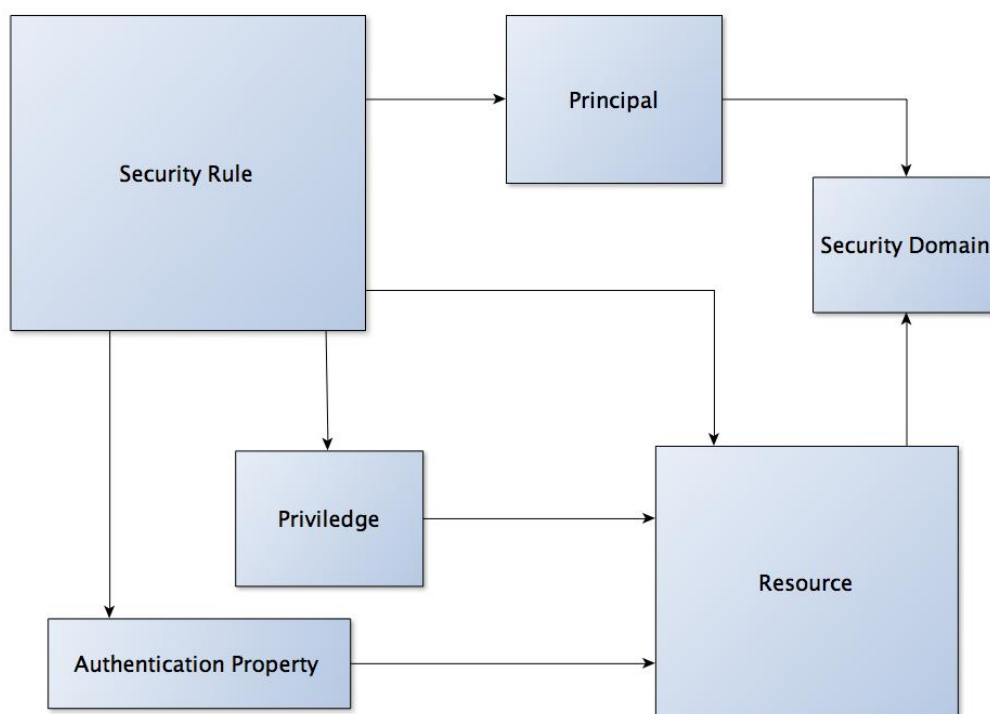

Figure 2.3: meta-modello PoSecCo IT level Security (from [6], page 14)

Un'altra importante famiglia di framework per l'accesso a policy è quella dei DL-Based [5] cioè quelli basati sulle logiche descrittive (Description Logics). L'approccio comune di questi sistemi è quello di definire le politiche con una rappresentazione basata su OWL (Ontology Web Language), e che quindi usa metodi di ragionamento di tipo OWL-based, per svolgere le operazioni sulle regole.
Uno di questi framework è PoSecCo (Policy and Security Configuration Management) [11], un progetto europeo il cui scopo è quello di proporre nuove metodologie e tool per configurare un orientato alla sicurezza. All'interno del framework le politiche possono essere formalizzate in tre differenti livelli di astrazione. Paraboschi et al. [6] propongono una soluzione per la specifica delle politiche chiamata IT policy layer, ed è basata sulla concettualizzazione di una policy chiamata meta-modello IT layer Security.

L'autenticazione e il controllo d'accesso alle politiche passa attraverso la rappresentazione della Figura 2.3. Ogni componente consiste in uno specifico sotto-modello che si focalizza su uno specifico aspetto della sicurezza delle politiche:

- il meta-modello *Principal* è usato per descrivere l'organizzazione delle identità, come gli utenti sono strutturati in gruppi e come i ruoli sono assegnati all'utente.
- Il meta-modello *Security Rule* descrive la struttura dei target e delle policy IT e introduce una tassonomia di autorizzazione e autenticazioni delle regole.
- Il meta-modello *Privilege* è usato per descrivere la struttura dei privilegi che sono specificati per le autorizzazioni di sistema.
- Il meta-modello *Authentication Property* è usato per descrivere la struttura delle proprietà che sono associate alle regole di autenticazione.
- Il meta-modello *Risorsa* descrive la struttura delle risorse che sono associate ai privilegi di autorizzazione e alle proprietà di autenticazione.
- Il meta-modello *Security domain* contiene la entità che denota il concetto di sicurezza di dominio, supportando la realizzazione delle politiche.

Una concettualizzazione di ontologia OWL è direttamente derivata dagli elementi recuperati da ogni singolo sotto-modello. I concetti principali di questa ontologia, usati per rappresentare le politiche sono presentati nella Figura 2.4.
Nella modello concettuale descritto una rule (ITSystemAuthorization) è una quadrupla nella forma $P=<\mu, \pi, \rho, \sigma>$ dove:

- μ è un'identità (ITSingleId), un gruppo (ITGroupId) o un ruolo (ITAuthzRuolo).
- π è un privilegio (ITPrivilege).
- ρ è una risorsa o un gruppo di risorse (ITResourceObject).
- σ è il segno della regola (positivo o negatvo).

Una identità può essere assegnata a uno o più ruoli; un ruolo può essere specificato come un sottoinsieme di un altro ruolo, permettendo di generare una gerarchia di ruoli (ITRoleAuthorization). Le classi di policy PoSecCo IT descritte finora sono rappresentate nella Figura 2.4.

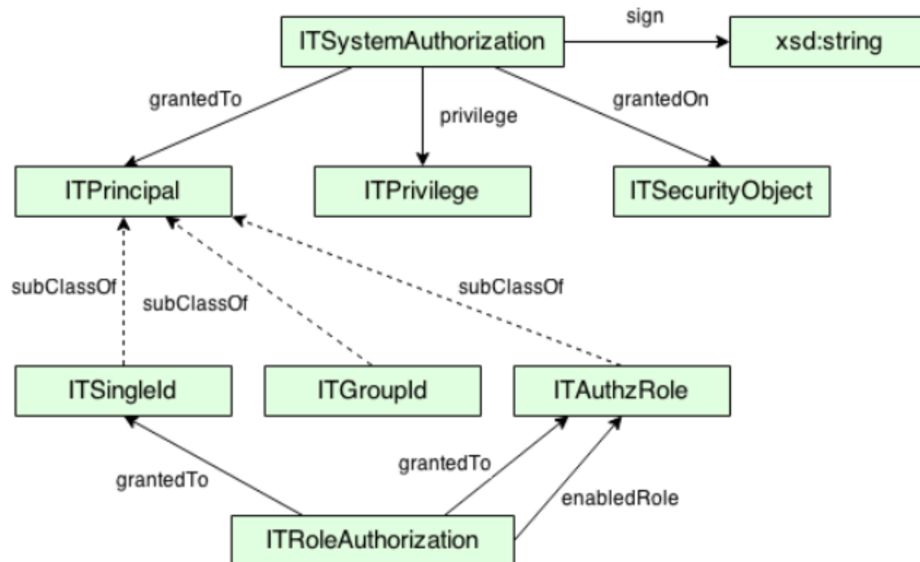

Figura 2.4 Classi di policy di PoSecCo IT relative a autorizzazioni e ruoli

Il "reasoning" DL è usato in PosSecCo per l'armonizzazione delle policy [7]. Non è fornita nessuna funzionalità di decisione per le policy. PosSecCo incarna sicuramente la definizione di architettura generale basata sulla sicurezza per ambienti generici, adattabile anche ad applicazioni business, ma non può essere definito una soluzione per applicazione di sicurezza distribuita a causa della completa mancanza di implementazione.

2.3 Altri framework per le politiche di accesso

Altri due framework per le politiche di accesso, basati su logica DL, sono OWL-POLAR e KaoS, tornando invece alle specifiche XML i più diffusi oltre a XACML sono:

- Web Services policy Language (WSPL): il cui scopo generale è di esprimere politiche di servizio indipendenti dal dominio [2].
- Platform for Privacy Preferences Project (P3P) [3]: è un protocollo che consente ai siti web di dichiarare la loro destinazione d'uso delle informazioni raccolte sulla navigazione degli utenti.

Infine molti sistemi di gestione di politiche, invece, fanno uso dei linguaggi logici come il Prolog (Programmation en logique) o il Datalog, sono linguaggi di programmazione basati sulla logica del primo ordine e associati ad una

intelligenza artificiale [4], i framework più conosciuti di questo tipo sono: Cassandra, PSPL (Portfolio and Service Protection Language) e Rei.

Capitolo 3

Requisiti e specifiche tecniche

Prima di parlare dell'implementazione del progetto è opportuno soffermarsi a riflettere su quelli che sono i requisiti e le aspettative del nostro lavoro. In questo capitolo ci impegneremo a fornire una descrizione formale del sistema partendo da requisiti e arrivando all'architettura che permetterà di soddisfare quei requisiti. Il capitolo affronterà queste questioni solo e unicamente dal punto di vista del design, quindi alcune scelte potrebbero poi essere state implementate in modo diverso o non essere state ancora implementate.
I requisiti di sistema sono esposti nella sezione 3.1 mentre l'architettura e i suoi workflow sono descritti nella sezione 3.2.

3.1 Analisi dei requisiti

Prima di scendere nel dettaglio dell'implementazione è bene analizzare quelli che sono i requisiti del sistema e di conseguenza anche i risultati che ci sia aspetta di ottenere. Dobbiamo quindi identificare gli stakeholders, le entità coinvolte e presentare come interagiscono gli attori con le funzionalità del sistema.

3.1.1 Stakeholders

Gli stakeholders per definizione sono tutti i soggetti, individui od organizzazioni attivamente coinvolti in un'iniziativa economica (progetto, azienda), il cui interesse è negativamente o positivamente influenzato dal risultato dell'esecuzione, o dall'andamento dell'iniziativa e la cui azione o reazione a sua volta influenza le fasi o il completamento di un progetto o il destino di un'organizzazione [8].
Per il sistema proposto si possono identificare due stakeholders: gli **Utenti** e gli **Amministratori di Policy.**

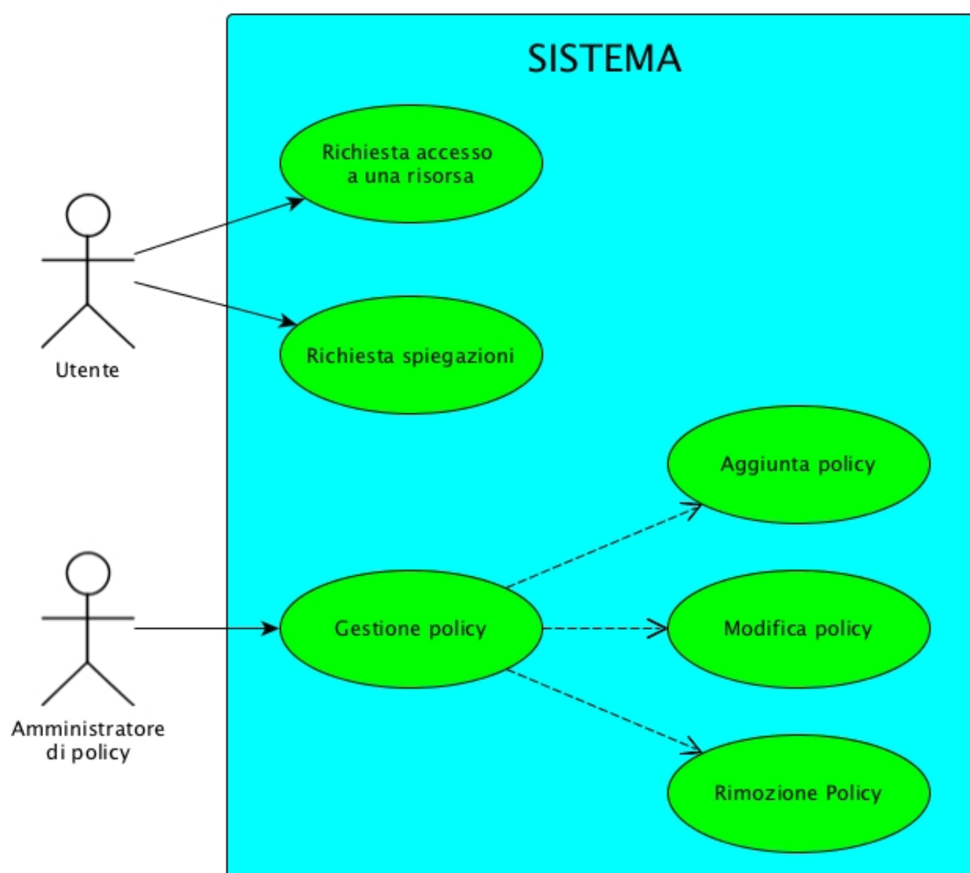

Figura 3.1 Diagramma Use-case

Un **utente** è una generica persona che ha accesso al sistema. Una volta autenticato l'utente può richiedere l'autorizzazione a compiere un'azione su una risorsa, nel nostro caso gli utenti possono chiedere permessi di lettura o di scrittura. Il sistema valuterà se concedere o no il permesso sulla base delle credenziali dell'utente e delle politiche inserite. Dopo aver ricevuto una risposta affermativa o negativa l'utente può richiedere una spiegazione al sistema riguardo la decisione presa, in particolar modo se gli è stato negato l'accesso. È importante sottolineare che ogni utente deve autenticarsi prima di procedere con la sua richiesta, solo in questo modo il sistema sarà in grado di garantire o no l'accesso e fornire una spiegazione del processo decisionale.
L'**amministratore di policy** è la persona incaricata di gestire le politiche nella loro totalità. Ha infatti la possibilità di inserire nuove regole nel sistema, modificare quelle esistenti ed eliminare quelle obsolete. Ogni volta che avviene un cambiamento nella repository delle politiche il sistema deve automaticamente aggiornarsi così che le nuove richieste degli utenti possano essere gestite correttamente tenendo presente dei cambiamenti fatti.

