Università di Pisa
Facoltà di Scienze
Matematiche, Fisiche e Naturali

Dipartimento di Informatica
Corso di Laurea in Informatica

Relazione di tirocinio

Generazione Automatica di Casi di Test per Sistemi di Controllo degli Accessi espressi tramite XACML

Tutore Interno:

Prof. Andrea Corradini

Candidato:

Said Daoudagh

Tutori Esterni:

Dott.ssa Eda Marchetti

Dott.ssa Francesca Lonetti

Anno Accademico 2009/2010

Indice

Elenco delle figure

Elenco delle tabelle

Elenco dei codici

Capitolo 1

Introduzione

Lo sviluppo tecnologico nel campo informatico e il crescente utilizzo di un ambiente distribuito come Internet da parte di numerosi utenti hanno portato alla necessità di proteggere le risorse e i servizi da un utilizzo improprio, malizioso e non autorizzato.

Molte realtà che risiedono su web migliorano il loro livello di sicurezza utilizzando i *sistemi di controllo degli accessi*. *XACML* (OASIS eXtensible Access Control Markup Language) è un linguaggio che permette di esprimere le *politiche* di controllo degli accessi ed è largamente utilizzato nei moderni sistemi di controllo degli accessi.

Data l'importanza del ruolo che ricopre, un sistema di controllo degli accessi subisce un processo di testing e verifica molto accurato.

XACML definisce un modello per le politiche ed un modello per le richieste. Il potenziale del modello delle politiche è stato ampiamente sfruttato per la generazione di casi di test, mentre non sono state ancora esplorate le potenzialità che il modello delle richieste può presentare.

La presente relazione documenta l'attività svolta durante il tirocinio, il cui scopo è realizzare un framework di una strategia che cerca di esplorare e sfruttare le potenzialità del modello delle richieste per la generazione automatica di casi di test per le politiche di controllo degli accessi.

Il tirocinio, con un carico di lavoro pari a 18 CFU, si è svolto presso il Software Engineering Laboratory (LabSE), un laboratorio di ricerca dell'ISTI (Istituto di Scienze e Tecnologie dell'Informazione "A. Faedo") appartenente al CNR (Consiglio Nazionale delle Ricerche).

Di seguito è riportato lo scopo del progetto di tirocinio ed è illustrato lo schema del presente elaborato.

1.1 Obiettivi del tirocinio

Il progetto di tirocinio ha come scopo la realizzazione di un'applicazione in grado di generare in modo automatico casi di test per sistemi di controllo

degli accessi a partire dalle politiche di controllo espresse tramite il linguaggio XACML [35].

La base di partenza del progetto è una versione preliminare dell'articolo [16] che descrive una strategia di testing di politiche di controllo degli accessi espresse tramite il linguaggio XACML denominata X-CREATE (XaCml REquests derivAtion for TEsting). Questa strategia definisce le specifiche di un framework in grado di automatizzare l'intero processo di testing di una politica di controllo degli accessi: la generazione dei casi di test, l'esecuzione dei casi di test e la valutazione dei risultati ottenuti nella fase di esecuzione.

Lo scopo del tirocinio è progettare e implementare un framework per X-CREATE al fine di verificare la sua bontà e confermare le ipotesi teoriche che stanno alla base di tale strategia e, se possibile, migliorare tale strategia. I casi di test sono applicati in due scenari diversi:

1. nel primo scenario si vuole verificare che la politica esprima l'intento dell'autore: cioè, verificare la correttezza delle specifiche della politica. In questo caso si assume che il modulo del sistema di controllo delegato alla formulazione della risposta sia corretto. Nei sistemi di controllo degli accessi tale modulo è denominato PDP (Policy Decision Point);

2. nel secondo scenario si vuole verificare la correttezza dell'implementazione del PDP. In questo caso si assume che le specifiche della politica siano corrette.

La verifica e la valutazione di tale strategia non sono oggetto di questo tirocinio.

Sistemi di controllo degli accessi

Un sistema di controllo degli accessi regola l'accesso di un soggetto ad una risorsa per effettuare alcune operazioni sotto determinate condizioni; la regolamentazione viene effettuata prima definendo e successivamente applicando delle politiche di controllo degli accessi in modo da autorizzare o meno l'accesso alla risorsa.

Una politica definisce le regole di controllo che stabiliscono le condizioni sotto le quali consentire o negare l'accesso. Quindi di fronte ad una richiesta che esprime la volontà di un soggetto di accedere ad una risorsa, un sistema di controllo è in grado di prendere una decisione, formulando una risposta, in base alle regole definite nella politica.

X-CREATE

X-CREATE è un framework che implementa una strategia di testing per sistemi di controllo degli accessi espressi in XACML e sfrutta le potenzialità del modello delle richieste per generare i casi di test, ossia le richieste che tale sistema accetta e riconosce come valide.

A partire dallo schema delle richieste, X-CREATE genera un insieme di richieste prive di valori conformi allo schema delle richieste, le quali possono essere istanziate in un secondo momento per una particolare politica. Quando si vuole testare una politica o un sistema di controllo, si popolano tali richieste con dei valori significativi contenuti della politica sotto test in modo da ottenere delle richieste significative ed eseguibili.

1.2 L'attività di tirocinio

L'attività di tirocinio si è concretizzata nell'analisi, progettazione e implementazione del cuore del framework per X-CREATE, ossia la generazione dei casi di test a partire dallo schema delle richieste e dalla politica sotto test. Inoltre, si è migliorato il modo in cui i casi di test sono generati e lo scopo di tali casi di test: con questa implementazione è possibile generare casi di test volti sia al testing dell'intera politica sia di particolari sezioni della politica qualora fosse necessario.

1.3 Organizzazione della relazione

La relazione è organizzata nel modo seguente:

- il Capitolo 2 illustra le tecnologie utilizzate durante le attività di tirocinio;
- il Capitolo 3 contiene una panoramica sul linguaggio XACML;
- il Capitolo 4 descrive il framework per X-CREATE;
- il Capitolo 5 contiene le conclusioni ed illustra gli sviluppi futuri.

Le Appendici A e B contengono rispettivamente la sintassi del linguaggio XACML e gli esempi utilizzati.

Capitolo 2

Tecnologie Utilizzate

Il presente capitolo illustra le diverse tecnologie utilizzate durante l'attività di tirocinio, dando per ogni tecnologia una descrizione delle principali caratteristiche sfruttate.

2.1 Il mondo Java

Il linguaggio di programmazione Java [5] è stato imposto dal committente come linguaggio di programmazione principale.

2.1.1 Generalità del linguaggio Java

Java è un linguaggio di programmazione orientato agli oggetti. La piattaforma di programmazione Java è fondata sul linguaggio stesso, sulla Macchina Virtuale Java (Java Virtual Machine o JVM) e sulle API Java. Java è un marchio registrato di Sun Microsystems.
Java venne creato per soddisfare quattro scopi:

1. essere completamente orientato agli oggetti;
2. essere indipendente dalla piattaforma;
3. contenere strumenti e librerie per il networking;
4. essere progettato per eseguire codice da sorgenti remote in modo sicuro.

Altri aspetti di interesse Rispetto alla tradizione dei linguaggi a oggetti da cui deriva, Java ha introdotto una serie di notevoli novità. Fra le più significative si possono citare probabilmente la possibilità di costruire GUI (interfacce grafiche) con strumenti standard e non proprietari, la possibilità di creare applicazioni multi-thread e il supporto per la riflessione.
Java offre una vastità di librerie standard che lo rendono altamente integrabile con altre tecnologie; alcuni esempi di funzionalità di librerie di Java sono:

- accesso ai database tramite JDBC;
- manipolazione di documenti XML;
- supporto nativo per gran parte dei protocolli della famiglia IP;
- supporto per le applicazioni multimediali, streaming audio e video.

2.1.2 Swing

Utilizzato per implementare la parte grafica, Swing è un framework per Java orientato allo sviluppo di interfacce grafiche [3]. Parte delle classi del framework Swing sono implementazioni di widget (oggetti grafici) come caselle di testo, pulsanti, pannelli e tabelle.
I widget Swing forniscono una GUI (graphical user interface - interfaccia utente grafica) più sofisticata rispetto alla precedente Abstract Window Toolkit (AWT). Essendo scritti in puro Java, i widget funzionano allo stesso modo su tutte le piattaforme (su cui Java gira), al contrario delle AWT le quali sono legate al sistema grafico nativo del sistema operativo. Lo svantaggio di Swing è quello di una più lenta esecuzione. Il vantaggio è una uniformità di visualizzazione tra svariate piattaforme.

2.1.3 JDBC

X-CREATE utilizza una base di dati di supporto al modulo delegato alla generazione delle richieste finali che ospita i risultati della fase di parsing. Per comunicare con il DBMS, Java mette a disposizione una API (Application Programming Interface).
JDBC (Java DataBase Connectivity) è una API per database scritta interamente in Java [6]. JDBC fornisce metodi ed interfacce per interrogare e modificare i dati in modo indipendente dalla specifica tecnologia. L'implementazione delle interfacce JDBC è chiamata Driver; le interfacce sono standard, mentre i driver contengono le specificità dei DBMS. Infatti, la maggior parte dei produttori dei DBMS mette a disposizione il proprio driver: esiste un driver JDBC per ogni DBMS.

Un'applicazione Java che utilizza JDBC per operare sui dati di cui necessita, deve effettuare quattro passi principali:

1. caricamento del driver;
2. apertura della connessione alla base di dati;
3. richiesta di esecuzione di istruzioni SQL;
4. elaborazione dei risultati delle istruzioni SQL.

2.1.4 Ambienti di sviluppo Java

Teoricamente, per sviluppare programmi in Java è sufficiente un qualsiasi editor di testo; in pratica, se si vuole scrivere qualcosa di più del classico "hello world", occorre un ambiente di sviluppo integrato. Esistono diversi IDE (Integrated Development Environment, ambiente di sviluppo integrato), alcuni gratuiti ed altri a pagamento.

2.1.4.1 Eclipse

Suggerito dal committente come ambiente di sviluppo, Eclipse [4] è un IDE open source sviluppato inizialmente da IBM. Nato come progetto proprietario, fu reso open source e ceduto ad un consorzio che include varie aziende.

Il cuore di Eclipse è la Eclipse Platform, il cui scopo è fornire i servizi necessari per integrare gli strumenti software di più alto livello che vengono implementati come plug-in.
Un aspetto cruciale dell'architettura a plug-in è quello prestazionale, dato che i plug-in possono raggiungere un numero elevatissimo: Eclipse utilizza una politica di load on demand dei plug-in. Ciò minimizza la quantità di memoria utilizzata ed il tempo di avvio durante il quale diventa necessario solo rilevare (ma non istanziare) tutti i plug-in.

Il Workspace Lo scopo fondamentale del workspace è la gestione delle risorse utente organizzate in uno o più progetti. Ogni progetto è associato ad una cartella della "cartella di workspace" di Eclipse.

Versioning Eclipse garantisce ai team di lavorare insieme in maniera flessibile ed avanzata: è stato implementato un supporto per i tool di versioning molto esteso. L'ambiente integra un client CVS e vi sono plug-in per la maggior parte dei tool di versioning utilizzati.

L'editor di Eclipse Durante la creazione di una classe, Eclipse fornisce una serie di consigli per creare una classe che rispetti le direttive suggerite dalle specifiche Java. Oltre ad evidenziare errori, gli editor di Eclipse possiedono ottime capacità di code-completion.

Esecuzione e Debugging Un aspetto di grande flessibilità di Eclipse è costituito dalle politiche di esecuzione dei programmi. La gestione delle esecuzioni avviene tramite i profili di esecuzione (Run configurations). La stessa flessibilità per i profili di esecuzione è stata replicata per le esigenze di debugging. La gestione del debug avviene tramite i profili di debugging (Debug configurations). Il debug è associato ad una prospettiva che introduce tutte le viste necessarie per analizzare lo stato del programma durante la sua esecuzione; infatti viene mostrato:

- il progetto, i suoi thread e il punto (classe/metodo) di debug corrente;
- un elenco dei vari breakpoint che sono stati definiti;
- l'elenco delle variabili visibili nello scope corrente.

2.2 Il mondo XML

2.2.1 XML

XML [8] (acronimo di eXtensible Markup Language) è un metalinguaggio di markup, ovvero linguaggio marcatore, che definisce un meccanismo sintattico che permette di costruire nuovi linguaggi di markup. Sviluppato da XML Working Group e raccomandato dal World Wide Web Consortium (W3C) con l'obiettivo di semplificare SGML [7] (Standard Generalized Markup Language).

La possibilità di definire linguaggi propri e la diffusione di XML hanno portato alla definizione di molteplici linguaggi che si occupano di risolvere molti problemi soprattutto negli ambienti distribuiti, legati anche alla sicurezza (come il linguaggio XACML).
Infatti, XML è utilizzato sia per descrivere documenti sia per serializzare dati, ed inoltre è in grado di esprimere in modo semplice dati e documenti con una struttura anche molto complessa.

I dati rappresentati da XML hanno la particolare caratteristica di essere rappresentati tramite una struttura ad albero costituita da un'unica radice e con un numero arbitrario di sottoalberi.

Ogni nodo dell'albero può contenere un numero arbitrario n di figli $x_1 \dots x_n$ delimitati da una etichetta a, dove $\langle a \rangle$ rappresenta l'inizio (o apertura) e $\langle /a \rangle$ rappresenta la fine (o chiusura) dell'etichetta; il nodo viene, quindi, indicato da $\langle a \rangle x_1 \dots x_n \langle /a \rangle$.

Un nodo può contenere valori e attributi. Con valori si indicano stringhe, interi e altre tipologie di dati che prendono il posto di tutti o alcuni dei figli del nodo: ad esempio, in $\langle a \rangle b \langle /a \rangle$ l'etichetta a contiene come valore la stringa b. Gli attributi sono identificatori dichiarati all'apertura di una etichetta contenenti valori, ad esempio in $\langle a\ id = \text{'1'}\ nome = \text{'}b\text{'} \rangle \langle /a \rangle$ l'etichetta a contiene gli attributi *id* e *nome* rispettivamente con i valori 1 e b.

2.2.2 XML Schema

XML Schema [10] è un linguaggio di descrizione del contenuto di un documento XML. XML Schema non è altro che un documento XML che utilizza un insieme di etichette speciali che formano l'*XML Schema Language*.

Un XML Schema ha un elemento radice etichettato con $\langle xs : schema \rangle$, il quale specifica tramite l'attributo *xmlns* che nel documento sono utilizzate

le etichette definite da uno specifico standard del W3C, come illustrato in Figura 2.1.

```
1 <?xml version"1.0"?>
2 <xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
3 ... Definizione della grammatica ...
4 </xs:schema>
```

Figura 2.1: Tipica struttura di un documento XML Schema.