3.1.2 Entità

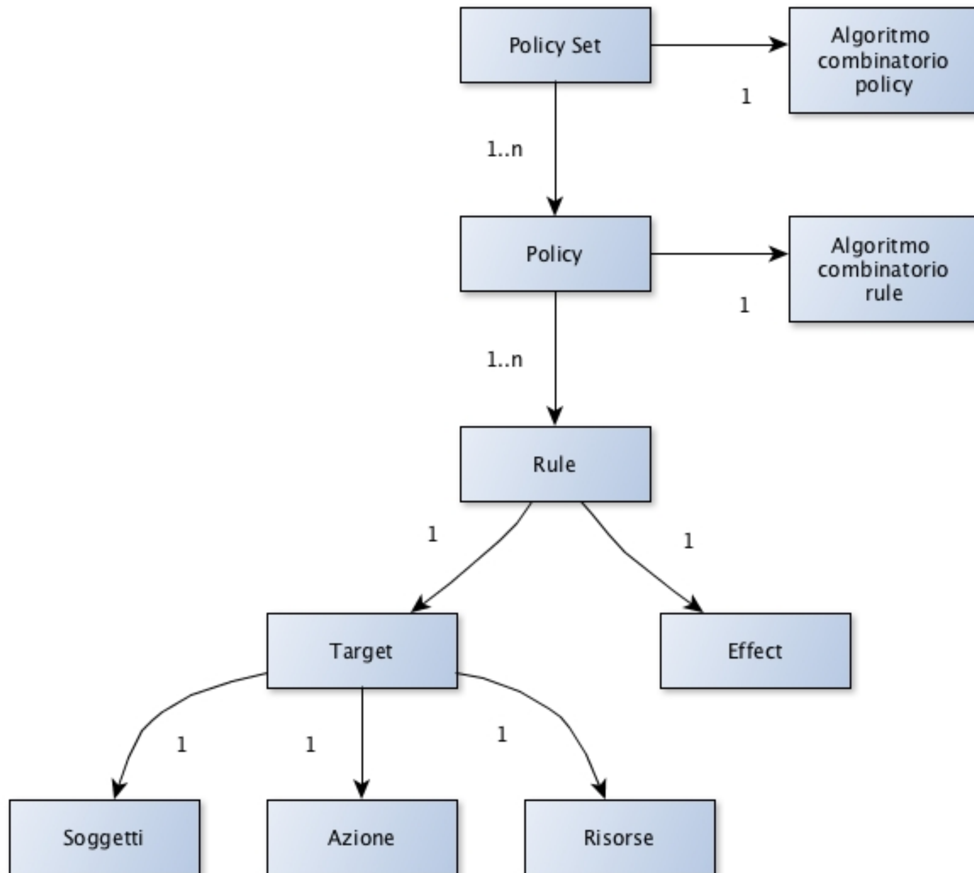

Figura 3.2: Diagramma delle entità del sistema

Le entità del sistema che sono state identificate sono le seguenti:

- *Policy Set*: un insieme non vuoto di policy, raggruppate da un algoritmo combinatorio.
- *Policy*: un insieme non vuoto di rule, raggruppate da un algoritmo combinatorio.
- *Rule*: l'istanza che permette o nega a un insieme di soggetti di compiere una determinata azione su un insieme di risorse.
- *Soggetto*: l'entità a cui è permesso e negato di compiere l'azione sulle risorse.
- *Azione*: l'azione permessa o negata dalla regola, ad esempio *lettura* o *scrittura*.
- *Risorsa:* l'entità su cui, se permessa, viene eseguita l'azione.

3.1.3 Requisiti

L'ultimo punto di questa analisi sono i requisiti trovati. Si possono suddividere in due categorie i **requisiti primari**, sono quelli *funzionali,* cioè quelli senza i quali il sistema non può garantire la corretta esecuzione dei suoi compiti primari. E i **requisiti secondari** che non sono strettamente necessari per garantire le funzionalità elementari del sistema, ma diventano indispensabili nel momento in cui lo si voglia utilizzare concretamente per risolvere problematiche della vita reale.
I **requisiti funzionali** sono:

- *Correttezza nelle risposte:* quando interrogato il sistema deve essere sempre in grado si prendere la decisione di autorizzazione corretta e fornirla all'utente.
- *Correttezza delle spiegazioni:* il sistema deve essere sempre in grado di fornire delle spiegazioni corrette per le decisioni prese, o negare l'accesso per mancanza di dati sufficienti per stabilire se l'utente può o non può essere autorizzato.
- *Gestione delle politiche:* il sistema deve garantire ai suoi amministratori la possibilità di inserire, modificare o eliminare politiche in qualsiasi momento questi decidano di farlo.

I **requisiti secondari** a loro volta possono essere suddivisi in due sottogruppi: quelli che potrebbero indurre il sistema a commettere degli errori di valutazioni, e quindi compromettere tutta la Knowledge-Base, che chiameremo *non-funzionali*. E quelli che impediscono di avere un risultato dal sistema che chiameremo *performanti*.
I **requisiti non-funzionali** sono:

- *Affidabilità:* il sistema deve sempre funzionare correttamente, anche nel caso di connessione persa tra client e server.
- *Coerenza e atomicità:* i dati del sistema devono essere sempre coerenti, cioè non devono verificarsi contraddizioni, anche se due utenti accedono contemporaneamente alla stessa risorsa.
- *Sicurezza:* la sicurezza informatica deve essere sempre garantita, ogni utente deve poter accedere solo alle funzionalità che gli sono consentite dalle sue credenziali di accesso.
- *Puntualità negli aggiornamenti:* il sistema deve aggiornarsi immediatamente quando viene inserita una nuova politica.

I **requisiti di performance** invece sono:

- *Disponibilità:* il sistema deve essere sempre disponibile, lunghi periodi di downtime non portano a errori concettuali gravi, ma di fatto impediscono agli utenti di accedere al sistema, rendendolo quindi inutile.

- *Velocità di risposta:* la velocità di risposta del sistema deve essere pressoché immediata, lunghi periodi di attesa infatti scoraggiano dal considerare il progetto come realmente applicabile in un contesto reale.

3.2 Architettura

Avendo scritto le politiche del nostro sistema in XACML, la scelta più logica per l'architettura del nostro framework è stata quella di sceglierne una XACML-based, così da mantenerci il più possibile conformi e coerenti.

3.2.1 Componenti principali

I componenti principali del sistema proposto sono perciò cinque, gli stessi descritti nella sezione 2.1.

- Policy Enforcement Point (PEP)
- Policy Decision Point (PDP)
- Policy Administration Point (PAP)
- Policy Information Point (PIP)
- Context Handler

Nella sezione seguente descriverò nel dettaglio questi componenti e le loro interazioni.

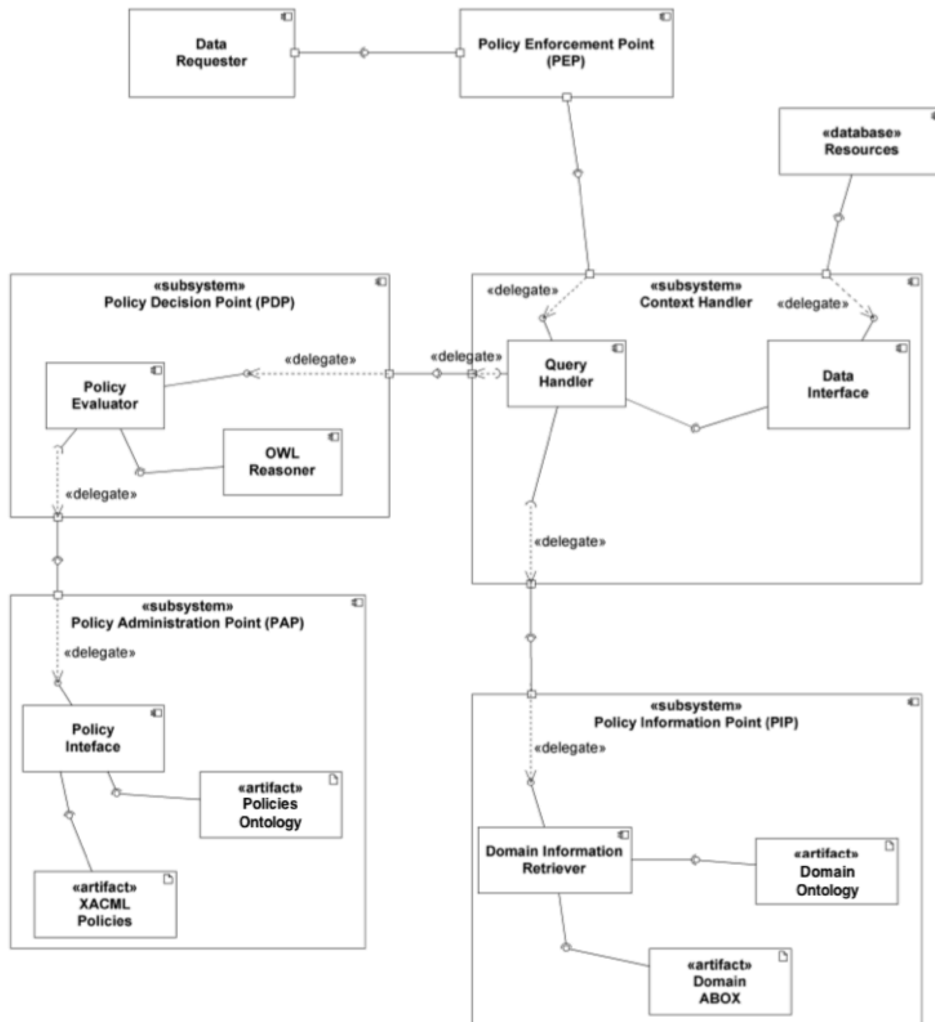

Figura 3.3: l'architettura del framework

Policy Enforcement Point (PEP)
Il primo componente che troviamo in tutti i flussi di esecuzione dell'utente è il PEP. Ha due compiti: il primo è quello di ricevere la richiesta di accesso a una risorsa e inoltrarla al PDP, attraverso il Context Handler, al fine di ottenere una decisione positiva o negativa per l'accesso richiesto. Una volta che la valutazione è stata effettuata e il risultato è stato mandato indietro al PEP, il suo secondo compito è quello lasciare che l'utente acceda alla risorsa o di comunicargli che la sua richiesta è stata rifiutata.

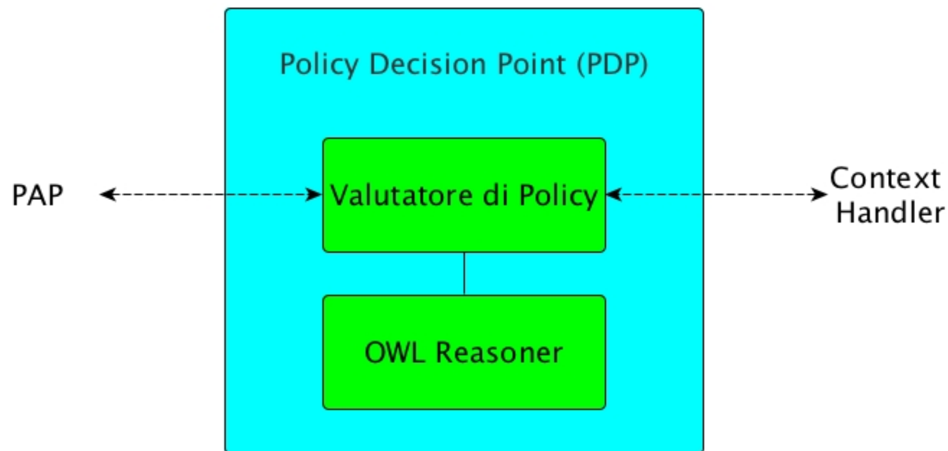

Figura 3.4: Struttura del PDP

Policy Decision Point (PDP)
Il PDP è il componente che si occupa di decidere se concedere o no all'utente di compiere l'azione richiesta su una data risorsa. Come si può vedere dalla Figura 3.4 per compiere la sua funzione il PDP utilizza un Valutatore di politiche che si interfaccia con un reasoner di tipo OWL DL. Quando riceve una richiesta il Valutatore di politiche richiede al PAP tutte le informazioni riguardo le politiche che regolano l'accesso alla risorsa richiesta. Dopo di che, usando il reasoner di cui è provvisto estrae tutta la conoscenza possibile da queste politiche, vale a dire tutti i permessi e le restrizioni che impongono. Infine confrontando i dati appena ottenuti con la richiesta dell'utente valuta se concedere o no l'accesso ai dati richiesti e inoltra la sua desicione al Context Handler.

Policy Administration Point (PAP)
Il PAP è l'entità più vicina alle politiche, i suoi compiti sono:

- gestire l'inserimento, la modifica e la rimozione di politiche senza che si creino conflitti,
- mantenere l'ontologia costantemente sincronizzata con le politiche,
- fornire al PDP le informazioni che gli vengono richieste.

Come mostra la Figura 3.5 il PAP è composto da tre elementi: una *Interfaccia di Policy*, una *Ontologia delle Policy* e un *file XACML*. Andando a modificare il file XACML è possibile aggiungere nuove politiche, modificare quelle esistenti o rimuovere quelle obsolete. L'Ontologia delle policy altro non è che una TBOX e una RBOX OWL che contengono al loro interno l'ontologia di dominio e le politiche tradotte in OWL. Mentre l'Intefaccia di policy è l'entità che si occupa

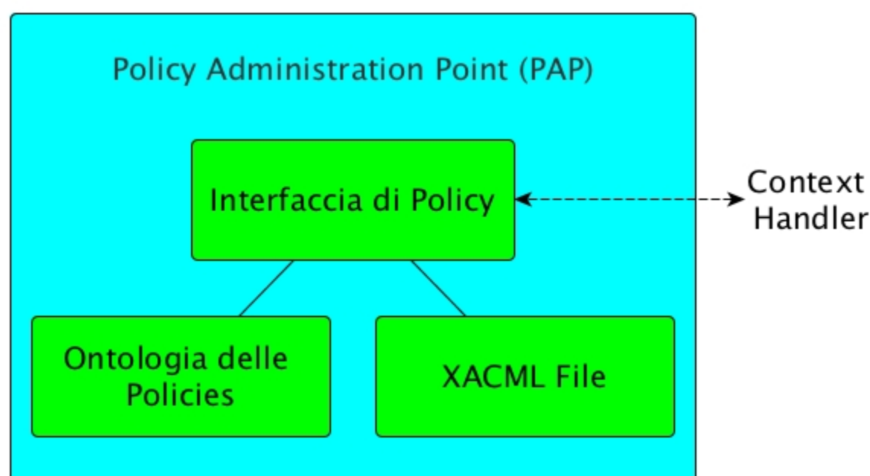

Figura 3.5: struttura del PAP

della traduzione delle politiche da XACML a OWL. Maggiori dettagli sulla traduzione saranno fornito nella sezione 4.

Policy Information Point (PIP)
Il PIP contiene e gestisce tutte le informazioni relative ai domini di applicazione. Dentro al PIP i domini sono rappresentati attraverso due ontologie differenti: una *Ontologia di dominio* e una *ABox di dominio*. Le informazioni sono raccolte e gestiste da un *Raccoglitore di informazioni*, che interagisce col Context Handler quando arriva una richiesta di accesso. L'ontologia di dominio è una TBox Owl che copre l'intero insieme di concetti (Classi OWL) e relazioni (Propietà di oggetto e dati OWL). La ABox di dominio e una Box OWL assertiva che contiene i diversi individui e risorse del dominio.

Context Handler
Il Context Handler è il componente che si occupa della comunicazione tra PEP, PDP e PIP. La sua risorsa principale è il *Query Handler*, che riceve la richiesta dal PEP, raccoglie tutte le informazioni di dominio utili dal PIP, e inoltra tutto quanto al PDP. Se la decisione presa è positiva il Query Handler si occupa anche di raccogliere la risorsa o le risorse richieste attraverso una *Interfaccia Dati*. Per concludere la riposta è spedita indietro al PEP, possibilmente insieme alla risorsa richiesta. E importante sottolineare che il Database da cui il Context Handler preleva la risorsa è esterno al componente stesso. La struttura del Context Handler è mostrata nella Figura 3.6.