In XML Schema si definiscono gli elementi del documento XML tramite l'etichetta $\langle xs : element \rangle$. Un elemento può avere come tipo di dato un tipo semplice oppure un tipo complesso.

I tipi di dato semplici sono relativi a quegli elementi che non possono contenere altri elementi (foglie), non prevedono attributi e sono definiti mediante l'etichetta $\langle xs : simpleType \rangle$. Vi sono numerosi tipi di dato predefiniti come ad esempio i tipi classici: stringa, intero, booleano, ecc.

```
1 <xs:element name="cognome" type="xs:string"/>
2 <xs:element name="eta" type="xs:integer"/>
3 <xs:element name="dataDiNascita" type="xs:date"/>
4 <xs:element name="eta">
5 <xs:simpleType>
6   <xs:restriction base="xs:integer">
7     <xs:minInclusive value="0"/>
8     <xs:maxInclusive value="120"/>
9   </xs:restriction>
10 </xs:simpleType>
11 </xs:element>
```

Figura 2.2: Esempio di dichiarazione di elementi con tipo semplice.

È possibile definire tipi semplici personalizzati derivandoli da quelli predefiniti. Ad esempio, si può definire un tipo di dato come restrizione del tipo stringa, vincolando i valori ad uno specifico insieme di stringhe o ad un insieme definito da un'espressione regolare, come mostrato in Figura 2.2.

I tipi di dato complessi si riferiscono alla definizione di elementi con attributi, i quali possono contenere altri elementi.

La definizione del tipo complesso consiste generalmente nella definizione della struttura prevista dall'elemento utilizzando l'etichetta $\langle xs : complexType \rangle$. Se l'elemento può contenere altri elementi, si può definire l'insieme degli elementi che possono stare al suo interno come una sequenza o come un insieme di valori alternativi specificando, con un intervallo, quanti elementi di quel tipo devono essere presenti (si veda, ad esempio, la Figura 2.3).

```
1 <xs:element name="persona">
2   <xs:complexType>
3     <xs:sequence>
4       <xs:element name="nomeCognome"
        type="xs:string"/>
5       <xs:element name="nomeFiglio" type="xs:string"
6       maxOccurs="10" minOccurs="0"/>
7     </xs:sequence>
8   </xs:complexType>
9 </xs:element>
```

Figura 2.3: Esempio di dichiarazione di elementi con tipo complesso.

In XML Schema è possibile dichiarare tipi di dato personalizzati e fare riferimento a tali dichiarazioni quando si definisce un elemento, come avviene in modo analogo per la definizione di tipi nei linguaggi di programmazione, come illustrato in Figura 2.4. Questo consente di rendere più leggibile lo schema e di concentrare in un unico punto la definizione di un tipo utilizzato diverse volte.

```
1 <xs:complexType name="tipoPrestito">
2  <xs:sequence>
3   <xs:element name="utente" type="stringtype"/>
4   <xs:element name="biblioteca"
    type="tipoBiblioteca"/>
5   <xs:element name="libro" maxOccurs="unbounded"
    type="tipoLibro"/>
6  </xs:sequence>
7  <xs:attribute name="prestitoId"
   type="tipoPrestitoId" use="required"/>
8 </xs:complexType>
9 <xs:element name="prestito" type="tipoPrestito"/>
```

Figura 2.4: Esempio di dichiarazione di tipi personalizzati.

È possibile comporre schemi includendo o importando schemi diversi ed è possibile comporre documenti XML utilizzando tag definiti in schemi diversi. Ad esempio, si possono integrare i documenti XML che descrivono gli utenti di una biblioteca con l'introduzione di nuovi elementi che definiscono i libri che possono essere presi in prestito da ciascun utente. Supponendo che tali elementi siano definiti in un apposito XML Schema, si possono combinare le etichette derivanti dai due schemi in un unico documento XML.

2.2.3 XQuery, XPath e XDM

2.2.3.1 XDM (XQuery/XPath Data Model)

Concettualmente, un documento XML è costituito da una struttura gerarchica chiamata *albero XML*, che consiste in nodi di vari tipi organizzati ad albero. Tale albero è definito tramite un modello di dati chiamato *XQuery/XPath Data Model* (XDM) [12].
XQuery ed XPath manipolano valori di XDM. Capire tale modello equivale a capire il modello relazionale quando si impara l'SQL.
Le componenti essenziali del modello XDM sono:

- Node (nodo): un elemento XML oppure un attributo;
- Atomic value (valore atomico): un dato semplice senza struttura;
- Item (elemento): un elemento generico che può riferire un nodo oppure un valore atomico;
- Sequence (sequenza): una lista ordinata di zero, uno o più elementi.

La Figura 2.5 illustra la relazione tra questi componenti.

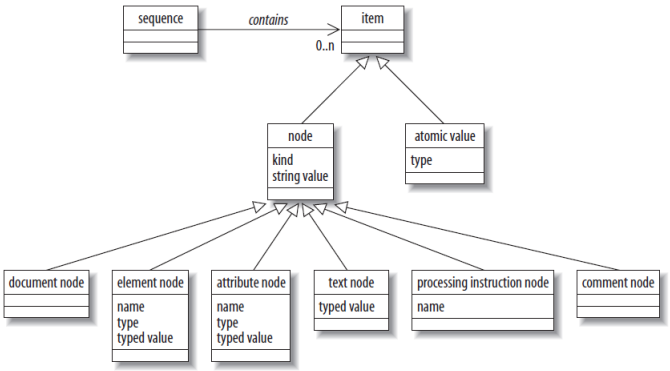

Figura 2.5: Modello dei dati XQuery/XPath.

In particolare i nodi sono utilizzati per rappresentare parti del documento XML come gli elementi e gli attributi. Vi sono sette tipi di nodi (document, element, attribute, namespace (non contemplato nella Figura 2.5), text, processing instruction, comment).
Tra i nodi sussistono relazioni di parentela:

Figlio Un nodo può avere zero, uno o più nodi figli. Un nodo *attribute* non è considerato figlio di un nodo. Un nodo *document* può avere un solo figlio.

Genitore Un nodo può avere un solo padre e il nodo *document* non ha padre. Il padre di un nodo *attribute* è il nodo che lo contiene.

Antenato Un nodo può avere zero, uno o più nodi antenati.

Discendente Un nodo può avere zero, uno o più discendenti.

Fratello Due nodi sono fratelli se hanno lo stesso padre. I nodi *attribute* non sono considerati fratelli.

2.2.3.2 XPath

XPath [9] è un linguaggio utilizzato per selezionare parti di un documento XML. È uno standard W3C dal 1999 ed ha una sintassi diversa dalla sintassi XML. Il modo in cui definisce i percorsi lungo l'albero XML per estrarre gli elementi è simile al modo in cui un sistema operativo definisce i percorsi ai file.

Una espressione XPath produce come risultato un insieme di nodi. Una espressione XPath è detta "Location Path" con la quale è possibile definire qualsiasi percorso identificando un insieme di nodi (Node Set). Una Location Path è costituita da una successione di Location Step separati dal simbolo "/" e letti da sinistra verso destra. Una Location Path ha la seguente forma:

$$\text{locationStep}_1/\text{locationStep}_2/ \ldots /\text{locationStep}_n/$$

Ogni Location Step ha la seguente forma:

$$\text{axis::nodeTest}[\text{filter}_1] \ldots [\text{filter}_n]$$

dove:

Axis individua la direzione da seguire nell'albero rispetto al nodo corrente; gli assi possibili sono 13:

- **child, descendant** figlio diretto e a qualunque livello del nodo corrente;
- **parent, ancestor** il genitore immediato e a qualunque livello del nodo corrente;
- **self** il nodo corrente;
- **attribute** gli attributi del nodo corrente;
- **preceding–sibling, following–sibling** i nodi allo stesso livello ma precedenti o seguenti il nodo corrente;
- **preceding, following** i nodi a qualunque livello che precedono o seguono il nodo corrente, eccetto il nodo corrente;
- **descendant–or–self, ancestor–or–self** come descendant e ancestor, ma considerando anche il nodo corrente;
- **namespace** il nodo namespace.

NodeTest individua il tipo e il nome completo del nodo identificato dal location step; il NodeTest identifica il tipo di oggetto da restituire. Se un asse identifica un nodo o *attribute*, questo può essere verificato attraverso un test sul nome.
Il test può essere:

- NameTest: condizione vera se il nodo (che sia elemento o attributo) possiede quel nome;
- NodeType: tipologie di nodi che diventano condizione di filtro. Le tipologie di nodi sono:
  - text() per identificare nodi di tipo testo;
  - comment() per identificare nodi di tipo commento;
  - node() identifica un generico nodo.

Filter è un predicato che raffina ulteriormente l'insieme di nodi selezionati dal location step: filtra l'insieme dei nodi rispetto alla direzione indicata dall'asse per produrre un nuovo insieme di nodi.

XPath offre più di 100 funzioni che si possono classificare in quattro tipologie:

1. funzioni node-set: agiscono su un node set e restituiscono un node set;
2. funzioni stringa: rivestono un ruolo fondamentale all'interno di XPath in quanto permettono di creare contenuti con notevole flessibilità a partire da contenuti già esistenti;
3. funzioni booleane: restituiscono tutte valori booleani;
4. funzioni numeriche: permettono di effettuare operazioni matematiche sugli argomenti.

2.2.3.3 XQuery

XQuery [11] è un linguaggio di interrogazione per documenti scritti in XML. XQuery è utilizzato come strumento di parsing per la politica sotto test; è introdotto in modo naturale con il database eXist-db il quale offre un'implementazione ottimizzata ed un ambiente di esecuzione.
Il linguaggio XQuery è stato progettato tenendo presente obiettivi specifici e identificando diversi requisiti tecnici tra i quali:

- essere in grado di trasformare e creare alberi XML e rendere possibile la combinazione di informazioni provenienti da documenti diversi;
- essere dichiarativo;
- essere in grado di utilizzare i namespace;

- essere coordinato con XML Schema e supportare tipi di dato semplici e complessi;
- prevedere almeno una sintassi XML.

Alla base di tutti questi requisiti vi è il desiderio di mantenere la stessa ideologia di SQL e di generalizzare il suo potere espressivo per manipolare documenti XML; infatti il linguaggio XQuery è stato sviluppato ispirandosi a SQL.
Il linguaggio XQuery è progettato come un superinsieme proprio di XPath; ogni espressione XPath è anche espressione di XQuery. Oltre al potere espressivo di XPath, XQuery ha la capacità di eseguire giunzione delle informazioni provenienti da fonti diverse e di generare nuovi frammenti XML. Inoltre XQuery introduce funzioni definite dall'utente permettendo così l'esecuzione di calcoli arbitrari. Una query è composta da due parti: un prologo e un corpo.

Prologo Il prologo della query è una sezione facoltativa che appare all'inizio di una query. Nonostante il nome, il prologo è spesso molto più grande del corpo. Il prologo può contenere varie dichiarazioni, separate da punto e virgola, che influenzano le impostazioni utilizzate per valutare la query. Questo include dichiarazioni di namespace, importazioni di schemi, dichiarazioni di variabili, dichiarazioni di funzioni o altro.

Corpo Il corpo della query può essere un'unica espressione oppure una sequenza di espressioni separate da virgole. Un'espressione XQuery è spesso detta FLWOR Expression. Una espressione FLWOR è simile ad uno statement SQL Select-From-Where; è composta da cinque clausole, alcune delle quali opzionali:

- clausola FOR: lega una variabile ad ogni elemento restituito da una epressione XPath. Possono esserci più clausole For all'interno di una FLWOR expression;
- clausola LET: associa ad una variabile il risultato di una espressione XPath;
- clausola WHERE: è utilizzata per definire filtri sui risultati derivati dalle clausole precedenti (for e let);
- clausola ORDER BY: definisce il tipo di ordinamento del risultato;
- clausola RETURN: specifica cosa deve essere restituito in output.

2.3 I DBMS

X-CREATE utilizza due DBMS completamente diversi che gestiscono due tipologie di dati (dati semi-strutturati e dati strutturati) per scopi diversi:

1. eXist-db [34]: DBMS nativo XML per la gestione delle politiche XACML. È utilizzato sia come repository delle politiche sia come ambiente di esecuzione delle XQuery;

2. MySQL [1]: RDBMS per la gestione della base di dati di supporto al modulo responsabile della generazione delle richieste finali.

2.3.1 eXist-db

eXist-db è un risultato open source per sviluppare un database XML che permette di memorizzare documenti XML in modo *nativo*. eXist-db può essere facilmente integrato nelle applicazioni che trattano XML in una varietà di possibili scenari.
eXist-db è completamente scritto in Java e può essere dislocato in vari modi, sia come processo server stand-alone, sia come servlet oppure direttamente embedded in un'applicazione.

I documenti XML, detti anche *risorse*, sono organizzati in *collezioni* gerarchiche. Ogni collezione è un insieme di risorse contenute in una cartella.

eXist-db utilizza XPath ed XQuery come linguaggi di interrogazione; si possono interrogare tutta o anche una parte della gerarchia di collezioni del database.
Pur essendo leggero, il motore di interrogazione di eXist-db implementa un processore efficiente basato su indici. Infatti, il cuore del sistema di archiviazione è basato su quattro indici:

- *collections.dbx* gestisce la gerarchia delle collezioni;
- *dom.dbx* gestisce tutti i nodi associando loro un identificatore univoco;
- *elements.dbx* indicizza gli elementi e gli attributi;
- *words.dbx* tiene traccia di tutte le occorrenze delle parole e viene utilizzato per le ricerche fulltext.

Questo sistema di indicizzazione migliorato permette una rapida identificazione delle relazioni strutturali fra i nodi, come padre-figlio e discendente-antenato.

Il database è appropriato per le applicazioni che trattano dalle grandi alle piccole collezioni dei documenti XML che sono occasionalmente aggiornate.

Per comunicare con eXist-db, l'accesso è fornito tramite l'HTTP, XML-RPC, SOAP e WEB-DAV.
Le applicazioni Java possono utilizzare le API XML:DB, una interfaccia

comune per l'accesso ai database nativi XML e ai database abilitati al trattamento di XML.

L'implementazione di eXist-db offre una buona interfaccia grafica che permette di utilizzarlo sia come semplice sistema di archiviazione sia come processore per XQuery.
Nel corso del tirocinio è stato utilizzato in due modi:

1. server stand-alone: sfruttando l'interfaccia grafica, è stato molto utile nella fase di apprendimento del linguaggio XQuery;

2. embedded: è il modo in cui eXist-db è dislocato in X-CREATE ed è utilizzato sia come sistema di archiviazione delle politiche XACML, sia come processore nella fase di parsing delle politiche XACML.