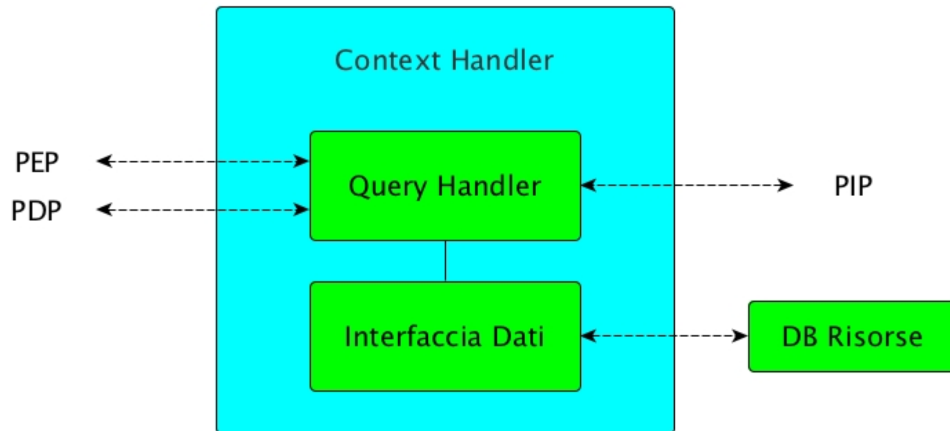

Figura 3.6: struttura del Context Handler

3.2.2 Flussi di esecuzione

Come descritto nella sezione precedente i possibili flussi di esecuzioni del nostro framework sono tre. Questo paragrafo si occupa delle descrizione approfondita dei vari flussi e della presentazione di un sequence diagram per ognuno di essi.

Richiesta di accesso alle risorse
Quando un utente richiede di accedere a una o più risorse la sua richiesta arriva al PEP, che la inoltra al Context Handler. Quest'ultimo raccoglie le informazioni di dominio, inerenti alla richiesta, dal PIP e manda tutto quanto al PDP. Il PDP fa richiesta al PAP di tutte le politiche che regolano l'accesso alle risorse coinvolte, poi invoca il reasoner e coi dati raccolti prende una decisione. La decisione presa fa il percorso inverso cioè passa dal PDP al PEP, passando per il Context Handlet, e arriva infine all'utente. La Sequenza è mostrata nella Figura 3.7.

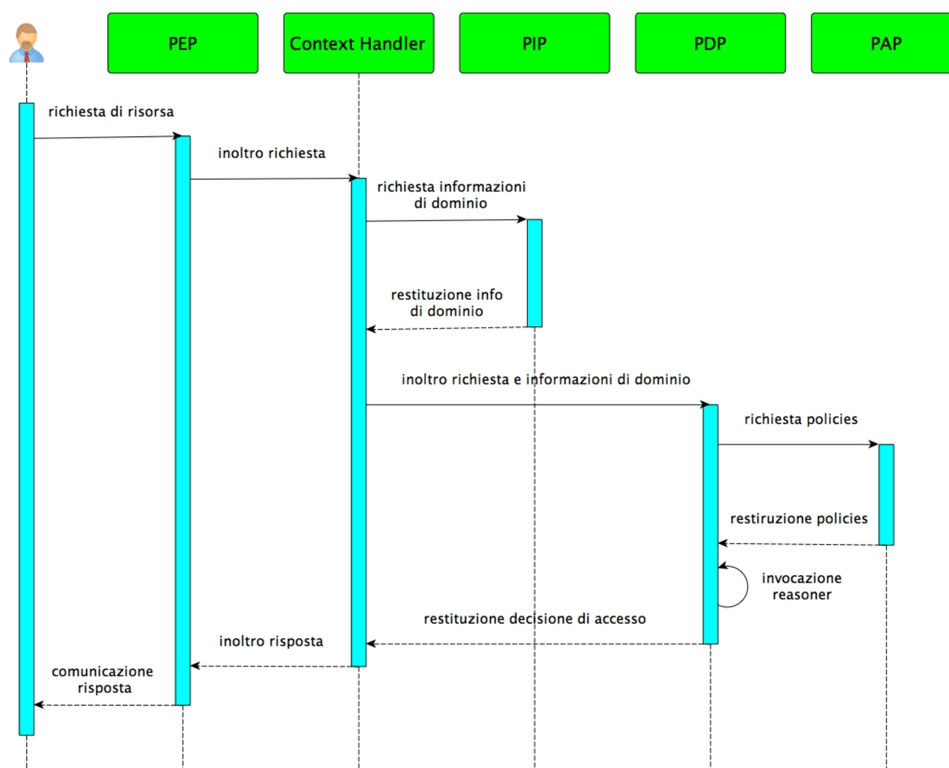

Figura 3.7: sequence diagram per richiesta di autorizzazione

Richiesta di spiegazioni

Una volta che l'utente riceve la decisione di accesso può richiedere al sistema delle spiegazioni a riguardo, in particolar modo se l'accesso gli è stato negato. La richiesta di spiegazioni arriva per primo al PEP che attraverso il Context Handler la inoltra al PDP, quest'ultimo invoca il reasoner e per conoscere perchè l'accesso è stato permesso o negato. Infine la spiegazione è mandata indietro all'utente seguendo il consueto percorso: PDP, Context Handler, PEP. Questa sequenza è mostrata nella Figura 3.8.

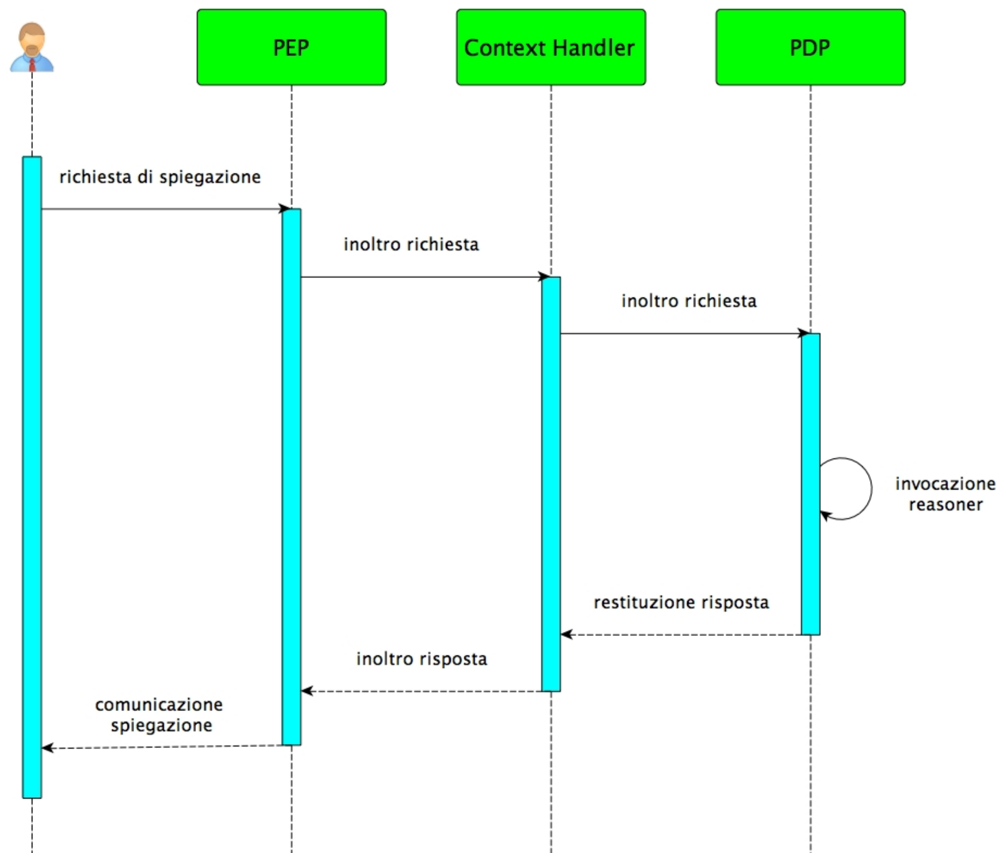

Figura 3.8: sequence diagram per richiesta di spiegazione

Gestione delle politiche

L'intero sistema di modifica delle politiche è gestito dal PAP, quando un amministratore modifica il file XACML delle politiche l'Interfaccia di policy richiede subito la nuova versione del file, esegue l'algoritmo di traduzione e inserisce le nuove politiche tradotte in OWL nell'ontologia esistente. Quando la procedura è stata completata un messaggio di successo viene inviato all'amministratore. Questa sequenza è mostrata nella Figura 3.9.

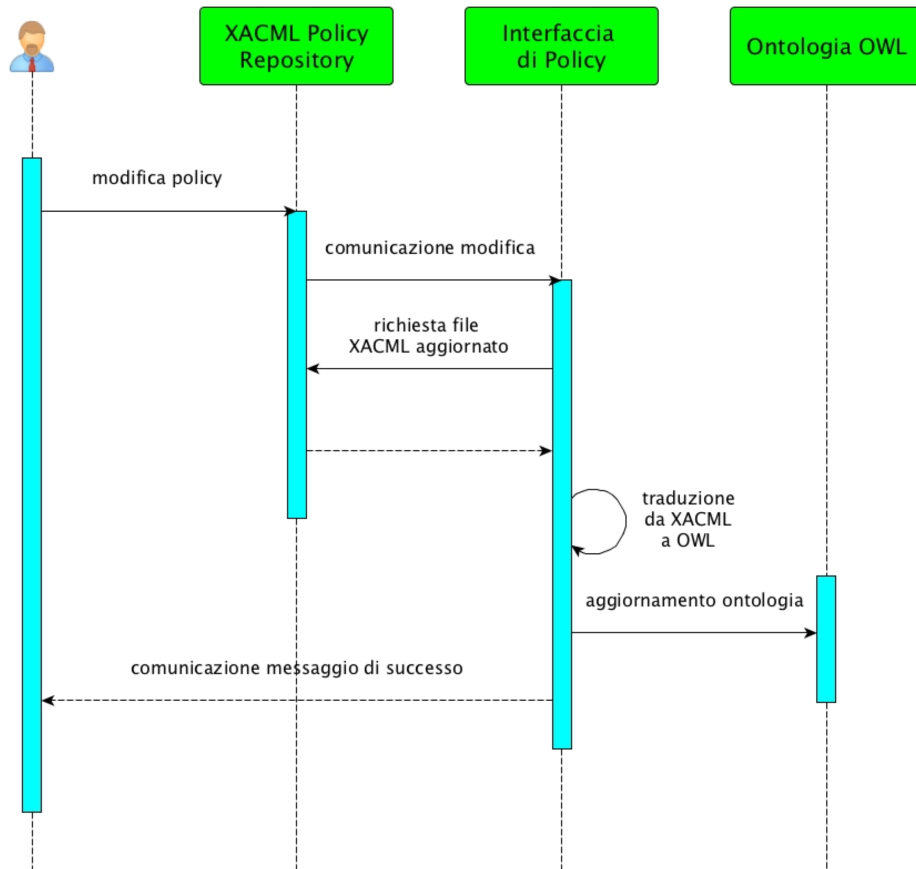

Figura 3.9: sequence diagram per la modifica delle politiche

Capitolo 4

Framework

Dopo aver fornito un'inquadratura generale dei requisiti e dell'architettura del sistema, entreremo nel dettaglio di quello che è stato realmente implementato e di come è stato fatto. I moduli implemenati sono: PDP, PAP e PIP, mentre il Context Handler e il PEP sono stati introdotti solo concettualmente per fornire una più completa visione d'insieme del sistema.
Il lavoro di questa tesi riprende da dove si è interrotto quello dell'elaborato "OWL-Based Representation and Enforcement of Data Access Policies" [12] e si può suddividere in due parti: la prima la traduzione delle politiche da XACML a OWL senza l'utilizzo della TopObjectProperty. E la seconda parte che consiste nella creazione del modulo che permette all'utente di richiedere al sistema le spiegazioni per le decisioni prese.

4.1 Traduzione delle policy

Il primo compito è stato quello di modificare il sistema esistente al fine di eseguire la traduzione delle politiche, da XACML a OWL, senza l'utilizzo della TopObjectProperty. Questo perché nessun reasoner tra quelli analizzati (Hermit, Pellet, Fact++) era in grado di fornire spiegazioni sui suoi ragionamenti se gli assiomi dell'ontologia erano costruiti con la suddetta proprietà. Come descritto nella sezione 3.2.1 questo processo è gestito dal Policy Administration Point (PAP) e in particolare dall'Interfaccia di policy. Nel PAP risiede anche il file XACML, ogni volta che questo file viene modificato l'Iterfaccia di policy esegue l'algoritmo di traduzione e costruisce una rappresentazione in OWL delle politiche, quest'ultima viene poi salvata in una ontologia OWL (RBox + TBox).
In questa sezione è descritto ogni aspetto del processo di traduzione, inclusa la struttura delle politiche e l'algoritmo di traduzione.

4.1.1 Il file delle policy XACML

Nel precedente capitolo abbiamo già descritto la struttura del file XACML, ma al fine di tradurre correttamente le politiche nell'ontologia OWL, è stato necessario introdurre alcune limitazioni all'espressività del linguaggio XACML.
Prima di tutto il nostro policy file consiste in un unico *PolicySet,* che a sua volta deve contenere solo due elementi *Policy.* Il primo contiene tutte le sigole *rule* che sono utilizzate per eseguire i controlli di accesso, mentre il secondo contiene una singola rule che nega tutto quanto. L'unico algoritmo combinatorio di policy applicabile è *first-applicable:* in questo modo se una qualunque rule del primo elemento policy è applicabile allora il sistema applica quella, altrimenti è applicata la seconda policy, vale a dire l'accesso alle risorse viene negato.
La policy principale è quindi l'insieme di rule che regolano l'accesso alle risorse. Anche all'interno di quest'ultima è stata introdotta una restrizione, infatti l'unico algoritmo combinatorio di rule applicato è il deny-overrides. L'algoritmo prende in considerazione tutte le regole e se ne trova una che permette l'accesso e una che lo nega allora l'accesso viene negato. La stesa cosa avviene se non è trovata nessuna rule che regola l'accesso alla risorsa richiesta, la policy finale infatti nega ogni richiesta di accesso. Ogni regola ha un *effetto* (*Permit/Deny*) e un *target*, che contiene una lista di *soggetti,* una lista di *risorse* e un'*azione.* Ogni regola può identificare una serie di soggetti o risorse uniti da congiunzioni AND o OR, come previsto dal modello XACML. Le regole devono essere definite secondo alcuni modelli predefiniti, come descritto nella sezione 4.1.2.

4.1.2 Modelli di rule

L'idea di base del processo di traduzione è quella di esprimere le regole definendo una catena di porprietà, rappresentando la relazione tra *soggetti* e *risorse*, come una sotto proprietà dell'*azione* da eseguire. L'elenco dei passaggi per ottenere questo risultato in quattro tipi di regole su cinque è il seguente:

1. Creare, se non esiste, una classe per i soggetti.
2. Creare una proprietà che collega tutti gli individui della classe dei soggetti con un oggetto ponte 'o' (everyClassSubjects).
3. Creare, se non esiste, una classe per le risorse.
4. Creare una proprietà che collega tutti gli individui della classe delle risorse con l'oggetto 'o' (everyClassResources).
5. Creare la seguente catena di proprietà: everyClassSubjects o inverseOfeveryClassResources.
6. Impostare la catena creata come una sotto proprietà dell'azione (che per esempio potrebbe essere canRead).

L'oggetto 'o' viene creato prima di iniziare il processo di traduzione e viene usato come espediente per creare la relazione universale tra gli individui delle due classi, cioè serve a connettere tutti gli elementi della classe Soggetti con tutti gli

ID	Referenza	Descrizione	Esempio
1	IBAC	Un singolo soggetto può accedere a una o più risorse	Marco Rossi può leggere i documenti di assistenza a Health Care
2	RBAC	Un gruppo di soggetti può accedere a una o più risorse	I consulenti medici possono modificare i documenti di Medical Regulation
3	ABAC	Solo i soggetti con uno specifico attributo possono accedere a una o più risorse	Le donne non possono leggere i documenti di Andrologia
4	ABAC	Solo i soggetti in una specifica relazione con un altro soggetto con specifici attributi possono accedere a una o più risorse	I tutori dei minori possono leggere il documento 305871
5	N/A	Solo i soggetti in una specifica relazione con un altro soggetto possono accedere alle risorse che si riferiscono a quest'ultimo soggetto	Un soggetto può leggere tutti i record del dipartimento in cui lavora

Tabella 4.1: modelli delle rule

elementi della classe Risorse. Come detto questo processo di traduzione è applicato nella maggior parte dei casi, anche se ogni regola presenta qualche leggera modifica, i cinque modelli di regole sono presentati nella Tabella 4.1. Ora andremo a descrivere i singoli modelli nel dettaglio, gli assiomi sono presentati usando la *sintassi Manchester* [9].

Tipo 1: Rule IBAC

Il primo modello di regola consiste nel concedere o negare il permesso, di accedere a una o più risorse, a un singolo soggetto. Questo modello rappresenta la più semplice e probabilmente la più usata forma di controllo di accesso: l'Identity-Based Access Control (IBAC) che garantisce o nega l'accesso basandosi sull'identità dell'utente che effettua la richiesta. Un esempio di schema XACML di questo tipo di regola è fornito nella Figura 4.1, in cui viene permesso a `marco_rossi` di `leggere` le risorse che appartengono alla classe `HealthcareAssistantDocument`.

```xml
<!—Rule che permette a marco_rossi di leggere i documenti HealthcareAssistantDocument -->
<Rule RuleId="Rule1" Effect="Permit"
        <Target>
                        <Subjects>
                           <Subject>
                                <SubjectMatch MatchId="urn;polimi:names:dbsp:1:function:ontology-id-equal">
                                <AttributeValue DataType="urn;polimi:names:dbsp:1:data-type:ontology-id">marco_rossi</AttributeValue>
                                <SubjectAttributeDesignator AttributeId="urn;polimi:names:dbsp:1:attribute:id" MustBePresent="true"
                                        DataType="urn;polimi:names:dbsp:1:data-type:ontology-id"/>
                                </SubjectMatch>
                           </Subject>
                        </Subjects>

        <Resources>
                        <Resource>
                           <ResourceMatch MatchId="urn;polimi:names:dbsp:1:function:ontology-id-equal">
                             <AttributeValue DataType="urn;polimi:names:dbsp:1:data-type:ontology_id">healthcareAssistantDocument</AttributeValue>
                                <ResourceAttributeDesignator AttributeId="urn;polimi:names:dbsp:1:attribute:class"
                                DataType="urn;polimi:names:dbsp:1:data-type:ontology-id"/>
                                </ResourceMatch>
                           </Resource>
        </Resources>

        <Actions>
                <Action>
                        <ActionMatch MatchId="urn;polimi:names:dbsp:1:function:ontology-id-equal">
                        <AttributeValue DataType="urn;polimi:names:dbsp:1:data-type:ontology-id">read</AttributeValue>
                                <ActionAttributeDesignator DataType="urn;polimi:names:dbsp:1:data-type:ontology-id"
                                AttributeId="urn;polimi:names:dbsp:1:attribute:id" />
                        </ActionMatch>
                </Action>
        </Actions>
        </Target>
 </Rule>
```

Figura 4.1: Esempio XACML di rule 1 (IBAC)

Questa regola è tradotta nell'ontologia OWL seguendo la seguente procedura. Prima di tutto viene generata una classe che contiene solo l'individuo `marco_rossi`:

```
Class:
     Class_marco-rossi
equivalentTo:
     {marco_rossi}
```

Poi viene creata la objectProperty `every_Class_marco_rossi` che collega tutti gli elementi di questa classe con l'elemento 'o':

```
Class:
     Class_marco-rossi
equivalentTo:
     [marco_rossi]
     everyClass_marco_rossi exactly 1 Class_O
```

La stessa procedura è applicata alle risorse. In questo caso c'è un'unica risorsa rappresentata dalla classe OWL `HealthcareAssistantDocument`, quindi non è necessario creare una nuova classe dal momento che già esiste. Viene invece generata la objectProperty che collega tutti gli individui appartenenti a questa classe all'elemento ponte 'o'.

```
Class:
     Class_HealthcareAssistantDocument
     everyClass_HealthcareAssistantDocument exactly 1 Class_O
```

L'ultimo passaggio prende in considerazione l'azione da eseguire sulle risorse. In questo caso la regola indica `read` come azione e l'effetto è `Permit`, quindi l'azione risultante è `canRead`. In generale se l'effetto è `Permit` il prefisso *can* è aggiunto al nome dell'azione altrimenti si aggiunte *canNot*. La corrispondente proprietà è quindi impostata come super-property della catena che proprietà specificata dalla regola:

```
objectPropertyChain:
     every_Class_marco_rossi o
     inverseOf(every_ Class_HealthcareAssistantDocument)

subPropertyOf:
     canRead
```

Grazie a questa procedura, l'individuo `marco_rossi` risulta collegato attraverso la proprietà `canRead` a tutti gli `HealthCareAssistantDocument.`

Tipo 2: Rule RBAC

Il secondo modello consiste nel garantire o negare l'accesso a un gruppo di soggetti che chiedono di operare su una o più risorse. Come il primo anche questo tipo di regola è molto comune: il Role-Based Access Control (RBAC) permette di accede alle risorse controllando l'appartenenza del soggetto a un determinato gruppo di utenti. Un esempio di questo tipo di regola è mostrato nella Figura 4.2, in cui viene dato il permesso di `write` a tutti i membri della classe `MedicalAssistant` sui documenti che appartengono alla classe `MedicalRegulationDocument`.

La procedura di traduzione è molto simile alla precedente, con la differenza che qui esistono già le classi OWL dei soggetti e delle risorse, devono solo essere create le objectPorperty che collegano tutti gli individui di entrambe le classi con il solito oggento ponte 'o':

```
Class:
     Class_MedicalAssistant
     everyMedicalAssistant exactly 1 Class_O
```

```xml
<!—Rule che permette ai medicalConsultant di modificare i medicalRegulationDocument-->
<Rule RuleId="Rule2" Effect="Permit">
        <Target>
                        <Subjects>
                          <Subject>
                               <SubjectMatch MatchId="urn;polimi:names:dbsp:1:function:ontology-id-equal">
                               <AttributeValue DataType="urn;polimi:names:dbsp:1:data-type:ontology-id">medicalConsultant</AttributeValue>
                               <SubjectAttributeDesignator AttributeId="urn;polimi:names:dbsp:1:attribute:class" MustBePresent="true"
                               DataType="urn;polimi:names:dbsp:1:data-type:ontology-id"/>
                               </SubjectMatch>
                          </Subject>
                        </Subjects>

        <Resources>
                <Resource>
                         <ResourceMatch MatchId="urn;polimi:names:dbsp:1:function:ontology-id-equal">
                        <AttributeValue DataType="urn;polimi:names:dbsp:1:data-type:ontology-id">medicalRegulationDocument</AttributeValue>
                        <ResourceAttributeDesignator AttributeId="urn;polimi:names:dbsp:1:attribute:class"
                               DataType="urn;polimi:names:dbsp:1:data-type:ontology-id"/>
                        </ResourceMatch>
                </Resource>
        </Resources>

        <Actions>
                <Action>
                        <ActionMatch MatchId="urn;polimi:names:dbsp:1:function:ontology-id-equal">
                        <AttributeValue DataType="urn;polimi:names:dbsp:1:data-type:ontology-id">write</AttributeValue>
                        <ActionAttributeDesignator DataType="urn;polimi:names:dbsp:1:data-type:ontology-id"
                        AttributeId="urn;polimi:names:dbsp:1:attribute:id" />
                        </ActionMatch>
                </Action>
        </Actions>
        </Target>
</Rule>
```

Figura 4.2: Esempio XACML di rule 2 (RBAC)

```
Class:
     Class_MedicalRegulationDocument
     everyMedicalRegulationDocument exactly 1 Class_O
```

Infine si considera l'azione `write` e l'effetto `Permit`, dunque la combinazione risultante è `canWrite`. Questa proprietà è impostata come super-property della catena di proprietà identificate dalla regola.

```
objectPropertyChain:
     everyMedicalAssistant o
     inverseof(everyMedicalRegulationDocument)

subPropertyOf:
     canWrite
```

Dopo questa procedura ogni membro della classe `MedicalAssistant` risulta connesso attraverso la proprietà `canWrite` a ogni elemento appartenente alla classe `MedicalRegulationDocument`.

Tipo 3: Rule ABAC semplice

Nel terzo modello di regola l'accesso è permesso o negato solo a quei soggetti che possiedono uno o più determinati attributi. Questa regola infatti è classificata

```xml
<!—Rule che non permette alle donne di leggere i documenti di Andologia-->
<Rule RuleId="Rule3" Effect="Deny">
        <Target>
                <Subjects>
                        <Subject>
                            <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">F</AttributeValue>
                            <SubjectAttributeDesignator AttributeId="urn;polimi:names:dbsp:1:attribute:dataProperty:hasSex" MustBePresent="true"
                                    DataType="http://www.w3.org/2001/XMLSchema#string"/>
                            </SubjectMatch>
                        </Subject>
                      </Subjects>

        <Resources>
                        <Resource>
                            <ResourceMatch MatchId="urn;polimi:names:dbsp:1:function:ontology-id-equal">
                            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">AndrologyDocument</AttributeValue>
                            <ResourceAttributeDesignator AttributeId="urn;polimi:names:dbsp:1:attribute:id"
                                    DataType="urn;polimi:names:dbsp:1:data-type:ontology-id"/>
                            </ResourceMatch>
                        </Resource>
        </Resources>

        <Actions>
                <Action>
                      <ActionMatch MatchId="urn;polimi:names:dbsp:1:function:ontology-id-equal">
                      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
                      <ActionAttributeDesignator DataType="urn;polimi:names:dbsp:1:data-type:ontology-id"
                  AttributeId="urn;polimi:names:dbsp:1:attribute:id" />
                      </ActionMatch>
                </Action>
        </Actions>
        </Target>
</Rule>
```

Figura 4.3: Esempio XACML di rule 3 (ABAC semplice)

come: Attribute-Based Access Control (ABAC) cioè controllo di accesso basato sugli attributi. Un esempio lo si può avere dalla Figura 4.3: in cui vediamo che non è consentito l'accesso in lettura ai documenti di andologia (`AndrologyDocument`) per tutte le donne (vale a dire tutti i soggetti che posseggono l'attributo `hasGender` uguale a "F"). Anche in questo caso la procedura di traduzione segue i soliti passaggi: viene generata una nuova classe OWL che contiene tutti gli individui che possiedono il valore "F" per la dataProperty `hasGender`:

```
Class:
     Class_hasGender_F
equivalentTo:
      hasGender value “F”
```

Poi tutti gli individui della classe creata sono collegati con l'oggetto "o":

```
Class:
     Class_hasGender_F
equivalentTo:
      hasGender value “F”
      everyClass_hasGender_F exactly 1 Class_O
```

`La solita procedura è applicata alle risorse:`

```
Class:
     Class_AndrologyDocument
     everyAndrologyDocument exactly 1 Class_O
```

E infine la catena di proprietà ottenuta viene impostata come sotto proprietà dell'azione indicata, in questo caso essendo la proprietà read e l'effetto Deny il risultato è `canNotRead:`

```
objectPropertyChain:
     everyClass_hasGender_F o
     inverseof(everyAndrologyDocument)

subPropertyOf:
     canNotRead
```

Dopo questa procedura ogni individuo femmina risulta connesso attraverso la proprietà `CanNotRead` a ogni documento appartenente alla classe `AndrologyDocument`.

Tipo 4: ABAC Rule composta
Il quarto modello di regola si differenzia leggermente dai precedenti, anch'essa è classificarta come ABAC Attribute-Based Access Control come la terza ma a differenza di quest'ultima l'attributo a cui fa riferimento non è del soggetto ma d un individuo con cui il soggetto è in relazione. In Altre parole l'accesso è permesso o negato a quei soggetti connessi tramite una particolare objectProperty a una classe di individui caratterizzati da certi attributi. L'esempio, illustrato nella Figura 4.4, mostra come gli individui tutori (`isTutorOf`) di soggetti con dataProperty `hasAge` minore di "`18`", possano leggere lo specifico doumento `document_305871`, o più brevemente ai tutori dei minori è consentito leggere il documento specificato. Il processo di taduzione, leggermente più lungo dei precedenti, comincia con la creazione di una classe per gli individui con dataProperty `hasAge` minore di "`18`":

```
Class:
      Class_hasAge_lt_18
equivalentTo:
      hasAge some int[<18]
```

```xml
<!—Rule che permette ai tutori dei minori di leggere il file document_305871-->
<Rule RuleId="Rule4" Effect="Permit">
        <Target>
                <Subjects>
                  <Subject>
                <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:lower-than">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#integer">18</AttributeValue>
                        <SubjectAttributeSelector Path="urn:polimi:names:dbsp:1:attribute:objectProperty:isTutorOf/urn:polimi:names:dbsp:1:
                        attribute:dataProperty:hasAge"  MustBePresent="true" DataType="http://www.w3.org/2001/XMLSchema#integer"/>
                </SubjectMatch>
                  </Subject>
                </Subjects>

                <Resources>
                  <Resource>
                        <ResourceMatch MatchId="urn:polimi:names:dbsp:1:function:ontology-id-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">document_305871</AttributeValue>
                        <ResourceAttributeDesignator AttributeId="urn:polimi:names:dbsp:1:attribute:id"
                        DataType="urn:polimi:names:dbsp:1:data-type:ontology-id"/>
                        </ResourceMatch>
                  </Resource>
                </Resources>

                <Actions>
                 <Action>
                        <ActionMatch MatchId="urn:polimi:names:dbsp:1:function:ontology-id-equal">
                        <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
                        <ActionAttributeDesignator DataType="urn:polimi:names:dbsp:1:data-type:ontology-id"
                  AttributeId="urn:polimi:names:dbsp:1:attribute:id" />
                 </ActionMatch>
                </Action>
                </Actions>
        </Target>
</Rule>
```

Figura 4.4: Esempio XACML di rule 4 (ABAC composta)

Dopodiché viene generata la classe dei tutori, e cioè di tutti gli individui collegati alla classe precedente dalla proprietà `isTutorOf`:

```
Class:
      Class_isTutorOf_hasAge_lt_18
equivalentTo:
      isTutorOf some Class_hasAge_lt_18
```

La classe appena creata viene collegata con una relazione universale all'oggetto "o":

```
Class:
      Class_isTutorOf_hasAge_lt_18
equivalentTo:
      isTutorOf some Class_hasAge_lt_18
      Class_isTutorOf_hasAge_lt_18 exactly 1 Class_O
```

In questo caso la risorsa è un singolo individuo viene perciò creata una classe equivalente all'individuo e collegata con l'oggetto "o":

```
Class:
      Class_document_305871
equivalentTo:
```

```
{document_305871}
everyClass_document_305871 exactly 1 Class_O
```

L'azione della regola è `read` e l'effetto è `Permit`. Quindi l'objectProperty `canRead` è impostata come super-property della catena di proprietà che collega i soggetti alla risorsa:

```
objectPropertyChain:
    every Class_isTutorOf_hasAge_lt_18 o
    inverseof(everyClass_document_305871)

subPropertyOf:
    canRead
```

Il risultato di questa procedura è che ogni tutore di minori può leggere l'oggetto document_305871.