2.3.2 MySQL

MySQL è un DBMS open source per la gestione di basi di dati relazionali.
Utilizzato da Combo-Test come strumento principale per generare le combinazioni n-wise, MySQL è divenuto requisito tecnologico nella seconda fase del progetto (dopo la scelta di Combo-Test).
X-CREATE lo sfrutta per rendere permanenti i risultati della fase di parsing delle politiche XACML e nella fase di generazione delle combinazioni. La base di dati supporta il modulo responsabile della generazione delle richieste finali.
La comunicazione con MySQL avviene tramite JDBC, di cui offre un'implementazione detta "Connector/J".

MySQL è composto da un client e da un server ed è disponibile sia per sistemi Unix sia per Windows, anche se il suo utlizzo prevale in ambiente Unix.
Per il disegno e la modellazione di database MySQL esiste "MySQL Workbench": integra il disegno, la modellazione, la creazione e l'aggiornamento di database in un unico ambiente di lavoro.

2.4 Combo-Test

Combo-Test [40] è un tool utilizzato da X-CREATE per generare le combinazioni di tuple dei quattro insiemi.
Combo-Test è implementato come un tool a riga di comando basato su un algoritmo combinatoriale che genera una test suite con un insieme minimale di casi di test.
Combo-Test prende in ingresso una serie di parametri, i loro valori e un numero n: genera un insieme di combinazioni di questi parametri tale che per ogni coppia (tripla, quadrupla, ...) di parametri, tutte le coppie (triple, quadruple, ...) di valori di questi parametri compaiono nell'insieme di

combinazioni. Ciò significa che l'insieme di combinazioni copre tutte le coppie (triple, quadruple, ...) possibili. Il numero n determina se considerare le coppie, triple, quadruple, ...; le combinazioni così ottenute prendono il nome di combinazioni *n-wise*.
L'algoritmo implementato garantisce la generazione dell'insieme minimale di combinazioni.

A partire da un insieme di tool che generano combinazioni n-wise, la scelta di Combo-Test è stata fatta seguendo alcuni requisiti imposti dal committente:

- essere facilmente integrabile con X-CREATE; Combo-Test è scritto in Java;
- essere libero da licenza;
- essere in grado di generare un insieme di combinazioni molto ridotto;
- non avere limitazioni sulla cardinalità dei valori che ogni parametro (insieme) può assumere.

Combo-Test utilizza un database come strumento principale per la generazione delle combinazioni e questo ha portato all'introduzione di un requisito tecnologico a metà progetto.

2.5 LaTeX

LaTeX [38] è un linguaggio di markup usato per la preparazione di testi, basato sul programma di composizione tipografica TeX. Fornisce funzioni di desktop publishing[1] programmabili e mezzi per l'automazione della maggior parte della composizione tipografica, inclusa la numerazione, i riferimenti incrociati, le tabelle e figure, l'organizzazione delle pagine, le bibliografie e molto altro.
Oltre a documenti stampabili, può produrre presentazioni della stessa resa grafica grazie alla classe Beamer.
LaTeX è distribuito con una licenza di software libero e questo lo ha reso disponibile per qualsiasi architettura: ne esistono pertanto versioni funzionanti per tutti i sistemi operativi.
Negli ultimi anni anche i docenti dell'Università di Pisa iniziano a promuovere LaTeX suggerendolo ai laureandi come strumento di redazione degli elaborati finali. Anche il gruppo del labSE ne fa largamente uso nella redazione degli elaborati che illustrano i risultati raggiunti nel loro lavoro di ricerca.

[1] Il desktop publishing è l'insieme delle procedure di creazione, impaginazione e produzione di materiale stampato dedicato alla produzione editoriale (come libri, giornali, riviste o depliant), usando un personal computer.

Capitolo 3

Il Linguaggio XACML

XACML, acronimo di eXtensible Access Control Markup Language, è un insieme di schemi XML che definiscono le specifiche di un linguaggio di politiche di controllo degli accessi.
Con XACML è possibile definire delle *regole* di controllo degli accessi strutturate in *politiche* e *insiemi di politiche*. Tali regole permettono di formulare una risposta di fronte ad una richiesta che esprime la volontà di un soggetto di effettuare alcune operazioni su determinate risorse. La risposta può essere positiva (*permit*) oppure negativa (*deny*). Inoltre XACML fornisce le specifiche di un ambiente per progettare e realizzare un sistema di controllo degli accessi.

Questo capitolo illustra le specifiche del linguaggio e descrive l'architettura di un sisitema di controllo degli accessi conforme alle specifiche XACML.

3.1 Architettura XACML

Le specifiche XACML non definiscono solo la sintassi e la semantica del linguaggio, esse definiscono anche un'architettura ed alcune raccomandazioni che specificano le linee guida per progettare e realizzare un sistema di controllo degli accessi.
Queste definizioni tentano di soddisfare alcuni requisiti che un sistema di controllo degli accessi deve rispettare. I requisiti più importanti sono:

- assicurare una protezione efficace delle risorse e del sistema stesso, dal punto di vista del controllo degli accessi;
- progettare un sistema di controllo degli accessi in grado di garantire l'indipendenza dalla piattaforma utilizzata;
- permettere di integrare il sistema di controllo degli accessi all'interno di un'applicazione già esistente.

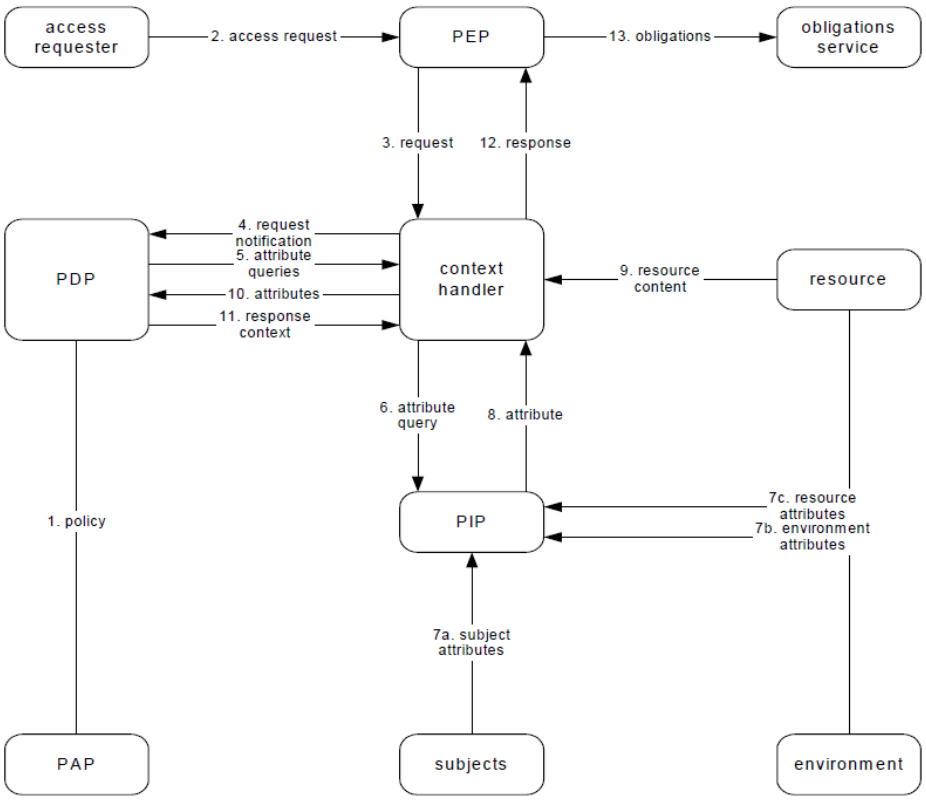

Figura 3.1: Diagramma di Flusso Xacml.

L'architettura globale del sistema definito da XACML si compone di molteplici componenti che collaborano per realizzare il sistema di controllo degli accessi, come mostrato in Figura 3.1. Di seguito si cercherà di illustrare le principali funzionalità di questi componenti e il modo in cui interagiscono fra loro.

PEP (Policy Enforcement Point) È quell'entità del sistema che effettua il controllo sugli accessi, facendo richieste di autorizzazione e facendo rispettare le decisioni di autorizzazione: il PEP è l'entità che protegge le risorse. Il PEP interagisce con le entità esterne al sistema (applicazioni o utenti) tramite le richieste di accesso ed è il responsabile dell'invio della richiesta alle entità del sistema; in base alla risposta che riceve, nega o consente l'accesso attenendosi alla decisione di autorizzazione.

PIP (Policy Information Point) È l'entità del sistema che ha la funzione di archiviare i valori dei vari attributi delle risorse, delle azioni e/o dell'ambiente. Esso fornisce i valori degli attributi al Context Handler.

PDP (Policy Decision Point) È l'entità del sistema che valuta le policy applicabili e produce la decisione di autorizzazione per l'esecuzione dell'azione sulla risorsa richiesta. Quando un utente cerca di accedere ad una risorsa, il PEP ne definisce gli attributi ed assegna al PDP il compito di decidere se autorizzare o meno la richiesta. La decisione è presa in base alla descrizione degli attributi dell'utente.

PAP (Policy Administration Point) È quella parte del sistema che produce le singole policy e i gruppi di policy (policy set) e le salva in un apposito repository.

Context Handler È l'entità del sistema che converte la richiesta dal suo formato nativo al formato canonico XACML, e viceversa, e permette la comunicazione tra tutte le altre entità del sistema.

Modello di flusso dati XACML L'interazione tra i moduli è rappresentata dal flusso di operazioni che effettuano. Il flusso di operazioni è il seguente:

1. il PAP scrive policy singole o set di policy e le rende disponibili al PDP. Questi oggetti rappresentano le politiche per uno specifico target;
2. chi richiede l'accesso alla risorsa effettua una richiesta al PEP;
3. il PEP manda la richiesta al Context Handler nel formato nativo, aggiungendo eventualmente attributi per il soggetto, la risorsa, l'azione e l'ambiente;
4. il Context Handler estrae gli attributi dei soggetti, delle risorse, dell'azione e dell'ambiente; genera una richiesta in formato XACML e la invia al PDP;
5. il PDP analizza la richiesta ed eventualmente richiede ulteriori attributi del soggetto, della risorsa, dell'azione o dell'ambiente non contenuti nella richiesta;
6. il Context Handler richiede gli attributi mancanti al PIP;
7. (a, b, c) il PIP ottiene gli attributi richiesti;
8. il PIP ritorna gli attributi richiesti al Context Handler;
9. eventualmente, e nel caso in cui le risorse coinvolte nella richiesta contengano dei dati strutturati (per esempio documenti XML), il context handler include il contenuto di tali risorse nella risposta al PDP;

10. il Context Handler invia al PDP gli attributi richiesti ed eventualmente il contenuto delle risorse;

11. il PDP valuta la richiesta in base alle politiche disponibili, genera una risposta in formato XACML (inclusa la decisione di autorizzazione) e la invia al Context Handler;

12. il Context Handler trasforma la richiesta dal formato canonico XACML al formato nativo compatibile con il formato della richiesta iniziale e la invia al PEP;

13. il PEP verifica gli obblighi e li fa rispettare;

14. (non mostrato) se l'accesso è permesso, il PEP autorizza il richiedente ad accedere alla risorsa, altrimenti gli nega l'accesso.

3.2 Le richieste XACML

In un ambiente distribuito come Internet, vi è un insieme di risorse e di servizi da condividere. Per controllare l'accesso ad una risorsa condivisa è possibile definire un insieme di regole. La volontà di un soggetto di accedere ad una risorsa è espressa tramite la formulazione di una richiesta XACML. Una richiesta di accesso ad una risorsa è una richiesta (request) che specifica i seguenti elementi:

- l'identità del richiedente, denominato soggetto (Subject);
- la risorsa alla quale il soggetto vuole accedere (Resource);
- l'azione che il soggetto vuole effettuare sulla risorsa (Action);
- l'ambiente di esecuzione dell'azione sulla risorsa (Environment).

Questi elementi possono possedere delle proprietà. Un soggetto può essere definito tramite un identificatore, l'istituzione alla quale appartiene, il ruolo che ricopre ecc...; invece, una risorsa può essere definita tramite un identificatore, un contenuto strutturato ed un tipo; così anche l'azione e l'ambiente possono essere definiti tramite un identificatore e altre proprietà che meglio li descrivono.
Si consideri la seguente richiesta espressa in linguaggio naturale:

> "una persona, paziente del reparto di ortopedia con identificatore paz100, vuole accedere alla cartella clinica che lo riguarda ed effettuare un'azione di lettura."

In questa richiesta si possono distinguere:

- il soggetto: paziente, paz100 e ortopedia;

- la risorsa: cartella clinica di paz100;
- l'azione: lettura.

		Attributo	Valore
Richiesta	**Soggetto**	ruolo	paziente
		identificatore	paz100
		reparto	ortopedia
	Risorsa	identificatore	cartella clinica
		proprietario	paz100
	Azione	identificatore	lettura

Tabella 3.1: Esempio di richiesta.

Le proprietà dei soggetti, delle risorse e delle azioni sono chiamate attributi (*attribute*). Ciascun attributo possiede un valore.
La richiesta precedente può essere definita in forma tabellare come in Tabella 3.1.
Oltre a contenere informazioni relative al soggetto, alla risorsa e all'azione, una richiesta può contenere informazioni aggiuntive legate all'ambiente, come per esempio informazioni sull'orario oppure la data.
Quando si vuole richiedere l'accesso ad una risorsa bisogna formulare una richiesta nel formato canonico XACML. A tale proposito, XACML definisce un linguaggio che stabilisce le regole da seguire per esprimere tale richiesta.

3.2.1 Il contesto di richiesta

Context Schema definisce le regole sintattiche per formulare richieste tramite documenti XACML per il contesto di richiesta e di risposta (definiti dal linguaggio per gestire gli accessi alle risorse).
Questi documenti rappresentano un'ipotetica richiesta che può essere sottomessa al PDP e sulla quale vengono poi definite una o più policy.
Queste informazioni di richiesta vengono rappresentate attraverso un "*contesto di richiesta*".
Il documento XACML che definisce questo contesto deve avere come root l'elemento `<Request>` che rappresenta la richiesta alla risorsa.
La Figura 3.2 rappresenta la tipica struttura di una richiesta.

L'elemento `<Request>` è uno strato di astrazione utilizzato dal linguaggio di policy, in quanto ad un PDP conforme a XACML non è necessario istanziare realmente il contesto di richiesta sottoforma di documento XML. Però tutti i sistemi che si attengono alle specifiche XACML devono produrre esattamente le stesse decisioni di autorizzazione, come se tutti gli input fossero trasformati nella forma di un elemento `<Request>`.

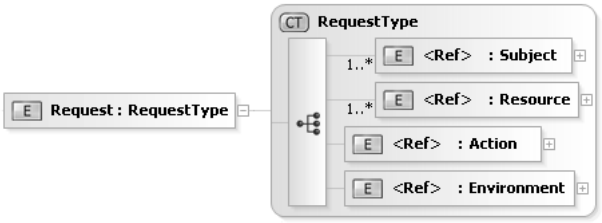

Figura 3.2: Elemento <Request>.