Tipo 5: Rule triangolo

La quinta e ultima regola del modello rappresenta una novità, essa infatti differisce completamente da tutti i modelli precedenti e non appare in nessun esempio della letteratura sul controllo di accessi. La particolarità di questa regola è che soggetti e risorse sono messi in relazione: l'accesso è consentito o negato solo se un particolare individuo ha una specifica connessione con i soggetti e le risorse. Abbiamo definito questa regola triangolo poichè la proprietà di cui si sta chiedendo il permesso può essere vista come un lato del triangolo in cui soggetti, risorse e individui comuni rappresentano i vertici. La Figura 4.5 mostra un esempio di questo modello: una persona può leggere ogni `MedicalRecord` del dipartimento (`Ward`) in cui lui o lei lavora. Per i passaggi dell'algoritmo di traduzione di questo tipo di regola si può fare riferimento alla tesi "OWL-Based Representation and Enforcement of Data Access Policies" [12] dal momento che non sono state apportate modifiche.

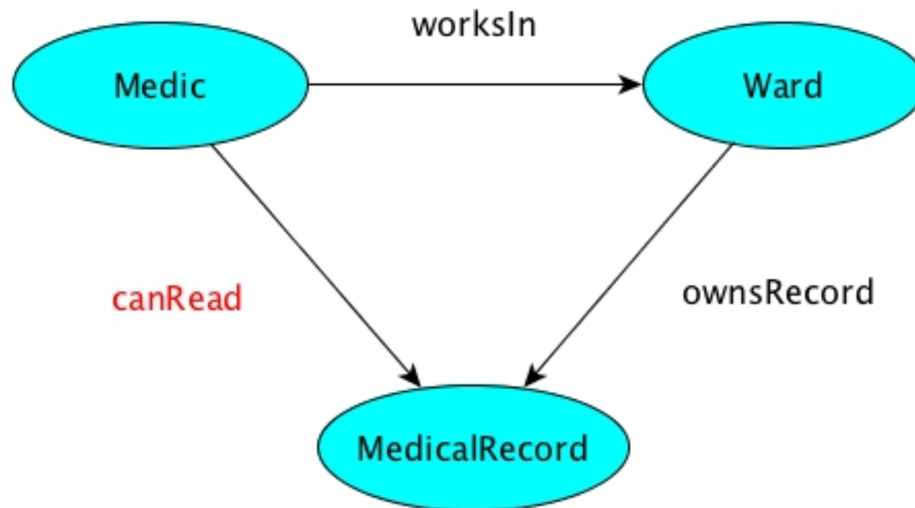

Figure 4.5: la relazione triangolo della rule 5

4.1.3 Composizione di rule

Come precedentemente affermato nella sezione 4.1.1, più soggetti o risorse possono essere collegati in una singola regola usando congiunzioni OR o AND. Nel primo caso soggetti (o risorse) multipli devono essere definiti nel target della regola attraverso multipli tag <Subject> mentre nel secondo caso devono essere aggiunti più elementi subjectMatch all'interno di un unico tag <Subject>. È facile intuire che a causa delle differenti procedure usate per tradurre i vari tipi di regole potrebbero sorgere alcuni problemi nel connettere varie politiche. La seguente sezione illustra brevemente come gestire i conflitti, per un'analisi più approfondita si rimanda a "OWL-Based Representation and Enforcement of Data Access Policies" [12].

Congiunzione OR

In caso di più soggetti connessi con una congiunzione OR è sufficiente definire regole multiple. La regola sarà tradotta in un numero di regole OWL equivalenti al numero dei soggetti, indipendentemente dai modelli di regole coinvolti. Ovviamente la stesa cosa vale per regole con più risorse. Un esempio potrebbe essere una politica che permette ai minori e ai tutori di minori di leggere il document_108143, in questo caso una regola di tipo 3 e una di tipo 4 sono unite in una congiunzione OR. Il sistema tratterà i due soggetti indipendentemente e le due regole OWL finali saranno:

- I minori possono leggere il document_108143
- I tutori dei minori possono leggere il document_108143

Un esempio XACML della congiunzione OR è visibile nella Figura 4.6.

```
<!—Regola che permette ai minori e ai loro genitori di leggere il document_108143-->
<Rule RuleId="Rule7" Effect="Permit">
  <Target>
    <Subjects>
      <Subject>
          <SubjectMatch MatchId=" urn:oasis:names:tc:xacml:1 .0 :function:lowerthan">
          <AttributeValue DataType=" http://www.w3. org/2001/XMLSchema#integer
          ">18</AttributeValue>
          <SubjectAttributeDesignator AttributeId="
          urn:polimi:names:dbsp:1:attribute:dataProperty:hasAge "
          MustBePresent="true" DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </SubjectMatch>
      </Subject>
      <Subject>
          <SubjectMatch MatchId=" urn:oasis:names:tc:xacml:1 .0 :function:lowerthan"><AttributeValue
          DataType=" http://www.w3. org/2001/XMLSchema#integer ">18</AttributeValue>
          <SubjectAttributeSelector Path=" urn:polimi:names:dbsp:1:attribute:objectProperty:isParentOf/
          urn:polimi:names:dbsp:1:attribute:dataProperty:hasAge "
          DataType=" http://www.w3. org/2001/XMLSchema#integer " /> </SubjectMatch>
      </Subject>
    </Subjects>
    <Resources>
      <Resource>
          <ResourceMatch<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
          document_108143</AttributeValue><ResourceAttributeDesignator AttributeId="
          urn:polimi:names:dbsp:1:attribute:id "
          DataType=" urn:polimi:names:dbsp:1:datatype:ontologyid "/> </ResourceMatch>
      </Resource>
    </Resources>
    <Actions>
      <Action>
          <ActionMatch MatchId=" urn:polimi:names:dbsp:1:function:ontologyidequal "><AttributeValue
          DataType=" urn:polimi:names:dbsp:1:datatype:ontologyid ">read</AttributeValue>
          MustBePresent=" true "
          MatchId=" urn:polimi:names:dbsp:1:function:ontologyidequal ">
          <ActionAttributeDesignator
          </ActionMatch>
      </Action>
    </Actions>
  </Target>
</Rule>
```

Figura 4.6: Esempio XACML congiunzione OR

Congiunzione AND

La situazione si complica quando soggetti multipli devono essere connessi con una congiunzione AND. I soggetti (o le risorse) non possono più essere considerati indipendenti l'uno dall'altro, ma devono essere uniti in un singolo soggetto (o risorsa) un esempio XACML di come appare questa regola prima della traduzione è visibile nella Figura 4.7. Questo processo genera delle complicazioni nel caso siano coinvolte regole di tipo 5 poiché il loro comportamento è completamente differente dalle altre, dobbiamo quindi distinguere tre diversi scenari.
In caso non sia presente nessuna regola di tipo 5 viene definita una nuova classe che rappresenta l'intersezione delle classi che identificano i soggetti, poi si prosegue con il solito algoritmo. Un esempio è rappresentato dal soggetto "ogni

medico che sia maschio” la nuova classe sarà quindi rappresentata dall’intersezione delle classi `Medic` e `Class_hasGender_M`.

```
<!-- Regola che permette ai medici maschi di legger i documenti di andrologia -->
<Rule RuleId="Rule8" Effect="Permit">
   <Target>
         <Subjects>
            <Subject>
                  <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:stringequal">
                  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">M</AttributeValue>
                  <SubjectAttributeDesignator AttributeId="
                  urn:polimi:names:dbsp:1:attribute:dataProperty:hasGender "
                  MustBePresent="true" DataType="http://www.w3.org/2001/XMLSchema#string"/>
                  </SubjectMatch>

                  <SubjectMatch MatchId=" urn:polimi:names:dbsp:1:function:ontologyidequal ">
                  <AttributeValue DataType=" urn:polimi:names:dbsp:1:datatype:ontologyid "> medic
                  </AttributeValue><SubjectAttributeDesignator AttributeId="
                  urn:polimi:names:dbsp:1:attribute:class "
                  MustBePresent=" true "
                  </SubjectMatch>
            </Subject>
         </Subjects>
         <Resources>
             <Resource>
                  DataType=" urn:polimi:names:dbsp:1:datatype:ontologyid "/>
                  <ResourceMatch MatchId=" urn:polimi:names:dbsp:1:function:ontologyidequal ">
                  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
                  AndrologyDocument</AttributeValue><ResourceAttributeDesignator AttributeId="
                  urn:polimi:names:dbsp:1:attribute:class "
                  DataType=" urn:polimi:names:dbsp:1:datatype:ontologyid "/> </ResourceMatch>
                  </Resource>
            </Resources>
         <Actions>
            <Action>
                  <ActionMatch MatchId=" urn:polimi:names:dbsp:1:function:ontologyidequal "><AttributeValue
                  DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
                  <ActionAttributeDesignator
                  </ActionMatch>
            </Action>
         </Actions>
    </Target>
</Rule>
```

Figura 4.7: Esempio XACML congiunzione AND

La situazione si complica quando è presente una regola di tipo 5, questo succede ad esempio se vogliamo esprimere una regola che permette ad ogni medico maschio di scrivere ogni `MedicalRecord` del reparto in cui lui lavora. Lo step iniziale è identico a quello precedente: viene creata una classe intersezione di `Medic` e `Class_hasGender_M`, si crea poi la proprietà riflessiva `identityOn_Class_Medic_and_hasgender_M` e infine si tratta la rule come una regola di Tipo 5, vale a dire con catene di proprietà impostate come sotto proprietà dell’azione da eseguire. Discorso analogo anche se un po’ più lungo e complesso avviene per la congiunzione AND di più modelli di rule 5.

4.1.4 Algoritmo di Traduzione

Nella sezione 4.1.2 e 4.1.3 abbiamo fornito una descrizione di come le singole regole vengano tradotte nella corrispondente rappresentazione OWL. Ora forniremo una overview di tutto il processo di traduzione da un pnto di vista più generale. La traduzione è effettuata da un programma JAVA, sviluppato usando OWL API [13]. È formalmente descritto in pseudo-codice nella Figura 4.8. Il programma disegnato per essere completamente dominio indipendente, riceve in ingresso il file XACML, contenente le politiche di dominio, e la ABOX di dominio dell'ontologia, e produce in output la Policy Ontology. L'ABOX di dominio contiene tutte le entità e le relazioni definite nell'ontologia di dominio più un arbitrario numero di individui generati automaticamente per il nostro esempio. La Figura 4.9 mostra il class diagram del programma, quest'ultimo possiamo pensarlo come suddiviso in due parti principali: la parte di Parsing e la parte di Traduzione. Il **Parsing** consiste nel leggere il file XACML e creare una rappresentazione JAVA delle politiche e delle regole che descrive. Viene creato un oggetto `Parser` e invocato il suo metodo `readFile()` che legge il file e lancia `parsePolicySet()`. Questo metodo, attraverso il lancio di altri metodi, crea una rappresentazione JAVA delle politiche. Ogni tag XACML ha una corrispondente

```
O = ontologia delle Policy caricata
R = set di rules
for each <rule: r> ∈ R
do {
        TS = 0 (set di proprietà, per la rule r che collega i soggetti all'oggetto 'o')
        S = set di soggetti della rule r
        for each <subject: s> ∈ S
        do {
                TM = 0 (set di classi subjectMatch per il soggetto s)
                M = set di subjectMatch del soggetto s
                for each <subjectMatch: m> ∈ M
                do {
                        crea o recupera la OWLClass c per m
                        TM ⇐ TM ∪ {c}
                }
                crea la objectProperty p per connettere tutti gli individui della
                        classe formata dalla congiunzione degli elementi in TM
                TS ⇐ TS ∪ {p}
        }
        TD = 0 (set di proprietà, per la rule r che collega le risorse all'oggetto 'o')
        D = set di risorse della rule r
        for each <resource: d> ∈ D
        do {
                TM = 0 (set di classi resourceMatch per la risorsa d)
                M = set di resourceMatch della risorsa d
                for each < resourceMatch: m> ∈ M
                do {
                        crea o recupera la OWLClass c per m
                        TM ⇐ TM ∪ {c}
                }
                crea la objectProperty p per connettere tutti gli individui della
                        classe formata dalla congiunzione degli elementi in TM
                Td ⇐ Td ∪ {p}
        }
        a: objetcProperty per l'azione specificata nella rule r
        for each <objectProperty: ts> ∈ TS
        do {
                for each <objectProperty: td> ∈ TD
                do {
                        axiom ax = {ts o inverseOf(td) ⊆ a}
                        O ⇐ O ∪ {ax}
                }
        }
}
```

Figura 4.8: Algoritmo traduzione delle policy

classe Java: un `PolicySet` contiene una lista di elementi `policy` e ognuno di loro contiene a sua volta una lista di oggetti `rule`, e così via.
Una volta che il parsing è completo il sistema avvia la **Traduzione**. Prima di tutto viene creato un oggetto `PolicyCreator` e la rappresentazione JAVA del `PolicySet`,

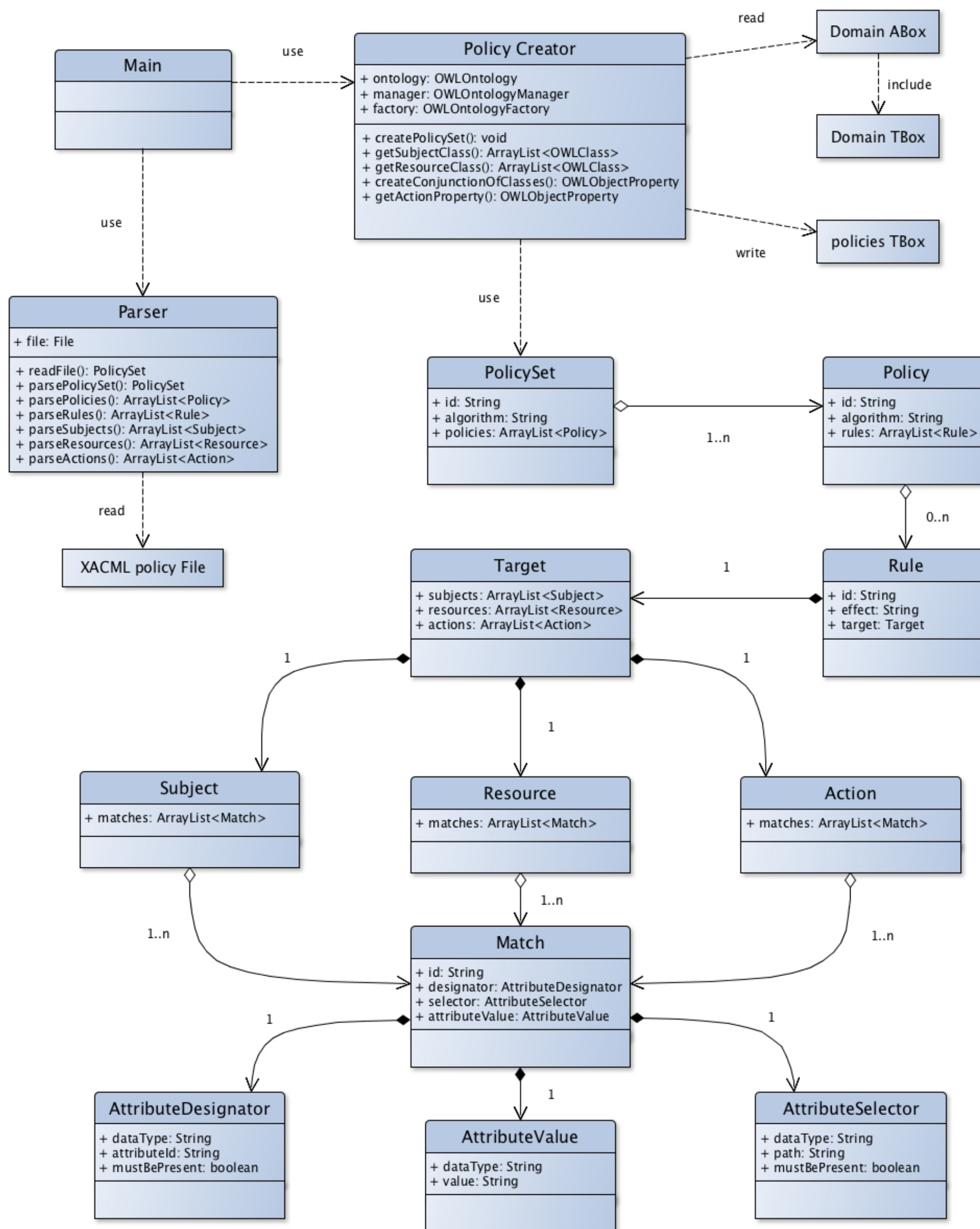

Figura 4.9: Class Diagram del programma Java di traduzione

ritornata dal `Parser` è presa come input dal suo metodo `createPolicySet()`. Quest'ultimo applica, per ogni rule, la procedura di traduzione descritta nelle sezioni 4.2.2 e 4.2.3. Il risultato è infine salvato nell'ontologia di destinazione.

4.2 Explanation

Nel seguente paragrafo parleremo di come il sistema si comporta quando un utente richiede delle spiegazioni (o explanation) su una decisione di accesso presa. In particolare la sezione 4.2.1 si occuperà di mostrare i possibili casi in cui il sistema si trova all'inizio del processo di elaborazione, la 4.2.2 di spiegare come effettivamente il sistema elabora e traduce le spiegazioni e la 4.2.3 illustrerà le modifiche che vengono apportate alle spiegazioni per raffinarle ulteriormente e renderle il più comprensibile possibile per l'utente.

4.2.1 Casi iniziali

La prima cosa importante da segnalare è che il sistema non è sempre in grado fornire spiegazioni dettagliate che motivino le sue decisioni di accesso. Possiamo infatti identificare quattro casi, in tre di questi il Reasoner presente nel Policy Decision Point (PDP) riesce con i dati dell'ontologia e le politiche di accesso a motivare la sua decisione di concedere o negare un accesso, nel quarto caso invece il sistema non dispone di sufficiente informazioni, i quattro casi sono i seguenti:

- Caso 1: è presente solo una regola che garantisce all'utente il permesso di accedere alla risorsa, il risultato sarà accesso consentito e il sistema sarà in grado di elaborare le spiegazioni.
- Caso 2: è presente solo una regola che nega all'utente il permesso di accedere alla risorsa, il risultato sarà accesso negato e il sistema sarà in grado di elaborare le spiegazioni.
- Caso 3: sono prensenti entrambe le regole una che garantisce all'utente il permesso di accedere alla risorsa e l'altra che lo nega, il risultato sarà accesso negato, perchè come visto nella sezine 4.1.1 l'algoritmo adottato è *deny-override*, e come nel caso 2 il sistema sarà in grado di elaborare le spiegazioni.
- Caso 4: non è presente nè una regola che permetta l'accesso nè una che lo nega, in questo caso come visto nella sezione 4.1.1 il sistema nega l'accesso grazie all'ultima regola del file delle politiche, ma non è in grado di elaborare e restiruire all'utente informazioni utili per spiegare il motivo per cui non abbia potuto accedere alla risorsa. Si limiterà quindi a stampare sul monitor "Permessi insufficienti".

Caso	Rule che permette l'accesso	Rule che vieta l'accesso
1	X	
2		X
3	X	X
4		

Figura 4.10: Casi delle explanations

I quattro casi sono espressi in modo grafico nella Figura 4.10, l'ultima riga, quella del caso 4, è evidenziata in rosso.

4.2.2 Elaborazione

Il processo di elaborazione inizia con la richiesta al Reasoner di fornire gli assiomi dell'ontologia che ha utilizzato per prendere una decisione di accesso. Come precedentemente detto il Reasoner utilizzato per fornire le spiegazioni è Pellet [10], dal momento che tra quelli più noti analizzati è l'unico che fornice delle spiegazioni. Attraverso la chiamata del metodo JAVA `getEntailmentExplanation()` della classe PelletExplanation il sistema ottiene gli assiomi usati dal reasoner, trattandosi si assiomi sono ovviamente espressi in linguaggio logico e non tutti sono utili e sensati per l'utente, il sistema genera quindi le explanation seguendo il seguente algoritmo:

1. Eliminare tutti gli assiomi che contengono riferimenti alla Classe O. Essi sono infatti utili solo per la costruzione della relazione universale tra i soggetti e le risorse, non hanno utilità per le explanations.
2. Identificare l'assioma che pone la catena soggetti-risorse come sottoproprietà dell'azione da eseguire. Questo assioma rappresenta la regola che il sistema ha usato per decidere se concedere o no l'accesso alla risorsa quindi, una volta trovato, è il primo ad essere tradotto e stampato.
3. Identificare e tradurre tutti gli altri assiomi, che si dividono in tre gruppi: quelli che decretano l'appartenenza di un individuo ad una classe (*assioma di appartenenza*), quelli che esprimono un attributo di un individuo (*assioma dataproperty)* e quelli che esprimono la relazione di un individuo con altri individui (*assioma objectproperty)*.

L'algoritmo è illustrato graficamente nella Figura 4.11.

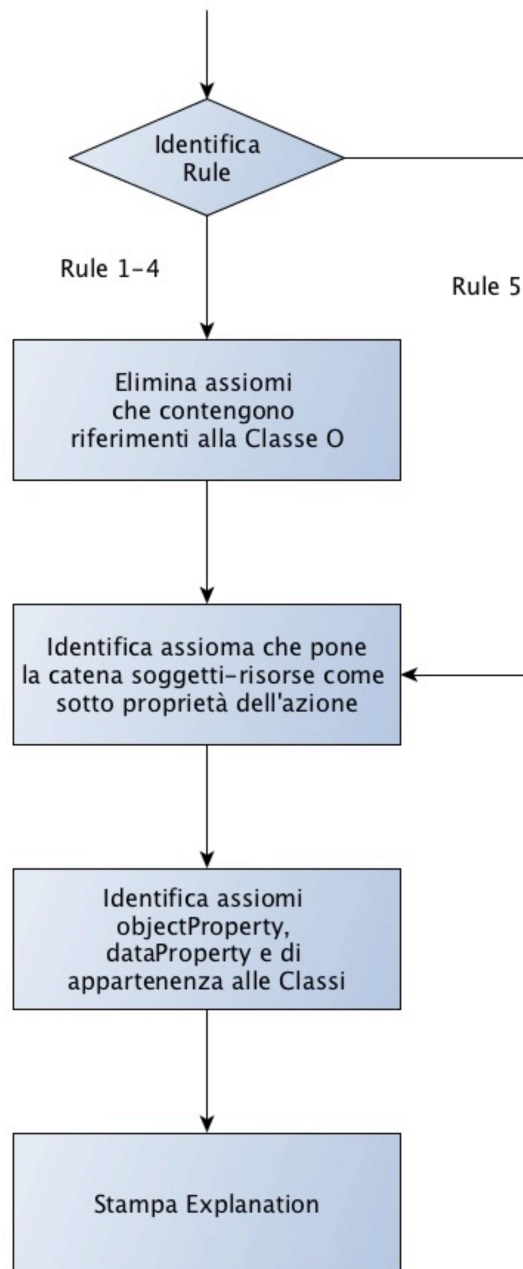

Figura 4.11: Algoritmo elaborazione Explanations

Vediamo ora caso per caso la richiesta di spiegazioni per ogni tipo di regola.

Rule tipo 1

Explanation for: marta_gatti canNotWrite exam_000439

1) Class_O **EquivalentTo** {O}
2) Class_marta_gatti **EquivalentTo** everyClass_marta_gatti **exactly** 1 Class_O
3) exam_000439 **Type** exam
4) exam **EquivalentTo** everyexam **exactly** 1 Class_O
5) canNotRead **SubPropertyOf** canNotWrite
6) everyClass_marta_gatti **o** **inverse** (everyexam) **SubPropertyOf** canNotRead
7) marta_gatti **Type** Class_marta_gatti

Figura 4.12: explanations per rule di tipo 1

L'utente Maria Gatti chiede spiegazioni sul perché non abbia potuto accedere in lettura al documento exam n° 000439. Gli assiomi che il sistema riceve sono quelli presenti nella Figura 4.12.
Per prima cosa il sistema elimina tutti quelli che contengono riferimenti alla Classe O, successivamente identifica l'assioma che rappresenta la regola usata per prendere la decisione su questa richiesta di accesso, in questo caso:

```
everyClass_marta_gatti o inverse everyexam subPropertyOf
canNotRead
```

L'assioma contiene in ordine: il soggetto (`everyClass_marta_gatti`), le risorse (`everymedicalexam`) e l'azione `canNotRead`, con una minima formattazione è quindi possibile ricostruire la regola: *marta_gatti can not read exam.*
L'altro assioma utile è un assioma di appartenenza che dichiara che il documento exam n° 000439 appartiene alla classe exam.
La explanation che il sistema ha elaborato per questa richiesta è quindi:

```
- marta_gatti can not read exam
- exam_000439 belongs to exam
```

Rule tipo 2

L'utente Federico Messina chiede spiegazioni sul perché lui abbia permessi di scrittura sul medicalRegulationDocument n° 577594. Gli assiomi che il sistema riceve sono quelli presenti nella Figura 4.13.
Per prima cosa vengono eliminati tutti quelli che contengono riferimenti alla Classe O, successivamente il sistema identifica l'assioma che rappresenta la regola usata per prendere la decisione su questa richiesta di permesso, in questo caso:

```
everymedicalConsultant o inverse(everymedicalRegulationDocument)
subPropertyOf canWrite
```

Explanation for: federico_messina canWrite medicalRegulationDocument_577594

Class_O **EquivalentTo** {O}
everymedicalConsultant **o inverse** (everymedicalRegulationDocument) **SubPropertyOf** canWrite
medicalRegulationDocument_577594 **Type** medicalRegulationDocument
medicalConsultant **EquivalentTo** everymedicalConsultant **exactly** 1 Class_O
medicalRegulationDocument **EquivalentTo** everymedicalRegulationDocument **exactly** 1 Class_O
federico_messina **Type** medicalConsultant

Figura 4.13: explanations per rule di tipo 2

L'assioma contiene in ordine: i soggetti (`everymedicalConsultant`), le risorse (`everymedicalRegulationDocument`) e l'azione canWrite, con una minima formattazione è quindi possibile ricostruire la regola: *medicalConsultant can write medicalRegulationDocument.*
Gli ultimi assiomi da analizzare sono due assiomi di appartenenza che dichiarano l'appartenenza di soggetti e risorse alle rispettive classi: Federico Messina appartiene alla classe medicalConsultant e la risorsa Medical Regulation Document n° 577594 appartiene alla classe medicalRegulationDocument.
La explanation che il sistema ha elaborato per questa richiesta è quindi:

```
- medicalConsultant can write medicalRegulationDocument
- medicalRegulationDocument_577594 belongs to class
      medicalRegulationDocument
- federico_messina belongs to class medicalConsultant
```

Rule tipo 3

L'utente Giulia Ricci chiede spiegazioni sul perché lei abbia permessi di lettura sul documento n° 196054. Gli assiomi che il sistema riceve sono quelli presenti nella Figura 4.14.
Come sempre per prima cosa viene eliminato tutto ciò che contiene riferimenti alla Classe O, successivamente il sistema identifica l'assioma che rappresenta la regola usata per prendere la decisione su questa richiesta, in questo caso:

```
everyClass_hasSex_string-equal_F o inverse
everyClass_document_196054 subPropertyOf canRead
```

Con questo assioma è ora possibile ricostruire la regola e visto che soggetti e risorse occupano sempre la stessa posizione, formattando il testo otteniamo: *hasSex_string-equal_F can read document_196054.*
L'ultimo assioma rimasto da analizzare è un assioma dataproperty `giulia_ricci hasSex "F"^^string` che tradotto diventa*: giulia_ricci has sex F*, non è necessario un assioma di appartenenza perché l'assioma dataoproperty descrivendo un attributo dell'utente Giulia Ricci, *has sex F*, ne sancisce anche

Explanation for: giulia_ricci canRead document_196054

Class_O **EquivalentTo** {O}
document_196054 **Type** Class_document_196054
giulia_ricci hasSex "F"^^string
Class_hasSex_string-equal_F **EquivalentTo** hasSex **value** "F"^^string
Class_hasSex_string-equal_F **EquivalentTo** everyClass_hasSex_string-equal_F **exactly** 1 Class_O
Class_document_196054 **EquivalentTo** everyClass_document_196054 **exactly** 1 Class_O
everyClass_hasSex_string-equal_F **o** **inverse** (everyClass_document_196054) **SubPropertyOf** canRead

Figura 4.14: explanations per rule di tipo 3

l'appartenenza alla classe *hasSex_string-equal_F.* Anche per la risorsa non è necessario assioma di appartenenza dal momento che la regola fa riferimento a una singola risorse, espressa nella regola come individuo, e non a una classe di risorse. La explanation che il sistema ha elaborato per questa richiesta è quindi:

```
- hasSex_string-equal_F can read document_196054
- giulia_ricci has sex F
```

Come si può vedere in questo caso la forma di espressione della explanation è ancora piuttosto grezza, per questo dopo la fase di elaborazione è stata introdotta una fase di raffinamento delle spiegazioni, che sarà oggetto del paragrafo seguente. Teniamo perciò a mente che quella riportata sopra è ancora una versione di passaggio della explanation e nella maggior parte dei casi non sarà mostrata all'utente.