3.3 Le politiche XACML

Un sistema di controllo degli accessi prende una decisione in base alla richiesta e alle informazioni che essa contiene. La decisione è formulata a partire da un insieme di regole (*rule*). Le regole inoltre possono essere raggruppate in politiche (*policy*) e queste ultime possono essere raggruppate in insiemi di politiche (*policySet*).
Di seguito si illustrerà in che modo XACML definisce questi elementi.

3.3.1 Le regole di controllo (Rule)

Si può definire una regola di controllo degli accessi come un insieme di specifiche che rispondono ai seguenti quesiti:

- Quali sono i soggetti coinvolti?
- Quali sono le risorse protette per le quali è richiesto un accesso?
- Quali azioni sono richieste?
- Ci sono altre condizioni da soddisfare?
- Quale decisione prendere?

Una regola di controllo degli accessi definisce un insieme di condizioni e una decisione da prendere. La decisione può essere positiva, per permettere l'accesso alla risorsa richiesta (*permit*), oppure negativa, per negarne l'accesso (*deny*). La decisione costituisce la risposta di una regola nel caso in cui le condizioni siano verificate.
Il modo più semplice e diretto per prendere una decisione è verificare se gli attributi dei soggetti, delle risorse, dell'azione e dell'ambiente corrispondono a determinati valori. In questo caso si parla di *target*, che rappresenta una prima tappa per sapere se una regola può essere applicata ad una richiesta o meno. Quindi, una regola può definire un insieme più complesso di condizioni da verificare.
Se una richiesta corrisponde al target di una regola, allora le condizioni di questa regola saranno valutate e, se saranno soddisfatte, la risposta della

regola sarà l'effetto specificato in tale regola.
In caso contrario, se il target della regola non coincide con la richiesta o se le condizioni non sono soddisfatte, si dice che la regola è non applicabile (*not applicable*).
Si consideri il seguente esempio di regola dove:

- il target è così composto:
  - il ruolo del soggetto è paziente;
  - la risorsa è cartella clinica;
  - l'azione è lettura;
- la condizione è formata dalla seguente restrizione:
  - l'identificatore della cartella clinica deve essere uguale all'identificatore del soggetto;
- l'effetto è permettere l'accesso.

Se un paziente vuole leggere la propria cartella clinica, allora la regola sarà applicata e ritornerà *permit*.

Se un paziente chiede di modificare la propria cartella clinica, la regola non sarà applicata dato che il suo target è solo l'azione di lettura.

Se un paziente chiede la lettura di una cartella clinica di un altro paziente, questa regola non sarà applicata perchè la sua condizione non è soddisfatta.
In generale, una regola non sarà applicata se non nella situazione specificata dal suo target e dalle sue condizioni.
XACML permette di esprimere una regola tramite l'elemento `<Rule>`, come mostrato in Figura 3.3.

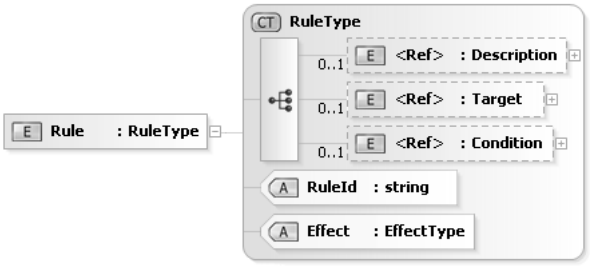

Figura 3.3: Elemento <Rule>.

Per affrontare diverse situazioni, è possibile definire più regole di controllo degli accessi tramite un loro raggruppamento: la politica.

3.3.2 Le politiche di controllo (Policy)

Una politica (*Policy*) è quell'entità che raggruppa più regole di controllo degli accessi ed è la più piccola entità che un PDP può valutare. Raggruppare regole in un unico insieme ha il vantaggio di ottimizzare la loro valutazione. Analogamente ad una regola, una politica possiede un *target* che restringe il suo campo d'applicazione ad un insieme limitato di richieste che soddisfano particolari condizioni. La Figura 3.4 illustra la struttura tipica di una politica XACML.
Si supponga di avere delle regole di controllo degli accessi, che proteggono le cartelle cliniche del reparto di ortopedia, raggruppate in una politica e che definiscono le seguenti situazioni:

1. un medico ortopedico può leggere e modificare le cartelle cliniche dei suoi pazienti, ma non può né leggere né modificare le cartelle cliniche dei pazienti seguiti da un altro medico ortopedico;

2. un paziente del reparto di ortopedia può leggere la propria cartella clinica, ma non la può modificare;

3. un paziente del reparto di ortopedia non può né leggere né modificare le cartelle cliniche degli altri pazienti del reparto di ortopedia;

4. il personale infermieristico del reparto di ortopedia può leggere le cartelle cliniche di tutti i pazienti del reparto.

Si può osservare in questo esempio che il campo di applicazione di questa politica è limitato alle azioni di lettura e modifica sulle risorse cartelle cliniche e ai soggetti medici ortopedici, pazienti e personale infermieristico. In particolare si può definire un target della politica come segue:

- il ruolo del soggetto può essere paziente, medico ortopedico oppure personale infermieristico;

- il nome delle risorsa è cartella clinica;

- il nome dell'azione che si può effettuare è lettura oppure modifica.

Una volta che una politica è applicata ad un contesto di richiesta, tutte le regole contenute nella politica saranno applicabili. Una selezione più fine di queste richieste può essere applicata tramite il *target* delle regole per limitare il numero di regole applicabili, ed avere così un'ottimizzazione nella valutazione della politica.
Quindi, se un paziente vuole leggere la sua cartella clinica, soltanto le regole 2 e 3 saranno applicate.

Dal momento che più regole possono essere contenute in una politica, e a ciascuna regola è associata una decisione, si possono avere più regole applicabili ad un contesto di richiesta. Di conseguenza si possono prendere

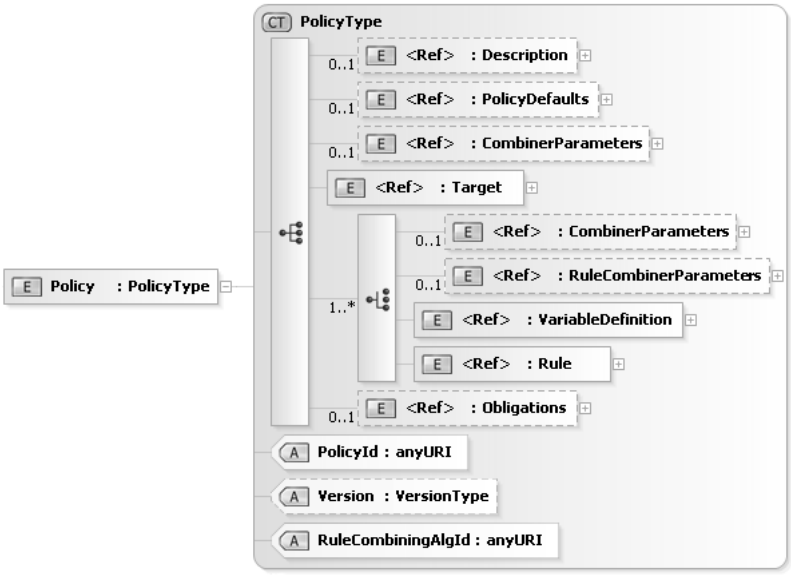

Figura 3.4: Elemento <Policy>.

più decisioni. Un PDP che valuta una richiesta si potrebbe trovare nella situazione di non sapere quale decisione prendere.
Per ovviare a questo, XACML definisce degli algoritmi di combinazione (*Rule Combining Algorithms*) per conciliare le decisioni che le regole individuali prendono all'interno della politica.
Gli algoritmi di combinazione standard sono: Deny-overrides, Permit-overrides, First applicable, Only-one-applicable. XACML, inoltre, permette di definire algoritmi personalizzati.

In congiunzione alla specifica delle regole di accesso una politica può definire degli obblighi, ovvero delle azioni supplementari che devono essere rispettate. Quando il PDP valuta una policy che contiene degli obblighi, restituisce un certo numero di questi obblighi al PEP nel contesto di risposta; gli obblighi sono delle direttive che il PEP ha il compito di far rispettare. Come esempio di obbligo, si potrebbe considerare l'invio di un messaggio di posta elettronica all'infermiere capo sala del reparto di ortopedia quando una cartella clinica viene modificata da parte di un medico ortopedico.

3.3.3 L'insieme di politiche (PolicySet)

Come le regole, anche le politiche possono essere raggruppate in base a qualche criterio; per esempio si possono raggruppare le politiche che proteggono l'accesso alle risorse del reparto di ortopedia in un unico insieme di politiche. XACML permette di definire insiemi di politiche introducendo un altro livello di astrazione (*PolicySet*).

Inoltre, XACML permette di aggregare insiemi di politiche in altri insiemi di politiche; in questo modo si ottiene una struttura ad albero degli insiemi delle politiche, delle politiche ed infine delle regole. La Figura 3.5 mostra la tipica struttura di un insieme di politiche XACML.
L'insieme delle politiche possiede un target che determina la sua applicabilità ad un contesto di richiesta. Per esempio si possono definire quali soggetti, quali risorse e quali azioni sono coinvolte nel reparto di ortopedia.

Nello stesso modo in cui avviene per le politiche, la valutazione dell'insieme di politiche da parte di un PDP può generare dei conflitti di autorizzazione: più politiche potrebbero essere applicate e di conseguenza potrebbero essere generate risposte differenti.
XACML definisce degli algoritmi di combinazione (*Policy Combining Algorithms*) per conciliare le decisioni che le singole policy o policySet prendono all'interno dell'insieme di politiche.
Anche ad un insieme di politiche possono essere associati degli obblighi.

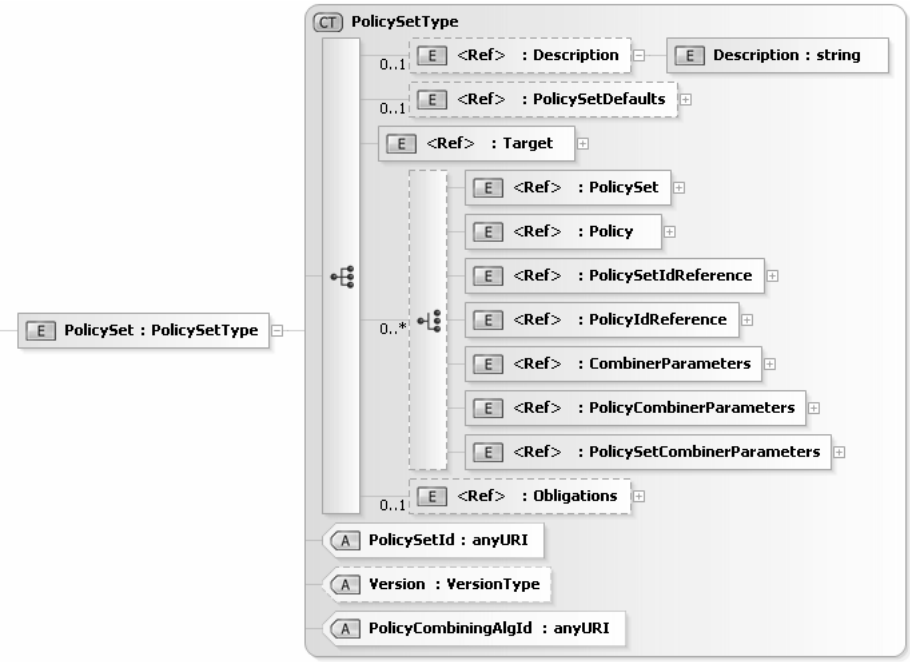

Figura 3.5: Elemento <PolicySet>.

3.4 Le risposte XACML

Una volta che il richiedente effettua una richiesta per eseguire un'azione su una particolare risorsa, e una volta che a questa richiesta vengono applicate delle policy dal PDP, il Context Handler deve produrre una risposta adegua-

ta per il richiedente.
La Figura 3.6 illustra la tipica struttura del contesto di risposta XACML.

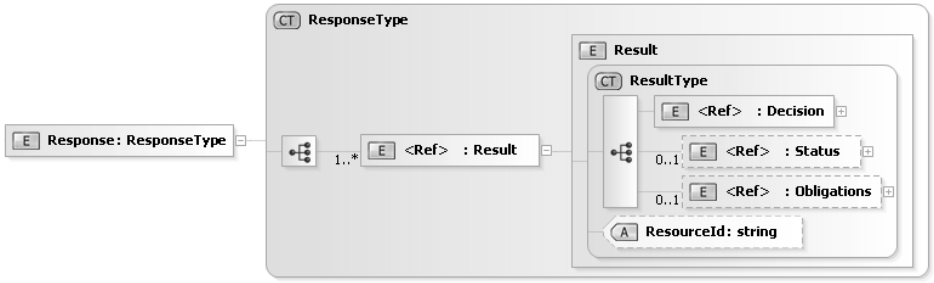

Figura 3.6: Elemento <Responce>.

L'elemento `<Response>` è l'elemento radice dei documenti che rappresentano il contesto di risposta. Anch'esso definisce un livello di astrazione per i sistemi di autorizzazioni che rispettano le specifiche fornite da XACML, in quanto esso deve essere trasformato nella forma di una decisione di autorizzazione corretta. Questo elemento incapsula la decisione di autorizzazione prodotta dal PDP; inoltre contiene una sequenza di uno o più risultati, il cui numero dipende dal numero delle risorse nella richiesta pervenuta.
L'elemento `<Result>` rappresenta una decisione di autorizzazione per l'accesso alla risorsa identificata dall'attributo `ResourceId`.
Un risultato può includere una serie di obblighi che devono essere rispettati dal PEP. Se il PEP non comprende o non può rispettare un obbligo, deve comportarsi come se il PDP abbia negato l'accesso alla risorsa richiesta.
L'elemento `<Decision>` contiene il risultato dell'applicazione della policy sulla richiesta; i valori permessi per questo elemento sono:

- *Permit*: l'accesso richiesto è permesso;
- *Deny*: l'accesso richiesto è negato;
- *Indeterminate*: il PDP non è stato in grado di valutare la richiesta di accesso;
- *NotApplicable*: il PDP non ha nessuna policy da applicare alla richiesta di accesso.

Capitolo 4

X-CREATE

Lo schema delle politiche definito da XACML è stato ampiamente sfruttato ai fini del test. Vi sono svariati tool che offrono la possibilità di genarare i casi di test a partire dallo schema delle politiche.

Tuttavia, ai fini della generazione di casi di test vi è un secondo schema definito da XACML: il Context Schema che rappresenta tutte le richieste che un PDP riconosce e accetta come valide.

Il potenziale di questo schema non è stato ancora sfruttato e nemmeno esplorato. Non vi sono ancora metodologie di testing delle politiche che sfruttano questo interessante ed importante schema.

X-CREATE è una strategia che coniuga il potenziale del Context Schema per descrivere i dati di input con un metodo per generare in modo sistematico test suite per il testing delle politiche di controllo degli accessi.

In questo capitolo verrà illustrato il cuore del framework per X-CREATE ossia il generatore automatico di casi test a partire dallo schema del contesto di richiesta. Verranno illustrati i principali requisiti del framework realizzato, i principali aspetti progettuali e implementativi.

4.1 Il modello di sviluppo software

Vi sono situazioni in cui i requisiti iniziali del software sono piuttosto definiti ma l'ampiezza dell'attività di sviluppo preclude un processo puramente lineare. Inoltre, vi può essere un urgente bisogno di fornire al committente un insieme limitato di funzionalità software con una certa rapidità per poi raffinare ed espandere queste funzionalità nelle release successive. In questi casi viene scelto un modello a processo progettato per produrre il software a incrementi.

Il modello incrementale combina alcuni aspetti del modello a cascata applicati iterativamente. Il modello a cascata, chiamato anche *ciclo di vita classico* o modello sequenziale, è basato su un approccio sistematico e sequenziale allo sviluppo di software, che inizia con la specifica dei requisi-

ti e procede attraverso la pianificazione, la modellazione, la costruzione e il dispiegamento, culminando in un supporto continuo del software completato.

Il modello incrementale consiste nell'applicare più sequenze lineari scalate nel tempo. Ogni sequenza lineare produce uno *stadio* operativo del software fino a giungere al prodotto finale.

X-CREATE è stato sviluppato seguendo il modello incrementale: è stato pianificato in modo da controllare tutti i rischi derivanti da uno sviluppo di tutte le funzionalità.

Nel modello adottato si possono individuare tre stadi principali:

1. rilascio di PolicyAnalyzer: è il primo incremento; è la fase più problematica e meno controllata. Coincide con la fase iniziale del progetto, dove i requisiti non sono del tutto chiari sia al committente sia al tirocinante. Lo studio del linguaggio XACML ha portato alla riformulazione di molti requisiti prima considerati consistenti e stabili;

2. rilascio di RequestGenerator: in questo stadio sono state aggiunte funzionalità e nuovi requisiti non considerati nel primo incremento; in particolare è stato aggiunto il requisito relativo alla progettazione di una base di dati che ha dilatato sensibilmente la durata di questa fase e la modifica della release rilasciata precedentemente;

3. rilascio di Gui (che coincide con il rilascio del prodotto): la Gui è stata progettata tenendo presente la possibilità di un'interazione molto semplice ed intuitiva;

4.2 Requisiti generali

X-CREATE (XaCml REquests derivAtion for TEsting) è una strategia descritta in [16] che definisce i requisiti generali di un framework in grado di generare casi di test per politiche di controllo degli accessi. In particolare, descrive i passi necessari per ottenere i casi di test a partire dal Context Schema. I passi principali sono:

- generazione delle richieste intermedie;
- analisi della politica sotto test;
- assegnamento dei valori alle richieste.

4.2.1 Generazione delle richieste intermedie

Dato il Context Schema, si applica l'approccio XML-based Partition Testing (XPT) proposto in [15]. Tale approccio genera delle istanze XML conformi allo Schema al quale fanno riferimento applicando il metodo Category Partition (CP) definito in [36].

Per la generazione delle richieste intermedie si utilizza il tool TAXI che implementa la strategia appena descritta. Le richieste intermedie sono delle istanze XML valide rispetto al Context Schema prive di valori: rappresentano delle generiche richieste che devono poi essere istanziate per una particolare politica in modo da ottenere richieste eseguibili e valutabile da parte di un PDP.

TAXI è un tool che implementa la metodologia XPT (XML Partition Testing) per la generazione automatica di istanze conformi ad uno Schema XML. TAXI è particolarmente indicato per il testing di applicazioni che ricevono in input istanze XML. La generazione delle istanze da parte del tool è sistematica in quanto basata su metodi allo stato dell'arte per il testing black-box. In particolare le istanze XML sono derivate considerando tutte le possibili combinazioni di elementi all'interno dello schema XML di riferimento.

In realtà le richieste intermedie possono essere generate con l'ausilio di qualsiasi tool in grado di generarle. X-CREATE, non dipende dallo specifico tool di generazione delle richieste intermedie: già a partire dalla formulazione dei requisiti si è deciso di emulare l'interazione con tale tool con la reperibilità delle istanze intermedie su file system.

Le richieste intermedie sono generate una volta per tutte indipendentemente dalla specifica politica che si vuole testare. Solo in un secondo momento tali richieste sono selezionate e istanziate per una specifica politica. La generazione delle richieste intermedie non è coordinata con le specifiche della politica sotto test. La cosa fondamentale è che le richieste intermedie siano conformi al Context Schema.

4.2.2 Analisi della politica sotto test

Per istanziare le diverse richieste intermedie è necessario che queste contengano dei valori validi e significativi ai fini del test per una particolare politica. Un assegnamento puramente casuale può non essere sufficiente a esercitare tutte le funzionalità della politica. I sistemi di controllo degli accessi, o meglio le politiche, sono istanziati su un dominio ben preciso nel quale è definito un vocabolario che definisce la semantica di tutti i valori che la politica manipola: assegnare valori al di fuori da tale vocabolario potrebbe non essere sufficiente a costruire richieste applicabili alle politiche e di conseguenza valutabile da parte di un PDP. Quindi è necessario estrarre i valori da assegnare agli elementi e agli attributi presenti nelle richieste inermedie: questi valori sono derivati dai valori contenuti nella politica sotto test.

Come detto nel Capitolo 3, il cuore di una richiesta sono gli attributi che definiscono le proprietà dei soggetti, le risorse, le azioni e l'ambiente, ed è in base a teli attributi che un PDP conforme alle specifiche XACML prende una decisione di autorizzazione. In sostanza le richieste (i casi di test) si devono tradurre in ricostruzione degli elementi `<Attribute>`. Tali

elementi sono ricostruiti a partire dalle informazioni contenute nella politica sotto test. In particolare sono definiti i passi principali affinché si ottengano dei valori significativi. Tali passi si possono vedere come *criteri di selezione* dei valori che compongono un elemento `<Attribute>` chiamato anche *entità*. Questi criteri sono si possono formulare come interrogazioni sul file XML che rappresenta una politica XACML.
I passi da seguire sono i seguenti:

1. definire 4 insiemi di valori: *SubjectSet*, *ResourceSet*, *ActionSet* ed *EnvironmentSet*;

2. per ogni elemento `<Target>` di ogni elemento `<Rule>`, `<Policy>` e `<PolicySet>` prendere:

   (a) di ogni discendente `<SubjectMatch>`, i valori degli elementi `<AttirbuteValue>` e i valori degli attributi `AttributeId`, `DataType`, `Issuer` e `SubjectCategory` dell'elemento `<SubjectAttributeDesignator>` e, qualora non fosse presente, prendere i valori dell'attributo `DataType` dell'elemento `<AttributeSelector>`. A partire da questi elementi costruire un elemento `<Attribute>` ed inserirlo nell'insieme *SubjectSet*;

   (b) di ogni discendente `<ResourceMatch>`, i valori degli elementi `<AttirbuteValue>` e i valori degli attributi `AttributeId`, `DataType` e `Issuer` dell'elemento `<ResourceAttributeDesignator>` e, qualora non fosse presente, prendere i valori dell'attributo `DataType` dell'elemento `<AttributeSelector>`. A partire da questi elementi costruire un elemento `<Attribute>` ed inserirlo nell'insieme *ResourceSet*;

   (c) di ogni discendente `<ActionMatch>`, i valori degli elementi `<AttirbuteValue>` e i valori degli attributi `AttributeId`, `DataType` e `Issuer` dell'elemento `<ActionAttributeDesignator>` e, qualora non fosse presente, prendere i valori dell'attributo `DataType` dell'elemento `<AttributeSelector>`. A partire da questi elementi costruire un elemento `<Attribute>` ed inserirlo nell'insieme *ActionSet*;

   (d) di ogni discendente `<EnvironmentMatch>`, i valori degli elementi `<AttirbuteValue>` e i valori degli attributi `AttributeId`, `DataType` e `Issuer` dell'elemento `<EnvironmentAttributeDesignator>` e, qualora non fosse presente, prendere i valori dell'attributo `DataType` dell'elemento `<AttributeSelector>`. A partire da questi elementi costruire un elemento `<Attribute>` ed inserirlo nell'insieme *EnvironmentSet*;

3. per ogni elemento `<Rule>` di ogni elemento `<Policy>` prendere tutti i valori degli elementi `<AttributeValue>` della sezione `<Condition>`

ed inserirli negli insiemi *SubjectSet*, *Resourceset*, *ActionSet* ed *EnvironmentSet*, se questi riferiscono rispettivamente gli elementi `<SubjectAttributeDesignator>`, `<ResourceAttributeDesignator>`, `<ActionAttributeDesignator>` o `<EnvironmentAttributeDesignator>`. Prendere, successivamente, tutti i valori degli attributi chiamati `AttributeId`, `Datatype`, `Issuer` (optionale) della sezione `<Condition>` e inserirli negli insiemi *SubjectSet*, *Resourceset*, *ActionSet* ed *EnvironmentSet*, se questi sono specificati rispettivamente negli elementi `<SubjectAttributeDesignator>`, `<ResourceAttributeDesignator>`, `<ActionAttributeDesignator>` o `<EnvironmentAttributeDesignator>`.

Inoltre, ai fini della robustezza e il testing negativo a ogni insieme di aggiungono delle entità generate in modo casuale.

Considerazioni sui requisiti

I requisiti o passi 2.a, 2.b, 2.c e 2.d hanno la potenziale peculiarità di attribuire una caratteristica, in termini di attributi, di una entità ad un'altra; cioè, inserire in un insieme un elemento `<Attribute>` che dovrebbe essere inserito in un insieme diverso. Per esempio un elemento Attribute relativo ad una risorsa potrebbe essere inserito nell'insieme SubjectSet anziché nel ResourceSet. Questo è dovuto probabilmente a una interpretazione errata del significato dell'elemento `<AttributeSelector>`, in quanto le specifiche XACML definiscono tale elemento in termini di espressione XPath. L'espressione XPath potrebbe identificare qualsiasi elemento `<Attribute>` all'interno della richiesta a prescindere da dove è definito l'elemento `<AttributeSelector>` all'interno della politica: anche se l'elemento `<AttributeSelector>` è contenuto nell'elemento `<SubjectMatch>` non è garantito che l'elemento `<AttributeSelector>` selezioni o riferisca un elemento `<Attribute>` che risiede nell'elemento `<Subject>` nella richiesta.

Per risolvere questo problema si deve prendere in considerazione l'espressione XPath definita nell'elemento `<AttributeSelector>` e, in base al percorso che definisce, inserire l'elemento `<Attribute>` costruito nell'insieme opportuno; nel caso in cui non si riesca a definire l'insieme al quale l'elemento `<Attribute>` appena costruito deve appartenere, si è deciso di inserirlo in tutti gli insiemi dal momento che potrebbe risiedere in qualsiasi elemento della richiesta.

Il terzo criterio è definito in termini generali e molto astratti. La sua formulazione non è affatto chiara e si indirizza a essere discusso e risolto.

L'elemento `<Condition>` rappresenta una funzione booleana applicabile agli attributi di un soggetto, risora, azione o ambiente; contiene un solo elemento `<Expression>` con la restrizione che deve ritornato un tipo di dato booleano. Tale elemento è un tipo astratto e può essere sostituito da sette elementi diversi:

<**Apply**> denota l'applicazione di una funzione ai suoi argomenti così codifica una chiamata di funzione. Questo elemento può essere applicato a ogni combinazione dei membri del *Group Substitution* dell'elemento `<Expression>`;

<**AttributeSelector**> identifica gli attributi dalla loro posizione nel contesto di richiesta;

<**AttributeValue**> può contenere un valore letterale dell'elemento `<Attribute>`;

<**Function**> può essere utilizzato per chiamare una funzione come argomento dell'elemento `<Apply>`;

<**VariableReference**> è utilizzato per riferire un valore definito all'interno dello stesso elemento `<Policy>`. Questo elemento può riferire una elemento `<variableDefinition>`. L'elemento `<VariableDefinition>` può essere utilizzato per definire un altro elemento `<Expression>`, e permettere la definizione per ricorrenza;

<**ActionAttributeDesignator**> permette di recuperare un insieme di valori contenuti nell'elemento `<Attribute>` dell'elemento `<Action>` all'interno del contesto di richiesta;

<**ResourceAttributeDesignator**> permette di recuperare un insieme di valori contenuti nell'elemento `<Attribute>` dell'elemento `<Resource>` all'interno del contesto di richiesta;

<**SubjectAttributeDesignator**> permette di recuperare un insieme di valori contenuti nell'elemento `<Attribute>` dell'elemento `<Subject>` all'interno del contesto di richiesta;

<**EnvironmentAttributeDesignator**> permette di recuperare un insieme di valori contenuti nell'elemento `<Attribute>` dell'elemento `<Environment>` all'interno del contesto di richiesta.

Come di può notare solo l'elemento `<Apply>` può essere figlio diretto dell'elemento `<Condition>` dato che è l'unico elemento che può restituire un tipo di dato booleano e che applica una funzione ai suoi argomenti, cioè applica una funzione ai suoi figli. Mentre gli altri elementi devono per forza risiedere al suo interno.

Come si può osservare la definizione è data in termini molto generali e questo può portare alla definizione di un elemento `<Condition>` in modo anche molto complesso. Capire quando un `<AttributeValue>` è associato ad uno degli elementi `<AttributeDesignator>` non è semplice ed intuitivo; e di conseguenza anche la sua trattazione risulta essere complicata.

Lo studio di XACML e la reperibilità di esempi di politiche contenenti diversi esempi di costruzione dell'elemento `<Condition>` ha portato all'osservazione che per identificare l'associazione tra `<AttributeValue>` e `<SttributeDesignator>` bisogna conservare tutta la gerarchia identificata dall'elemento `<Condition>` e utilizzare le diverse funzioni come ulteriore restrizione per la selezione dei valori ricercati: ci si deve basare sulla semantica delle funzioni per assegnare i valori adeguati alle diverse entità.