Rule tipo 4

L'utente Lorenzo Sala chiede spiegazioni sul perché abbia permessi di lettura sul documento n° 305871. Gli assiomi che il sistema riceve sono quelli presenti nella Figura 4.15.
Per prima cosa il sistema elimina gli assiomi che contengono riferimenti alla Classe O, successivamente identifica l'assioma che rappresenta la regola usata per prendere la decisione su questa richiesta, in questo caso:

```
everyClass_isTutorOf_hasAge_lower-than_18 o inverse
everyClass_document_305871 subPropertyOf canRead
```

L'assioma contiene in ordine: i soggetti (`isTutorOf_hasAge_lower-than_18`), la risorse (`document_305871`) e l'azione `canRead`, formattando è quindi possibile ricostruire la regola: *isTutorOf_hasAge_lower-than_18 can read document_305871.*
Gli altri due assiomi, che il sistema identifica, e che sono utili per costruire la nostra spiegazione, sono: un assioma objectproperty che mette in relazione due individui e un assioma dataproperty che descrive un attributo di uno di questi due individui. L'assioma objectproperty è: `lorenzo_sala isTutorOf gabriele_rossetti` che collega il soggetto Lorenzo Sala all'individuo Gabriele Rossetti con la objectProperty `isTutorOf.` Mentre l'assioma dataproperty è:

```
Explanation for: lorenzo_sala canRead document_305871
Class_O EquivalentTo {O}
Class_isTutorOf_hasAge_lower-than_18 EquivalentTo isTutorOf some Class_hasAge_lower-than_18
lorenzo_sala isTutorOf gabriele_rossetti
gabriele_rossetti hasAge 2
Class_isTutorOf_hasAge_lower-than_18 EquivalentTo everyClass_isTutorOf_hasAge_lower-than_18 exactly 1 Class_O
document_305871 Type Class_document_305871
Class_document_305871 EquivalentTo everyClass_document_305871 exactly 1 Class_O
everyClass_isTutorOf_hasAge_lower-than_18 o inverse (everyClass_document_305871) SubPropertyOf canRead
Class_hasAge_lower-than_18 EquivalentTo hasAge some integer[< 18]
```

Figura 4.15: explanations per rule di tipo 4

`gabriele_rossetti hasAge 2` che ci comunica che l'attributo `hasAge` dell'individuo Gabriele Rossetti ha valore uguale "2".
La explanation che il sistema ha elaborato per questa richiesta è quindi:

```
- isTutorOf_hasAge_lower-than_18 can read document_305871
- lorenzo_sala isTutorOf gabriele_rossetti
- gabriele_rossetti has age 2
```

Anche in questo caso, come nel precedente, è necessaria raffinare il formato della spiegazione prima di presentarla all'utente.

Rule tipo 5

L'utente Simone Esposito chiede spiegazioni sul perché non abbia potuto accedere di lettura alla risorsa medicalRecord n° 591420. Gli assiomi che il sistema riceve sono quelli presenti nella Figura 4.16.
In questo caso non abbiamo assiomi che contengono riferimenti alla Classe O da eliminare, quindi il sistema identifica subito quella che la regola usata per prendere la decisione su questa richiesta di accesso, in questo caso:

```
inverseOf_hosts o ownsRecord o identityOn_medicalRecord
subPropertyOf canNotRead
```

Il procedimento in questo caso cambia leggermente, come si può vedere infatti il pattern della regola è cambiato, i soggetti e le risorse non sono più identificati come individui o classi di individui ma con una serie di proprietà, rimane invece invariata l'azione. I soggetti e le risorse sono quindi identificati come quegli individui che rispettano la catena di proprietà elencata, in questo caso gli ospitati nel dipartimento (soggetti) che possiede determinati medical record (risorse).
Formattando un po' la regola utilizzando anche i domini e i codomini delle proprietà elencate otteniamo: *inverseOf_hosts ward that ownsRecord some medicalRecord can not read those medicalRecord*. Dove *inverseOf_hosts* significa l'inverso della proprietà hosts che, dal punto di vista del linguaggio, vuol dire passare dalla forma attiva alla forma passiva.

Explanation for: simone_esposito canNotRead medicalRecord_591420

1) oncology owns medicalFolder_695335
2) inverseOf_hosts **InverseOf** hosts
3) medicalRecord_591420 containedInMedicalFolder medicalFolder_695335
4) medicalRecord **EquivalentTo** identityOn_medicalRecord **some Self**
5) inverseOf_hosts **o** ownsRecord **o** identityOn_medicalRecord **SubPropertyOf** canNotRead
6) owns **o inverse** (containedInMedicalFolder) **SubPropertyOf** ownsRecord
7) oncology hosts simone_esposito
8) containedInMedicalFolder **Domain** medicalRecord

Figura 4.16: explanations per rule di tipo 5

Gli altri assiomi che il sistema identifica sono di tipo objectproperty e fanno appunto riferimento alle objectProperty che compongono il primo assioma. Quest'ultimi servono a costruire i collegamenti che portano i soggetti alle risorse, dopo averli un minimo formattati la explanation che il sistema ha elaborato per questa richiesta è:

```
- inverseOf_hosts ward that ownsRecord some medicalRecord can not
      read those medicalRecord
- oncology owns medicalFolder_695335
- medicalRecord_591420 containedInMedicalFolder
      medicalFolder_695335
- oncology hosts simone_esposito
```

Anche in quest'ultimo caso la raffinazione della spiegazione la renderà più comprensibile all'utente.

4.2.3 Revisione

Come anticipato nella sezione precedente in alcuni casi il sistema non è ancora in grado di dissociarsi completamente dal linguaggio logico degli assiomi della ontologia OWL e di presentare le spiegazioni in linguaggio umano. Ecco perché era necessaria una fase di revisione delle explanation per renderle più comprensibili all'utente. Sono stati adottati due metodi: il primo che agisce a livello più basso fa uso delle Annotation OWL fornite dalle omonime API. Il secondo metodo invece agisce più ad alto livello e colma le lacune del primo.

Con le Annotation è possibile assegnare una etichetta (o label) di tipo String a ogni individuo o classe della nostra ontologia. I nomi degli utenti inseriti nel sistema nel formato "nome_cognome" avranno come label "Nome Cognome" le classi "hasSex_string-equal_F" avranno come label "Women" e così via. In questo modo al momento della stampa della explanation il sistema non stamperà il nome dell'individuo o della classe ma la sua etichetta.

Il secondo metodo funziona invece con la sostituzione di precisi pattern di stringhe. Esso utilizza un file chiamato "Format Rule", editabile dagli amministratori del sistema, che ha il seguente aspetto:
pattern_di_parole_da_sostituire, pattern_di_parole_sostitute
Prima di stampare a video la spiegazione il sistema richiama un metodo che, con l'ausilio del Policy Constructs controlla se nell'explanation è presene uno dei pattern di sostituzione previsti e nel caso effettua lo scambio. In questo modo il costrutto "inverseOf_hosts ward" può facilmente diventare "People hosted in ward".
Visti i due metodi vediamo ora di capire perché è necessario l'utilizzo di entrambi. Il primo ha decisamente molti più vantaggi del secondo: lavora più a basso livello, è più elegante ed è context-free, vale a dire che non dipende dal contesto, la sostituzione è effettuata in tutti i casi nello stesso modo. Proprio quest'ultimo pregio è anche il suo svantaggio: ci possono essere infatti situazioni in cui la sostituzione dipenda dal contesto, una particolare objectProperty potrebbe essere tradotta diversamente in base alla parola che la segue. Proprio per questo è stato inserito il secondo metodo, con l'ausilio del file infatti è possibile mappare non solo una classe o una proprietà ma interi pezzi di frase che possono poi essere sostituiti con un'alternativa migliore. Le Annotation sono quindi perfette per i nomi degli utenti e delle classi, mentre la sostituzione con file è stata usata soprattutto per sostituire le proprietà soprattutto nelle regole di tipo 5 in cui le objectProperty possono incatenarsi tra di loro in numerosi modi diversi.

Vediamo ora come si presentano le spiegazioni presentate nella sezione 4.2.2 dopo la fase di revisione:

1) Marta Gatti richiesta negata accesso in lettura al file exam 000439

```
- Marta Gatti can not read Exams
- Exam 000439 belongs to Exams
```

2) Federico Messina richiesta acconsentita accesso in scrittura al file Medical Regulation Document

```
- Medical Consultant can write Medical Regulation Document
- Medical Regulation Document 577594 belongs to class
      Medical Regulation Document
- Federico Messina belongs to class Medical Consultant
```

3) Giulia Ricci richiesta acconsentita accesso in lettura al file Document 196054

```
- Women can read Document 196054
- Giulia Ricci is a woman
```

4) Lorenzo Sala richiesta acconsentita in scrittura al file Document 305871

```
- Tutors of minors can read Document 305871
- Lorenzo Sala is tutor of Gabriele Rossetti
- Gabriele Rossetti has age 2
```

5) Simone Esposito richiesta negata accesso in lettura al file Medical Record 591420

```
- People hosted in ward that owns some Medical Record
      can not read those Medical Record
- Oncology owns Medical Folder 695335
- Medical Record 591420 is contained in Medical Folder
      695335
- Oncology hosts Simone Esposito
```

Capitolo 5

Performance

Come descritto nei capitoli precedenti il prototipo del framework sviluppato esegue principalmente tre processi: **Policy Translation** (traduzione di politiche), **Policy Decision** (decisione di politiche), **Access Explanation** (spiegazione di accesso). Queste tre parti sono completamente disgiunte, quindi le performance andranno valutate distintamente per ogni processo. La sezione 5.1 si occuperà di esporre le sperimentazioni effettuate e il loro esito per due dei processi sopra elencati, le performance della parte di decisione sono già state trattate nell'elaborato "OWL-Based Representation and Enforcement of Data Access Policies".

5.1 Valutazione delle performance

Uno dei più importanti requisiti per un sistema di controllo di accessi è un buon tempo di risposta. Il processo di decisione deve essere praticamente immediato, quindi un buon tempo di risposta dovrebbe stare sotto al secondo. Questo vale soprattutto per la parte di Decisione di Accesso, un discorso diverso può essere invece fatto per la Traduzione delle politiche e per la Spiegazione di accesso. Questi due processi partono infatti solo quando viene modificato il file delle politiche XACML o quando un utente richiede esplicitamente spiegazioni riguardo una decisione di accesso, non necessitano quindi di un tempo di risposta immediato.
La sezione 5.1.1 si occuperà di mostrare i tempi di risposta del sistema durante il processo di traduzione delle politiche, mentre la sezione 5.1.2 farà lo stesso per il processo di spiegazione di accesso. Tutti i test sono stati eseguiti usando un PC con processore Intel Core 2 Duo 2.4 GHz e 4 GB di RAM 1067 MHz DDR3.

5.1.1 Traduzione di policy

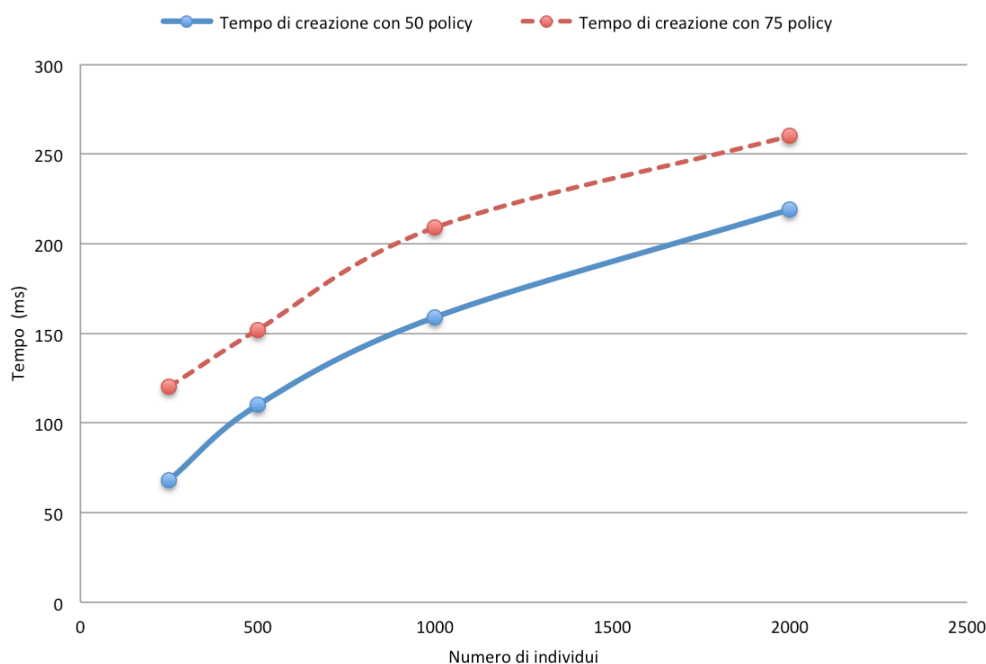

Figura 5.1: Tempo di creazione di policy con numero crescente di individui.

Durante lo sviluppo tutti i test sono stati effettuati su un piccolo dominio formato da una ABox di 250 individui e un file di politiche XACML contenente solo 5 politiche (o meglio una singola politica con 5 regole). Al fine di avere una stima più reale delle performace del sistema abbiamo dovuto creare domini più grandi e abbiamo analizzato come evolve il tempo di risposta incrementando il numero di individui e il numero di politiche. Tutti gli esperimenti descritti in questa sezione fanno riferimento al processo di traduzione descritto nella Sezione 4.1.4.
Le variabili coinvolte nel nostro sistema sono due: il numero di politiche e il numero di individui, sono state quindi effettuate due misurazioni: la prima tenendo fisso il numero di politiche e variando il numero di individui, mentre la seconda tenendo fisso il numero di individui e variando il numero di politiche.
La Figura 5.1 mostra il primo caso, ossia: la variazione del tempo di creazione delle politiche in formato OWL al variare del numero di individui. Sono stati presi in considerazioni due casi medi, uno con 50 e l'altro con 75 policy, mentre il numero di individui varia da 250 a 2000.
In entrambi i casi l'evoluzione è logaritmica andando da 68 ms a 219 ms nel primo caso, e da 120 ms a 260 ms nel secondo. A questi tempi va poi sommato il tempo di parsing del file XACML, che non dipende dal numero di individui, e in media è di 800 ms con 50 policy e di 1200 ms con 75 policy.