I requisiti formulati in questo modo consentono la generazione di casi di test volti al testing dell'intera politica: i 4 insiemi di entità/tuple sono considerati come insiemi piatti; non vi è alcuna relazione tra le entità se non quella di appartenere allo stesso insieme. Le uniche richieste finali generabili sono volte al testing dellŠintera politica in quanto vengono considerate tutte le entità di tutti gli insiemi. Osservando che una politica XACML è un'istanza XML che può essere rappresentata tramite un albero XML, si evidenzia che tra le diverse entità sussistono delle relazioni di parentela. Dato che unŠentità risiede in un nodo dell'albero XML e dato che esistono relazioni tra i nodi dell'albero, allora le entità possono ereditare le relazioni esistenti tra i nodi in cui risiedono. Quindi gli insiemi non sono piatti ma hanno una struttura ad albero; questo permette di definire relazioni di parentela tra entità dello stesso insieme (relazioni intra-insieme) e relazioni tra entità di insiemi diversi (relazioni inter-insieme). Vedere la politica come un albero rende possibile considerare un qualsiasi suo sottoalbero: questo permette di generare le richieste finali considerando solo le entità che vi risiedono. Ciò equivale a generare casi di test sia per parti specifiche della politica, sia per l'intera politica.

4.2.3 Assegnamento dei valori alle richieste

Una volta che si sono costruiti i 4 insiemi e una volta che le richieste intermedie sono disponibili bisogna popolare le richieste intermedie con tali valori per eleggerle a richieste finali eseguibili.La strategia definisce i passi da seguire per ottenere le richieste finali a partire dalle richieste intermedie e dagli insiemi che contengono le entità Subject, Resource, Action ed Environment.

L'assegnamento dei valori è realizzato con i seguenti passi:

1. derivare l'insieme delle entità Subject, Resource, Action e Environment, eseguito nel passo precedente;

2. generare il ValuesSet con le combinazioni delle entità subjet, resource, action e environment seguendo un approccio combinatoriale. In particolare, i valori sono inseriti nell'insieme applicando per primo la tecnica combinatorile Pair-Wise; poi, se ci sono più richieste intermedie che devono essere riempite, viene applicato il Three-Wise ed infine il

Four-Wise in modo che tutte le combinazioni possibili siano generate (si veda la Sezione 2.4 e [40]);

3. prendere i valori dal ValuesSet uno per uno e utilizzarli per riempire gli elementi e gli attributi di Subject, Resource, Action ed Environment delle richieste intermedie. Se il ValuesSet è stato utilizzato interamente per riempire le richieste intermedie, allora prendere di nuovo i valori del ValuesSet nello stesso ordine. Se ci sono altri elementi e attributi nei Subject, Resource, Action ed Environment nella richiesta intermedia, allora riempirli estraendo casualmente i valori dagli insiemi rispettivamente SubjectSet, ResourceSet, ActionSet e EnvironmentSet, escludendo i valori delle entità Subject, Resource, Action ed Environment già assegnati.
   Questo passo definisce le specifiche dell'algoritmo responsabile di riempire opportunamente le richieste intermedie eleggendole a richieste finali. Come si vedrà nel Paragrafo 4.6.1 quest'algoritmo presenta alcuni problemi relativi alla bontà del contenuto delle richieste intermedie; in particolare con questo algoritmo vi è una grande probabilità che le richieste intermedie strutturalmente diverse possano collidere nella medesima richiesta finale dopo il loro popolamento qualora la politica sotto test non contenga abbastanza valori.

4.3 Architettura di X-CREATE

X-CREATE è un framework che prende in ingresso una politica con l'aggiunta opzionale di un insieme di richieste intermedie, e restituisce un insieme di richieste conformi allo schema del contesto di richiesta XACML (Figura 4.1). Tali richieste sono la test suite per una eventuale batteria di test. In

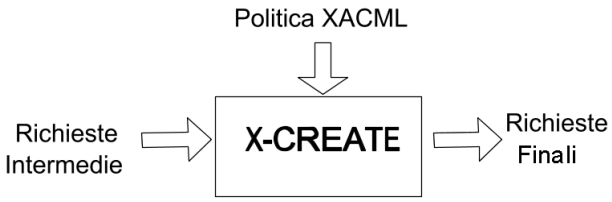

Figura 4.1: X-CREATE.

particolare si possono avere due tipi di richieste:

richieste semplici sono derivate direttamente dalle combinazioni dei quattro insiemi; per ogni combinazione si genera una richiesta semplice contenente le informazioni della richiesta e si compone al massimo di quattro elementi: un elemento `<Subject>`, un elemento `<Resource>`, un elemento `<Action>` ed eventualmente un elemento `<Environment>`;

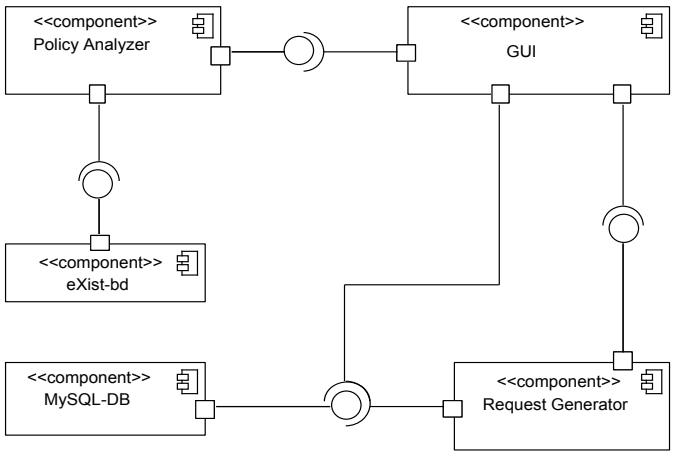

Figura 4.2: Diagramma C&C.

richieste derivate dalle richieste intermedie sono derivate a partire dalle richieste intermedie: per ogni richiesta intermedia si genera una richiesta finale tramite popolamento a partire dalle combinazioni.

Per generare le richieste finali, X-CREATE esegue un parsing della politica per estrarre gli attributi che rappresentano le entità subject, resource, action ed environment tramite una espressione XQuery e formare così quattro insiemi rappresentati come alberi radicati che conservano buona parte della struttura della politica sotto test. A partire da questi insiemi, il sistema genera le combinazioni delle entità in essi contenute seguendo un approccio combinatoriale e infine genera tante richieste semplici quante ne desidera l'utente, oppure richieste derivate dalle richieste intermedie fornite dall'utente.

Il framework per X-CREATE è composto da due componenti principali e una interfaccia (Figura 4.2):

- un analizzatore della politica sotto test, chiamato PolicyAnalyzer che ha il compito di ricevere, memorizzare ed analizzare la politica XACML in modo da derivare i 4 insiemi;
- un generatore di richieste conformi al Context Schema, chiamato RequestGenerator, che ha il compito di generare le combinazioni, generare le richieste finali semplici e le richieste finali a partire dalle richieste richieste intermedie;
- un interfaccia grafica (GUI) che permette di interagire con l'utente e che coordina le responsabilità dei due componenti principali.

4.4 PolicyAnalyzer

Per portare a termine il primo stadio ed offrire un prodotto base che serve come piattaforma per una prima validazione della strategia X-CREATE, è stato necessario realizzare il componente PolicyAnalyzer. Il PolicyAnalyzer ha la responsabilità di derivare quattro insiemi di entità a partire dalla politica XACML e fornisce un'implementazione dei criteri di selezione.

In questo fase è stato necessario fare un'analisi attenta dei requisiti in modo da offrire strumenti utili per la valutazione dei risultati raggiunti sia in modo da progettare un prodotto facilmente estendibile data la natura sperimentale del progetto: questo permette una facile estensione sia in termini di funzionalità sia in termine di integrazione.

Oltre ad analizzare i requisiti si è fatto uno studio intensivo delle tecnologie da utilizzare. In particolare lo studio del linguaggio XACML, la possibilità di utilizzare eXist-db come archivio. Questo secondo aspetto ha evidenziato che eXist-db offre un ottimo processore XQuery e questo ha portato alla considerazione di utilizzare XQuery come principale strumento di parsing.

PolicyAnalyzer offre la possibilità di vedere i quattro insiemi sia in forma tabellare, ogni insieme è rappresentato da una tabella che risiede in memoria centrale, sia come file XML che conserva le entità di tutti gli insiemi come elementi strutturati su file system: questo permette all'utente di analizzare in un secondo momento le entità selezionate e verificare in modo accurato i criteri di selezione.

Il risultato dell'analisi della politica è anch'esso un file XML che lo possiamo chiamare *politica derivata*. La politica derivata mantiene alcune proprietà della politica analizzata.

I nodi rispettano la gerarchia di partenza e sono ristrutturati in modo da avere solo le informazioni necessarie al fine del test.
Il processo di analisi si può assimilare ad una funzione applicata alla politica XACML e il cui risultato è un file XML, chiamata politica derivata:

$$F : XML \rightarrow XML$$

in particolare la funzione è la seguente:

$$G : XACML \rightarrow XML$$

La funzione G è una funzione che applica i criteri di selezione delle entità alla politica ed è implementata come una espressione XQuery.

Il policyAnalyzer si compone di:

un dirver per eXist-db il driver eXist-db è responsabile della comunicazione con il database eXist-db ed offre la possibilità di creare, rimuovere e navigare collezioni di risorse, di creare, cancellare recuperare

una risorse, ed infine permette di eseguire, e ricevere il risultato di una XQuery e naturalmente la connessione e disconnessione a/da eXist-db;

un parser XQuery il parser XQuery è la parte principale di questo componente, permette di analizzare la politica sotto test e restituire come risultato un file XML che contiene le entità sotto forma di elementi. Il file XML è una ristrutturazione della politica XACML e che ne mantiene la struttura in modo da conservare le relazioni gerarchiche tra gli elementi principali quali PolicySet, Policy, Target, Rule e Condition. Questi elementi sono i principali detentori delle informazioni riguardanti le entità da selezionare;

un parser DOM questo parser ha il compito di trasformare il risultato dal suo formato nativo XML ad una rappresentazione in memoria centrale sotto forma di albero in modo da facilitarne l'elaborazione;

L'espressione XQuery ha la seguente forma:

un prologo contiene tutte le dichiarazioni dei namespace e la definizione di tutte le funzioni che implementano i criteri di selezione costruendo in questo modo il contesto di valutazione della espressione XQuery;

un corpo formato da tre espressioni:

- due espressioni di aggiornamento (eXist-db offre alcune funzionalità limitate di XUpdate):
  - espressione di risoluzione dei riferimenti di `<VariableRefence>`: sostituisce il riferimento ad una variabile con la sua definizione; questo permette di avere una visione lineare della politica e permetterne una facile elaborazione, ma ha anche lo svantaggio di espandere la politica e replicare la stessa informazione in più punti della politica e quindi eseguire lo stesso calcolo più volte; si è preferito la semplicità per avere un maggior controllo dei risultato dato che il processore XQuery offerto da eXist-db è molto efficiente;
  - espressione di risoluzione dell'elemento `<AttributeSelector>`: sostituisce questo elemento con un elemento <AttributeDesignator> in base alle informazioni contenute nell'espressione XPath;
- espressione di parsing della politica utilizzando le funzioni definite nel prologo che restituisce il risultato in un unico elemento strutturato.

In Figura 4.3 è mostrato il diagramma di attività della fase di parsing della politica.

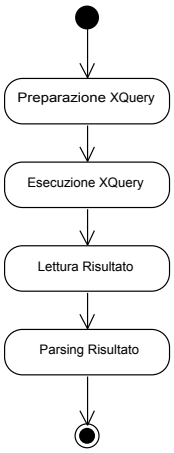

Figura 4.3: Diagramma di attività del policyAnalyzer.

Per il testing negativo e robustezza PolicyAnalyzer aggiunge valori casuali ai 4 insiemi; in partcolare aggiunge un entità in quei nodi dove sono presenti altre entità: l'elemento `<AttributeValue>` è generato in modo casuale e gli altri attributi sono scelti da un'entità di quel nodo in modo da permettere la sua selezione qualora facesse parte di di una richiesta sottoposta a un PDP.

4.5 RequestGenerator

Il secondo stadio è volto ad offrire funzionalità concrete che si traducono in generazione di test e quindi di richieste finali eseguibile per poter iniziare a fare le prime valutazioni significative.

Per portare a termine quest'obiettivo è stato necessario ricercare un tool in grado di generare le combinazioni delle entità seguendo un approccio combinatoriale. In particolare il tool doveva essere in grado di eseguire le combinazioni pairwise, three-wise e four-wise. I tool sono reperibili in [2]. Fondamentali sono stati i requisiti che doveva avere:

- essere facilmente integrabile con X-CREATE; Combo-Test è scritto in Java;
- essere libero da licenza: data la natura sperimentale del progetto nato nell'ambito della ricerca, X-CREATE sarà rilasciato con licenza open source destinato ad un target prevalentemente scientifico, la licenza non doveva essere né infettiva né restrittiva;
- essere in grado di generare un insieme di combinazioni molto ridotto;

- non avere limitazioni sulla cardinalità dei valori che ogni insieme può assumere.

Combo-test si appoggia a un DBMS e questo ha portato l'introduzione di un nuovo requisito funzionale: i quattro insiemi generati da PolicyAnalyzer devono essere permanenti così anche le combinazioni. Il fatto di rendere permanente i tali risultati ha duplice valenza: il parsing di una politica viene effettuato solo una volta così anche la generazione delle combinazioni in aggiunta per gli utilizzi successivi i tempi di risposta di X-CREATE risultano ridotti.

Le funzionalità di questo stadio sono manifestate nella progettazione e realizzazione del componente RequestGenerator che ha il compito di generare le richieste finali a partire dai valori degli insiemi SubjectSet, ResourceSet, ActionSet ed EnvironmentSet forniti da PolicyAnalyzer. Per farlo costruisce un insieme di combinazioni delle entità, e in base al tipo di richiesta finale, restituisce una richiesta finale semplice conforme al Context Schema per ogni combinazione oppure una richiesta finale per ogni richiesta intermedia.

Il componente requestGenerator è composto da:

un generatore di combinazioni è il componente responsabile della generazione delle combinazioni (chiamato *CombinationGenerator*) ed è il responsabile dell'interazione con Combo-Test. In questa fase è stato necessario integrare Combo-Test rendendolo incapsulato in X-CREATE; in particolare è stato modificato il modo in cui riceve i dati dei 4 insiemi e e il modo in cui restituisce il risultato. Combo-Test nel suo stato originale opera su dati contenuti in un file di input che risiede nel file system e restituisce i risultati in un file di output che risiede sempre nel file sysstem.
L'integrazione consiste quindi nel renderlo invocabile direttamente da *CombinationGenerator*; in particolare le operazioni principali sono:

1. preparare gli insiemi in modo che Combo-Test li possa usare;
2. avviare la funzionalità principale di Combo-Test;
3. reperire i risultati forniti da Combo-Test.

La generazione delle combinazioni si esegue una volta sola per ogni politica e a partire da un nodo dell'albero che rappresenta la politica; in particolare si generano i seguenti insiemi di combinazioni:

1. pair-wise (PW);
2. three-wise (TW);
3. four-wise (FW).