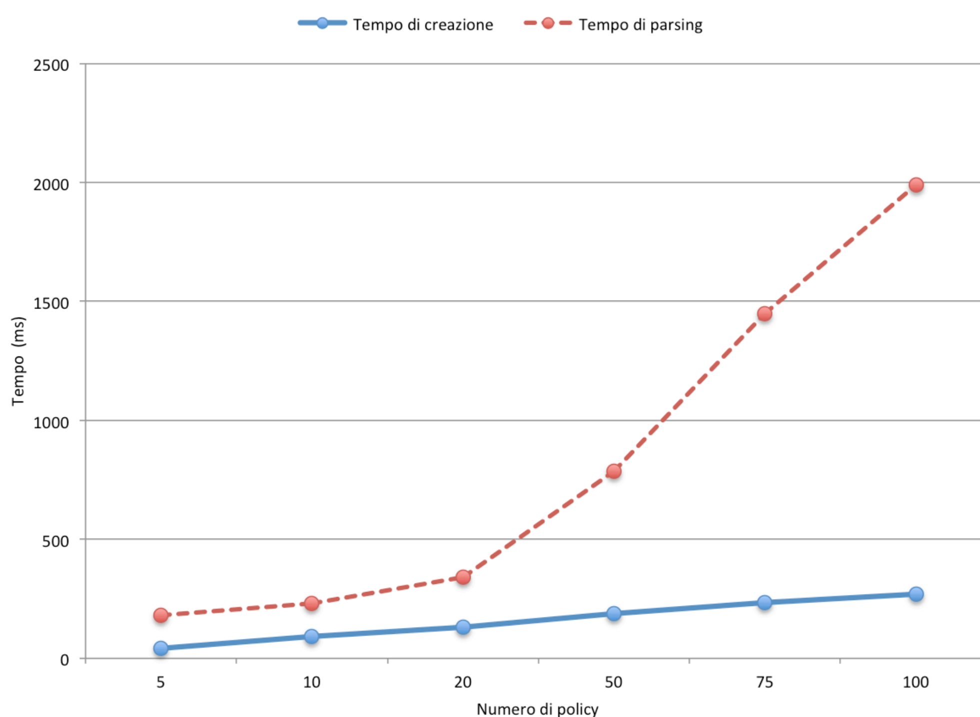

Figura 5.2: Tempo di creazione di policy con numero crescente di policy.

La figura 5.2 mostra invece il secondo caso: la variazione del tempo di traduzione al variare del numero di politiche, il numero di individui è tenuto fisso a 1000 (che può essere considerato un valore medio realistico) mentre il numero di politiche varia da 5 a 100.
L'evoluzione del tempo di creazione è sempre logaritmica, andando da 41 ms con 5 policy a 271 ms con 100 policy. Anche in questo secondo caso per ottenere il tempo totale di traduzione dobbiamo sommare questi tempi al tempo di parsing, che ovviamente dipende fortemente dal numero di politiche, e in caso di domini con tante regole influisce molto sul tempo totale. Come si può vedere dai grafici comunque anche nel caso peggiore la somma del tempo di parsing e del tempo di creazione di politiche supera di poco i 2 secondi, risultato più che accettabile. Possiamo quindi considerare le performance del processo di traduzione soddisfacenti.

5.1.2 Spiegazioni di accesso

Quando un utente chiede spiegazioni per una decisione di accesso, il framework lancia il reasoner che restituisce tutti gli assiomi usati per prendere la decisione e genera le spiegazioni. E' immediato capire come in questo caso più che nel processo di traduzione un alto numero di politiche o di individui peggiori

considerevolmente le performance del sistema: alti numeri di individui e politiche

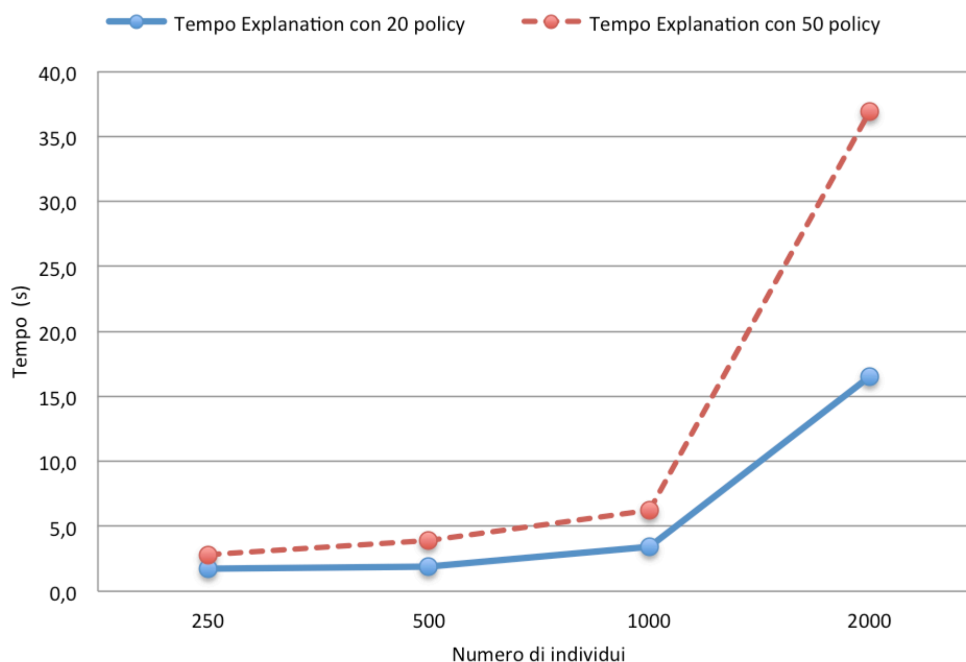

Figura 5.3: Tempo per Explanation con numero crescente di individui.

implicano un aumento esponenziale di Object Property (relazioni) da analizzare.
Anche in questo caso abbiamo condotto due esperimenti: la variazione del tempo per ottenere le spiegazioni al variare del numero degli individui, tenendo fisso il numero di politiche, e la variazione del tempo di risposta al variare del numero di politiche, tenendo fisso il numero di individui. I risultati degli esperimenti hanno confermato quello che ci aspettavamo.
Nella Fugura 5.3 sono mostrati i tempi di risposta del sistema nel primo dei due casi: gli individui variano da 250 a 2000, con 20 policy i tempi vanno da 1,7 a 16,5 secondi, mentre con 50 vanno da 2,8 a 36,9 secondi.
Nella figura 5.4 è invece mostrata la variazione del tempo di risposta al variare del numero di policy mantenendo il numero di individui fisso a 1000: con 5 policy la risposta è di 1,6 secondi mentre con 100 e di 128,6 secondi.
In entrambi i casi possiamo notare curve con andamento esponenziale, all'aumentare sia del numero di individui che del numero di politiche.
Questi risultati non sono accettabili: con valori alti, ma comunque raggiungibili in un contesto reale, il tempo di risposta del sistema è eccessivo. Bisogna però considerare che i test sono stati effettuati su un laptop con hardware ormai obsoleto, su un elaboratore moderno il sistema potrebbe raggiungere

performance accettabili nell'ordine di grandezza di un paio di secondi al massimo.

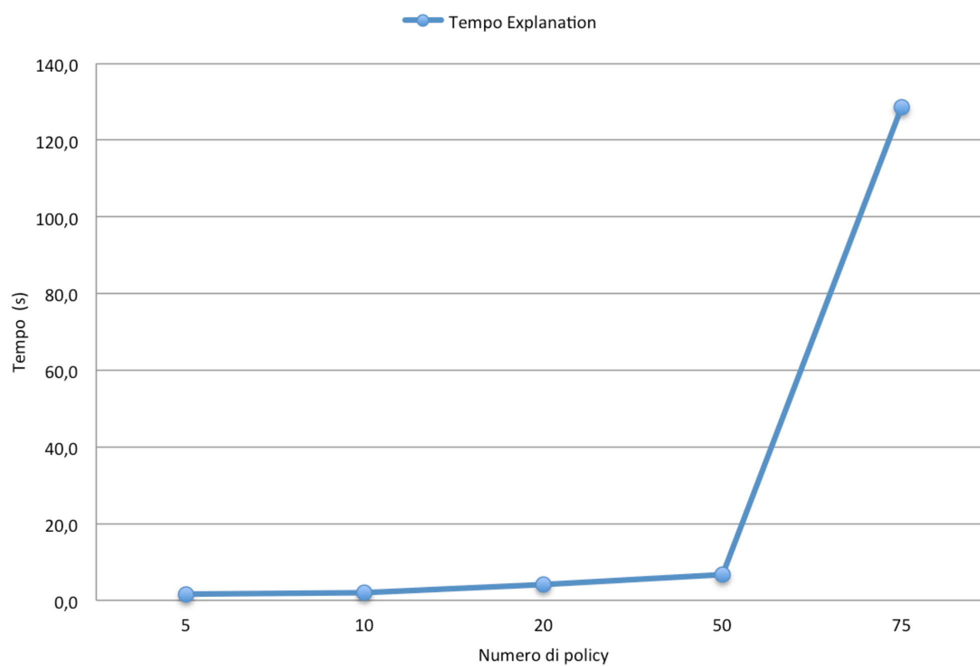

Figura 5.4: Tempo per Explanation con numero crescente di policy.

Capitolo 6

Conclusioni

Lo scopo di questa tesi è stato quello di definire e implementare un framework per esprimere e far rispettare politiche di accesso, che sia anche in grado di fornire delle spiegazioni su i ragionamenti compiuti, funzione che al momento non esiste in letteratura.
Il risulatato è stato ottenuto combinando l'architettura e il linguaggio di politiche XACML con i benefici delle ontologie OWL e delle tecnologie di Reasoning. Lo scopo di un architettura di questo genere è quello di ottenere i massimi benefici in termini di capacità espressiva delle politiche, scalabilità, riusabilità e armonizzazione delle politiche.
L'arcitettura è stata formalizzata, come descritto nella Sezione 3.X, partendo dagli standard dell'architettura XACML e estendendola al fine di renderla adatta alla nostra soluzione. Nonostante tutti i mduli siano stati concettualmenti pensati e disegnati, non tutti sono stati attualmente implementati. Per cominciare ci siamo concentrati sullo sviluppo del PAP, sviluppando un algoritmo che traducesse correttamente le politiche XACML in una TBox OWL, seguendo una procedura che permettese al Reasoner di estrarre la conoscenza usata nei suoi processi, per fornire all'utente le spiegazioni richieste. Successivamente ci siamo dedicati al PDP il modulo che prende le decisioni sulle richieste di accesso e fornisce le Explanation. È presente anche il modulo PIP che include una rappresentazione OWL di un esempio di dominio in termini di ABox e TBox. Mancano invece le implementazioni di PEP e Context Handler come anche l'interconnessione tra i vari moduli.
I risultati raggiunti da questo framework lo rendono unico tra quelli che gestiscono l'accesso ai dati tramite politiche, è infatti il primo tenta di fornire spiegazioni uman friendly all'utente per le decisioni di accesso che ha preso, introducendo una parte di Artificial Intelligence nel layer che si posiziona sopra alle strutture dati.
Importanti risultati sono stati anche ottenuti in termini di espressività delle politiche: sono state introdotte rule di Tipo 5, come destritto nella sezione 4.2, introducendo la possibilità di controllare gli accessi basandosi sulle relazioni che connettono il richiedente alla risorsa richiesta. La nostra versione estesa dello XACML, grazie alla sua flessibilità, ci permette di esprimere un insieme molto

vario di tipi di politiche, che sembrano coprire una grande varietà di concetti che un amministratore potrebbe voler esprimere. La versione modificata dello XACML si combina perfettamente con l'algoritmo di traduzione definito risultando universale e efficiente dal punto di vista dei tempi. Il processo di richiesta delle spiegazioni invece pur funzionando correttamente non si è dimostrato all'altezza del sistema dal punto di vista delle performance, come descritto nella sezione 5.1.

6.1 Sviluppi futuri

Il lavoro presentato lascia qualche punto aperto per futuri sviluppi che potranno migliorare o estendere l'attuale implementazione. Per quanto riguarda le migliorie la prima situazione da affrontare è quella delle performance, una riduzione dei tempi di Reasoning per la generazione automatica delle spiegazioni renderebbe infatti il sistema utilizzabile al 100% in un contesto reale.
Un'altra modifica all'attuale implementazione potrebbe consistere nel rendere il sistema in grado di fornire spiegazioni per un accesso negato anche senza che sia presente una regola che vieta l'accesso alla risorsa richiesta. In questo caso il sistema dovrebbe trovare l'insieme più piccolo di condizioni o attributi che mancano all'utente affinchè gli venga concesso l'accesso alla risorsa.
La seconda categoria di sviluppi futuri consiste nell'implementazione dei moduli e delle caratteristiche che per il momento sono state definite al massimo solo da un punto di vista concettuale. Tra queste si annoverano l'imlementazione di PEP e Context Handler, come previsti nell'architettura, oltre allo sviluppo delle interconnessioni tra i vari moduli, al momento sono tutti indipendenti l'uno dall'altro e non sono provvisti di un reale sistema di comunicazione.
Infine l'ultimo aspetto da affrontare è quello di armonizzazione delle politiche, l'obiettivo è quello di trovare una procedura automatica per garantire la coerenza e la consistenza tra le differenti politiche di dominio.
Lo sviluppo di tutti questi moduli e funzioni porterebbe alla realizzazione di un completo e innovativo Framework per le Politiche di Accesso.

Bibliografia

[1] Oasis (2005). Oasis xacml version 2.0 specification. Online: http://docs.oasis-open. org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf

[2] M. Karusseit, T. Margaria and H. Willebrandt (2008), *Policy expression and checking in XACML, WS-Policies, and the jABC*, 1st ed., pp. 20-26.

[3] W3.org (2006), 'The Platform for Privacy Preferences 1.1 (P3P1.1) Specification'. Online: http://www.w3.org/TR/P3P11/

[4] Deransart.fr (1999), 'Prolog: The ISO Standard Documents'. Online: http://www.deransart.fr/prolog/docs.html.

[5] M. Krotzsch, F. Simancik and I. Horrocks (2013). A description logic primer. Ithaca: Cornell University Library. Online: http://arxiv.org/abs/1201.4089

[6] Paraboschi, Neri, Mutti, Psaila, Salvaneschi, Verdicchio and Basile (2013). 'D2.5 - IT Policy Metamodel and Language', *PoSecCo WP2, Business and IT level policies*. Online: http://cordis.europa.eu/docs/projects/cnect/9/257129/080/deliverables/001-D25ITPolicymetamodelandlanguagefinal.pdf

[7] Paraboschi and Neri (2013). 'D2.4 - Policy Harmonization and Reasoning', *PoSecCo WP2, Business and IT level policies*. Online: http://www.cspforum.eu/D2.4_Policy_Harmonization_and_reasoning.pdf

[8] Treccani.it (2015), 'Stakeholder'. Online: http://www.treccani.it/enciclopedia/stakeholder.

[9] M. Horridge and P.F. Patel-schneider (2008). Manchester syntax for owl 1.1. In OWLED 2008, 4th international workshop OWL: Experiences and Directions (2008) Live Extraction 1223. 2008. Online: http://ceur-ws.org/Vol-496/owled2008dc_paper_11.pdf

[10] GitHub (2008), 'Complexible/pellet'. Online: http://clarkparsia.com/pellet/.

[11] Posecco.eu (2013), 'Policy and Security Configuration Management (PoSecCo)'. Online: http://www.posecco.eu.

[12] Filippo Pellegrini, Fabio Marfia (2015), 'OWL-Based Representation and Enforcement of Data Access Policies', Laurea Magistrale, Politecnico di Milano. Online: https://www.politesi.polimi.it/handle/10589/107345

[13] Owlapi.sourceforge.net (2015), 'University of Manchester OWL API 4'. Online: http://owlapi.sourceforge.net.

[14] F. Marfia, M. Arrigoni Neri, F. Pellegrini and M. Colombetti (2015), 'An OWL-based XACML Policy Framework', *Scitepress.org*. Online: http://www.scitepress.org/portal/PublicationsDetail.aspx?ID=3t46QlWJe3A=&t=1.