Tali insiemi hanno un rapporto di inclusione cioè:

$$PW \subseteq TW \subseteq FW$$

però ai fini del test servono solo insiemi combinazioni n-wise la cui intersezione è l'insieme vuoto, per cui si generano i seguenti insiemi:

$$PW$$

$$TW \backslash PW$$

$$FW \backslash (TW \cup PW)$$

una base di dati e il relativo gestore la base di dati è l'entità più importante del RequestGenerator se non dell'intero framework, e permette la permanenza di tutte le informazioni di una politica conservandone la struttura e tutte le combinazioni generabili da tale politica. Il gestore è rappresentato tramite un insieme di funzionalità che permettono di effettuare tutte le operazioni necessarie per la sua gestione.
La operazione principali sono:

1. connessione al DBMS che ospita la base di dati;
2. disconnessione dal DBMS;
3. creazione ed eliminazione delle tabelle;
4. visita una o più tabelle (eseguire query senza effetti laterali);
5. aggiornamento delle tabelle.

Per la sua importanza è stata dedicata la Sezione 4.5.1.

un generatore di richieste semplici è il componente responsabile della generazione delle richieste semplici: data una politica e dato un nodo dell'albero di tale politica genera una richiesta semplice per ogni combinazione generabile da quel nodo.

un generatore di richieste derivate è il componente responsabile della generazione delle richieste finali derivate a partire dalle richieste intermedie: data una politica, dato nodo appartenente alla politica e dato un insieme di richieste intermedie, per ogni richiesta intermedia genera una richiesta finale tramite popolamento a partire dalle combinazioni generabile dal nodo dato.
Le operazioni fondamentali di questo componente sono:

1. reperire le istanze intermedie;
2. parsing delle istanze intermedie;
3. modifica delle richieste intermedie eleggendole a richieste finali.

4.5.1 La base di dati

Per portare a termine il secondo stadio è stato necessario ricercare un tool in grado di generare le combinazioni n-wise delle entità dei 4 insiemi costruiti dal PolicyAnalyzer.

Combo-Test implementa un algoritmo in grado di generare le combinazioni appoggiandosi ad un database; questa caratteristica ha incuriosito il committente e ha portato l'introduzione di requisiti tecnologici e requisiti funzionali:

- requisiti funzionali: rendere permanente il risultato della fase di parsing, cioè memorizzare i 4 insiemi in una base di dati in modo da effettuare una volta per tutte l'analisi della politica in modo da renderla disponibile per i successivi utilizzi ottimizzando così il tempo di risposta di X-CREATE.
  Rendere permanenti i risultati della fase di generazioni delle combinazioni. I primi utilizzi di Combo-Test hanno evidenziato un sensibile calo di prestazioni all'aumentare della cardinalità degli insiemi: il tempo di esecuzione aumentava in maniera significativa;

- requisiti tecnologici: dato che Combo-Test può utilizzare MySQL, anche X-CREATE ne deve fare uso.

Successivamente è stato necessario progettare e realizzare una base di dati in modo da ospitare i risultati sia della fase di parsing sia i risultati ottenuti nella fase di generazione delle combinazioni.

Di seguito è riportata, in maniera molto sintetica, la narrativa del contesto in cui opera la base di dati, il dettaglio della base di dati e l'elenco delle operazione.

Obiettivi della base di dati L'obiettivo della base di dati è rendere permanenti i dati ottenuti nella fase di parsing: i valori delle entità subject, resource, action ed enviaronment. Requisito fondamentale è riuscire a conservare la struttura gerarchica con cui sono rappresentati gli insiemi delle entità.
Inoltre, l'obiettivo è conservare le combinazioni generabili da Combo-Test: in particolare bisogna tenere traccia delle combinazioni pairwise, three-wise e le combinazioni four-wise in modo rendere più veloce la costruzione delle test suite qualora fosse necessario eseguire diverse batterie di test per la stessa politica.

Descrizione della realtà Il risultato dell'analisi della politica è anch'esso un file XML che lo possiamo chiamare Politica Derivata. La politica derivata

mantiene alcune proprietà della politica analizzata. I nodi rispettano la gerarchia di partenza; i nodi sono ristrutturati in modo da avere solo le informazioni necessarie al fine del test.

Il processo di analisi si può assimilare ad una funzione applicata alla politica XACML e il cui risultato è un file XML, chiamata politica derivata:

$$F : XML \rightarrow XML$$

In particolare la funzione è la seguente:

$$G : XACML \rightarrow XML$$

Una politica derivata, di seguito chiamata semplicemente politica, è rappresentata come un albero radicato formato da uno o più nodi etichettati. Una politica ha un nome e un testo (il testo originale). Un nodo può essere un nodo radice, un nodo interno oppure un nodo foglia; quindi un nodo può avere zero o un padre e può avere zero, uno o più figli. Un nodo può ospitare zero, una o più tuple. Un nodo ha un'etichetta (chiamata semplicemente nome) e un identificatore che lo identifica in modo univoco all'interno dell'albero. Un nodo può essere di tipo PolicySet, Policy, rule, Function, Condition oppure Apply. Inoltre ad un nodo può essere associata una funzione e a solo quei nodi di tipo Function e Apply. Inoltre bisogna tenere traccia della discendenza di un nodo deve essere possibile ricostruire l'intero albero di una politica in modo da permetterne la visualizzazione e la navigazione.

Una tupla risiede in un solo nodo. Una tupla può essere di tipo Subject, Resource, Action oppure Environment. Quindi una tupla rappresenta un'entità dei quattro insiemi ed è caratterizzata dai seguenti attributi: AttributeValue, AttributeId, DataType, Issuer e SubjectCategory dove gli ultimi due sono opzionali e l'attributo SubjectCategory ha significato solo per le tuple di tipo Subject. Una tupla possiede un idTupla che la identifica in modo univoco all'insieme al quale appartiene.

Per testare una politica si effettuano diverse test suite. I casi di test sono generati secondo la tecnica combinatoriale n-wise; in particolare sono effettuate le tecniche pairwise, 3-wise e il 4-wise.

Una combinazione è una sequenza di 1, 2, 3 o 4 tuple, a seconda della presenza di tuple del tipo indicato poc'anzi. Quindi una tupla può far parte di zero, una o più combinazioni. Una combinazione è generata a partire da un nodo. Un nodo può generare zero, una o più combinazioni. Una combinazione è generata con una delle tecniche combinatoriale, quindi una combinazione ha un tipo che può essere 2-wise, 3-wise oppure 4-wise. Inoltre quando una combinazione è formata da un'unica tupla questa è associata ad un tipo astratto chiamato 1-wise.

Nella Tabella 4.1 sono riportato in modo sintetico i dati identificati: ad ogni classe identificata è riportato l'elenco degli attributi che possiede.

Classe	Attributi
Politiche	Codice politica Nome politica Testo politica
Nodi	Codice nodo IdNodo Codice Xacml Funzione
Tipi nodo	Codice tipo nodo Nome tipo nodo
Tuple	codice tupla IdTupla AttributeId DataType AttributeValue Issue SubjectCategory
Tipi tupla	Codice tipo tupla Nome tipo Tupla
Combinazioni	Codice combinazione Valore combinazione
Tipi combinazione	Codice tipo combinazione Nome tipo combinazione

Tabella 4.1: Elenco delle classi e loro attributi.

Diagramma ad oggetti delle classi Nella Figura 4.4 è illustrato lo schema concettuale delle base di dati nel quale si possono osservate tutte le associazioni e loro molteplicità tra le classi.

Commenti allo schema concettuale Una combinazione deve essere composta da almeno una entità e al massimo 4 entità, quindi non ci possono essere combinazioni che riferiscono a zero entità oppure a più di quattro entità. Inoltre, quando una combinazione riferisce a due, tre o quattro entità queste devono essere tutte di tipo diverso, per cui una combinazione non può riferirsi a due o più entità dello stesso tipo.

Un nodo per definizione è discendente di se stesso. Secondo il modello dei dati XACML, un nodo di tipo Rule può essere figlio solo di un nodo tipo Policy e i suoi figli possono essere solo di tipo Condition. Un nodo Condition può essere solo figlio di un nodo di tipo Rule. Un nodo di tipo Function può

essere figlio dei soli nodi di tipo Condition e Aplly e nello stesso modo un nodo Apply può essere solo figlio dei nodi di tipo Function e Condition. Un nodo di tipo Policy può avere padre solo un nodo di tipo PolicySet, e un nodo di tipo PolicySet può essere padre dei soli nodi di tipo Policy e policySet, e padre solo un nodo di tipo PolicySet.

Il nodo che può non avere padre può essere solo un nodo di tipo PolicySet oppure di tipo Policy: tale nodo viene chiamato nodo radice. Una Politica può e deve avere uno e solo un nodo radice.

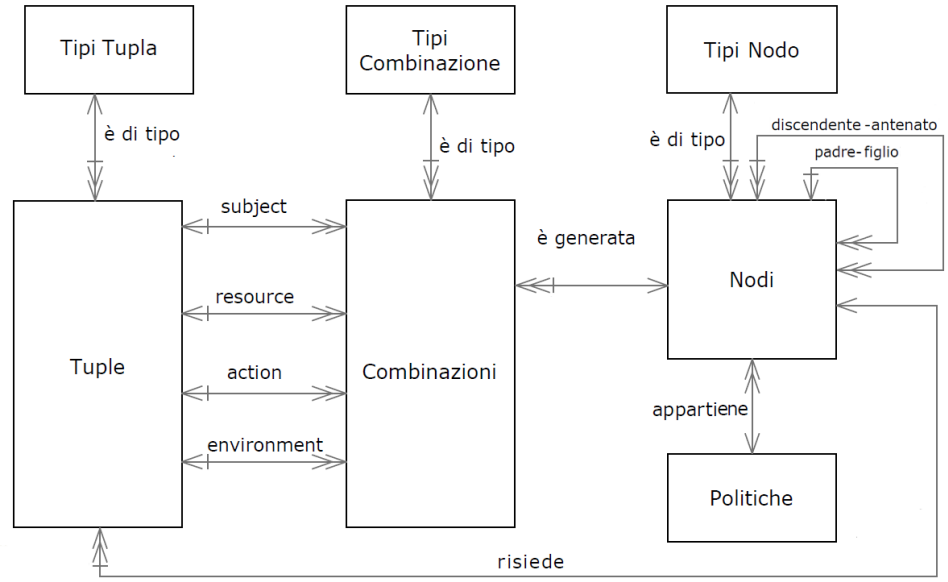

Figura 4.4: Schema concettuale.

Definizione delle operazioni della base di dati Di seguito sono riportate le operazioni principali identificate, oltre alle operazioni di inserimento, cancellazione:

1. Data una politica P, selezionare il nodo radice della politica;
2. Dato un nodo N, selezionare tutti i nodi figli;
3. Dato un nodo N, selezionare tutti i suoi discendenti;
4. Dato un nodo N, selezionare tutte le tuple di tipo TT che risiedono nei suoi discendenti; TT può essere : Subject, Resource, Action oppure Envirnoment;
5. Dato un nodo N, selezionare tutte le tuple che risiedono nel nodo;

6. Dato un nodo N, selezionare tutte le combinazioni di tipo 1-wise che sono generate a partire dal nodo N;
7. Dato un nodo N, selezionare tutte le combinazioni di tipo 2-wise che sono generate a partire dal nodo N;
8. Dato un nodo N, selezionare tutte le combinazioni di tipo 3-wise che sono generate a partire dal nodo N;
9. Dato un nodo N, selezionare tutte le combinazioni di tipo 4-wise che sono generate a partire dal nodo N;
10. Per ogni combinazione C di tipo TC, generata a partire da un nodo N, selezionare le tuple identificate dalla combinazione C;
11. Selezionare tutti i nodi che appartengono alla politica di nome Nome-Politica.

4.6 X-CREATE GUI

L'interfaccia grafica ha la responsabilità di permettere un'interazione molto semplice ed intuitiva con l'utente e durante la sua progettazione si è tenuto in conto di automatizzare gran parte dei passi necessari al fine di ottenere le test suite desiderate. L'interfaccia grafica è progettata per permettere una navigazione a schede che consente di visualizzare più politiche contemporaneamente, mantenendo però il tutto all'interno di una sola finestra, in modo da semplificare la navigazione tra l'una e l'altra.

In particolare l'utilizzo di eXist-db è trasparente all'utente in quanto è dislocato in modo embedded: questo per non costringere l'utilizzatore a dover dipendere da questo DBMS e costringerlo a reperirlo ed installarlo. Inoltre le combinazioni sono generate in modo automatico quando si sceglie di operare su una politica.

4.6.1 Le Strategie di testing

Con X-CREATE è possibile generare le richieste finali seguendo 4 diverse strategie di generazione combinatoriale, definite e formalizzate nella fase finale del progetto. Queste strategie di generazione hanno caratteristiche diverse e sono a due a due collegate. Ci sono due strategie per generare casi di test per tutta la politica e due strategie che considerando la gerarchia della politica che permettono la generazione dei casi di test per parti specifiche della politiche. Di seguito sono illustrate tali strategie.

Simple Combinatorial Generation

Con questa strategia è possibile generare delle test suite semplici seguendo un approccio combinatoriale per tutta la politica. Ogni elemento della test suite è una richiesta semplice conforme al Context Schema formata da uno, due, tre o quattro elementi di diverso tipo. È possibile ottenere una richiesta semplice per ogni combinazione. Potenzialmente le test suite generabili possono essere molte numerose dal momento che sono considerate tutte le combinazioni generate a partire da tutte le entità della politica.

Il numero di richieste generabili con questa strategia è pari a

$$\prod_{i=1}^{4} v_i \quad \text{dove } v_i \text{ è la cardinalità di ognuno dei 4 insiemi}$$

Combinatorial Generation from Intermediate Requests

Con questa strategia è possibile generare delle test suite a partire dalle richieste intermedie per l'intera politica.

L'integrazione con il tool responsabile della generazione delle richieste intermedie è stato emulato tramite l'interfaccia grafica offrendo all'utente la possibilità di selezionare le istanze XML, che rappresentano le richieste intermedie, dal file system offrendo un FileChooser quando viene selezionata tale strategia. Per ogni richiesta intermedia viene generata una richiesta finale tramite popolamento.

Hierarchical Simple Combinatorial Generation

Questa strategia permette la generazione delle test suite volte al testing di una parte della politica. Le combinazioni dalle quali sono derivate coinvolgono solo entità derivate da quella parte della politica. Questo ci permette di generare casi di test volti al testing di un sottoinsieme radicato della politica; per esempio si potrebbe voler generare dei casi di test per una specifica regola, per un insieme di regole quindi una politica, oppure un insieme di politiche quindi un policySet.

Hierarchical Combinatorial Generation from Intermediate Requests

Con questa strategia è possibile generare delle test suite volte al testing di una parte della politica. Per ogni richiesta intermedia si genera una richiesta finale tramite popolamento.

Osservazioni sulle strategie

Si supponga di avere le due richieste intermedie mostrate in B.1 e in B.2, le cui strutture in forma tabellare sono mostrate in Tabella 4.2.

Elemento	**Richiesta B.1**	**Richiesta B.2**
Subject	2	3
Resource	3	2
Action	3	4
Environment	4	2

Tabella 4.2: Cardinalità elementi Attribute nelle richieste intermedie.

Entrambe le richieste hanno un elemento Subject, un elemento Resource, un elemento Action ed un elemento Environment. I numeri nelle caselle della Tabella 4.2 rappresentano il numero di elementi Attribute di ogni elemento Subject, Resource, Action ed Environment della rchiesta intermedia B.1 e della richiesta intermedia B.2.

Si supponga inoltre che il parsing di una politica sotto test abbia portato al risultato mostrato in Tabella 4.3 dove tutti gli insiemi sono formati da due entità. Con questi elementi è possibile ottenere 16 combinazioni.

AttributeId	DataType	AttributeValue
SubjectSet		
ruolo	string	paziente
ruolo	string	RandomValueS1
ResourceSet		
resourceID	string	cartellaClinica
resourceID	string	RandomValueR1
ActionSet		
actionID	string	lettura
actionID	string	RandomValueA1
EnvironmentSet		
envID	dayTimeDuration	P7D
envID	dayTimeDuration	RandomValueE1

Tabella 4.3: Esempio di risultato del parsing di una politica.

Una volta che si applicata l'algoritmo di generazione delle richieste finali derivate dalle richieste intermedie scegliendo la strategia *Combinatorial Generation from Intermediate Requests* si ottengono le richieste finali mostrate in B.3 e in B.4. Osservando le richieste finali si nota che entrambe contengono lo stesso contenuto e che presentano alcuni elementi vuoti; questo perché non ci sono abbastanza entità nei 4 insiemi.

Quando un PDP riceve tali richieste li considera identiche dato che hanno lo stesso contenuto e dato che nella valutazione delle richieste il PDP non considera gli elementi vuoti perché non sono mai selezionati. Quindi, anche se le richieste intermedie sono strutturalmente diverse vi è un'altissima probabilità che collidano nella medesima richiesta finali, seguendo l'algoritmo proposto, qualora la politica sotto test contenga poche entità. Infine, questo problema è più accentuato quando si considera la strategia *(*Hierarchical Simple Combinatorial Generation) dato che si considerano parti della politica che potrebbero contenere pochissime entità.

4.7 Aspetti implementativi

Nella fase di implementazione e in tutte le altre fasi si è tenuto in considerazione la possibilità di una facile estensione del prodotto e questo si è verificato grazie ad una progettazione attenta e modulare; seguire il modello incrementale è la discriminante che ha permesso di rendere modulare e facilmente estendibile X-CREATE.

Durante tutta la fase di implementazione si è tenuto conto della possibilità di riutilizzo del codice da parte di terzi che dovranno estendere X-CREATE con nuove funzionalità; per cui tutti i nomi delle classi, delle variabili e delle costanti utilizzate sono nomi significativi basati sulla loro funzionalità, e ove necessario il codice è accompagnato da una descrizione più che esaustiva, alle volte anche prolissa per chiarire meglio il significato di alcune parti.

Il progetto è suddiviso in package ognuno con una responsabilità limitata e ben definita (tutti i package hanno il seguente prefisso *it.cnr.isti.labse.xcreate.*):

dbDrivers contiene le classi responsabili della comunicazione con i BDMS utilizzati da X-CREATE;

filler contiene le classi responsabili della generazione delle richieste finali semplici e le richieste finali a partire dalle richieste intermedie; inoltre contiene la classe responsabile della generazione delle combinazioni che integra Combo-test. Sia

guiXCREATE contiene la classe main (*XCreateMain*) e tutte le classi necessarie alla gestione dell'interfaccia grafica; inoltre, si è prestato molta attenzione a separare la parte grafica da quella di elaborazione;

policyAnalyzer contiene tutte le classi relative la gestione del parsing della politica sotto test;

sql contiene tutte le classi necessarie per la gestione della base di dati; in particolare contiene tutto il codice SQL responsabile delle operazioni di visita, modifica, creazione e cancellazione.

Di particolare interesse è la classe *AntenatoDiscendente* che calcola la discendenza tramite la programmazione dinamica utilizzando la tecnica della memoizzazione. La memoizzazione è una tecnica di programmazione che consiste nel salvare in memoria i valori restituiti da una funzione in modo da averli a disposizione per un riutilizzo successivo senza doverli ricalcolare. Una funzione può essere "memoizzata" soltanto se soddisfa la trasparenza referenziale, cioè se non ha effetti laterali e restituisce sempre lo stesso valore quando riceve in input gli stessi parametri.

In particolare il calcolo della discendenza di tutti in nodi dell'albero della politica viene calcolato in un'unica visita dell'albero. La discendenza di un nodo è calcolata in termini della discendenza dei suoi figli. Un nodo foglia è discendente di sé stesso.

util contiene classi generiche di utilità;

xQuery contiene l'espressione XQuery e le classi per la sua gestione. Durante l'implementazione del parser della politica sotto test si è utilizzato l'ambiente di esecuzione offerto da eXist-db soprattutto per eseguire le verifiche;

xSDResources contiene le risorse necessarie alla validazione delle richieste e delle politiche;

Inoltre è presente anche il package *comboTest.src.au.csiro.aehrc.combotest* che contiene la classe *ComboTester* che implementa l'algoritmo di generazione delle combinazioni con la tecnica combinatoriale; per intergare ComboTest è stato necessario modificare tale classe.

4.7.1 Funzionamento di X-CREATE

Prima di eseguire X-CREATE è necessario avere installato sulla propria macchina un server MySQL oppure essere collegati ad una rete dove è disponile un server MySQL. Nel file *xcreate.properties* presente nella cartella di lavoro, modificare i parametri tra parentesi angolate <> con i valori opportuni:

jdbc.url=jdbc:mysql://<hostname>/<dbname>
jdbc.username=<username>
jdbc.password=<password>

X-CREATE appena avviato effettua un controllo sulla consistenza della base di dati, controllando che tutte le tabelle siano già presenti; se manca almeno una tabella, ricrea l'intera base di dati. Inoltre crea, se non esiste, la cartella *Temp_ X-CREATE* dove verranno salvate tutte le test suite

generabili. In particolare il contenuto della cartella è organizzato in modo gerarchico seguendo la seguente struttura.

X-CREATE crea una cartella per ogni politica sotto test avente lo stesso nome della politica (senza estensione *.xml*) nella quale sono presenti 4 cartelle, una per ogni strategia di testing. Le cartelle delle strategie gerarchiche presentano a loro volta un contenuto strutturato basato sul nome del nodo dal quale sono generate le diverse test suite. I nomi di tali cartelle hanno il seguente formato: *pkSubtreeRootNode_* <*pkNodo*>, dove <*pkNodo*> rappresenta la chiave primaria del nodo all'interno della tabella relazionale *Nodi*. Inoltre quando viene aggiunta una politica, nella cartella riservata a tale politica viene memorizzato il file XML, risultato della fase di parsing della politica: il nome di tale file ha la seguente struttura *result_* <*nomePolitica*>. Questo file è un utile strumento di verifica dei criteri di selezione.

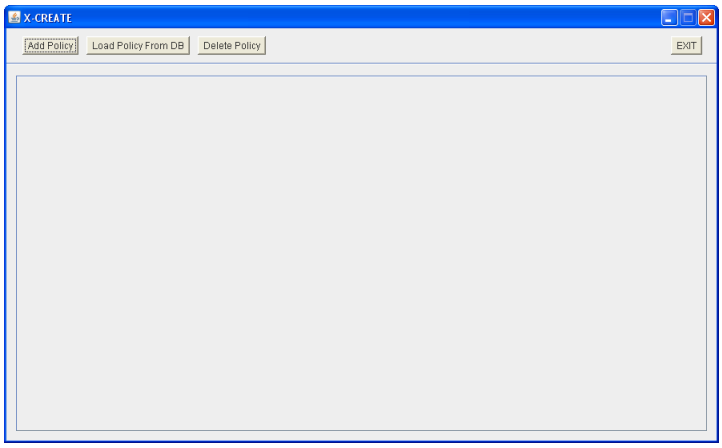

Figura 4.5: Welcome window.

X-CREATE si presenta come un'unica finestra dove nella parte alta (si veda la Figura 4.5) vi sono gli strumenti necessari ad aggiungere una politica (il pulsante *Add Policy*), reperire una politica già aggiunta in una fase precedente (il pulsante *Load Policy From DB*), cancellare una politica precedentemente aggiunta (il pulsante *Delete Policy*) ed infine, abbandonare l'applicazione (il pulsante *EXIT*).

Per generare casi di test per una politica i primo passo è renderla disponibile a X-CREATE; tramite il pulsante Add Policy si presenta un FileChooser per selezionare il file .xml che contiene la politica (Figura 4.6). Quando l'utente conferma la scelta X-CREATE verifica se l'istanza XML è valida rispetto allo Policy Schema invitandolo a riprovare l'inserimento di politica conforme con un opportuno messaggio qualora l'esito della verifica fosse negativo. Se l'istanza è valida si effettuano le seguenti operazioni:

1. creazione di una cartella con il nome della politica nella cartella *Temp_X-CREATE* all'interno della quale vengono create 4 cartella una per ogni

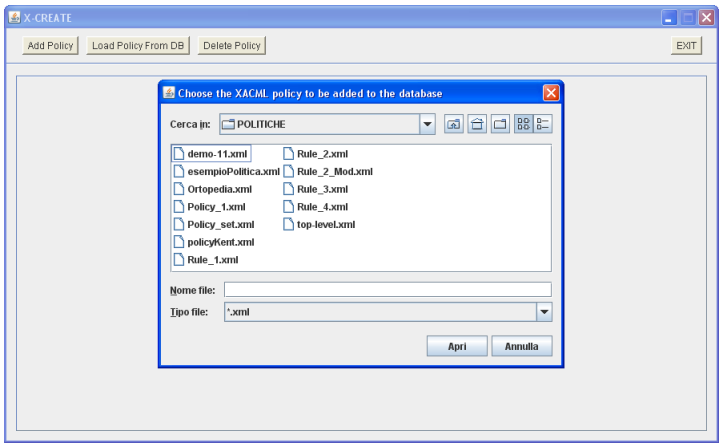

Figura 4.6: Add Policy.

strategia di generazione possibile;

2. caricamento della politica su eXist-db;
3. parsing della politica, cioè esecuzione della XQuery;
4. lettura del risultato, salvataggio del risultato nella cartella creata nel passo 1, e ricostruzione in memoria della struttura ad albero che rappresenta la politica contenente tutte le tuple;
5. aggiunta dei valori random nei nodi opportuni;
6. caricamento nella base di dati la politica, i nodi e le tuple;
7. generazione delle combinazioni a partire dal nodo radice della politica;
8. salvataggio delle combinazioni;
9. ricostruzione dell'albero della politica e la sua visualizzazione (Figura 4.8).

La fase di caricamento della politica può impiegare anche alcuni secondi per via della generazione delle combinazioni: il tempo di attesa è in qualche modo proporzionale alla cardinalità dell'insieme di combinazioni possibili; tale tempo non dipende direttamente da X-CREATE, ma di pende dal tool Combo-Test responsabile della generazione delle combinazioni.

Quando una politica è già stata caricata nella base di dati, può essere richiamata per la generazione di ulteriori test suite. In questo caso si agisce sul pulsante Load Policy e si presenta una lista di delle politiche presenti nella base di dati (Figura 4.7). In questo caso il tempo di reazione di X-CREATE è immediato in quanto le operazioni effettuate sono solo quelle di

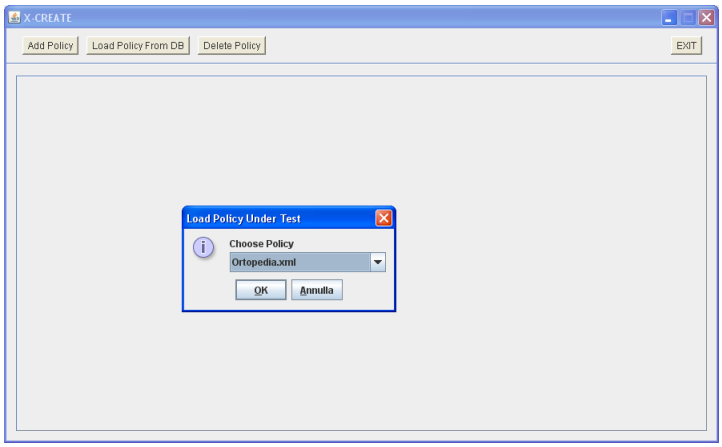

Figura 4.7: Load Policy.

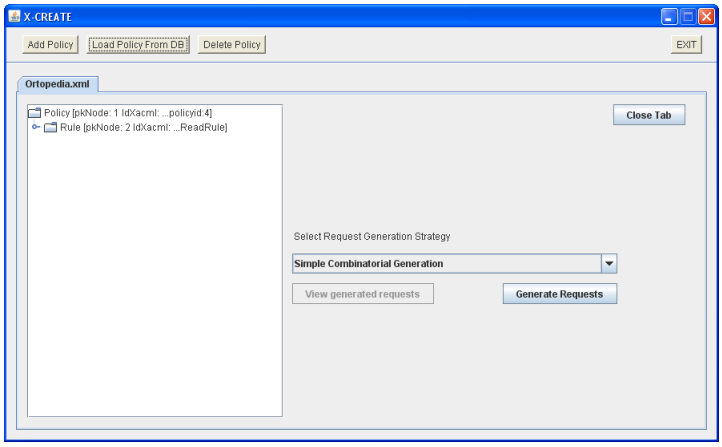

Figura 4.8: Added Policy Loaded Policy.

ricostruzione della gerarchia e l'aggiunta di una scheda relativa alla politica nella finestra principale (Figura 4.8).

Si possono caricare e aggiungere più politiche nella stessa finestra in modo da generare contemporaneamente più test suite per più politiche (Figura 4.9).

Come si può notare ogni scheda è formata da:

- una *etichetta* che riporta il nome della politica;
- un *Tree View* che visualizza la gerarchia della politica; ogni nodo è rappresentato da un elemento etichettato nel seguente modo "*NomeNodo [pkNode:*<*pkNodo*>* IdXacml: . . .* <*suffissoIdNodo*>*]*" dove <pkNodo> rappresenta la chiave primaria del nodo all'interno della tabella Nodi nella base di dati e <suffissoIdNodo> rappresenta gli ultimi 10 carat-

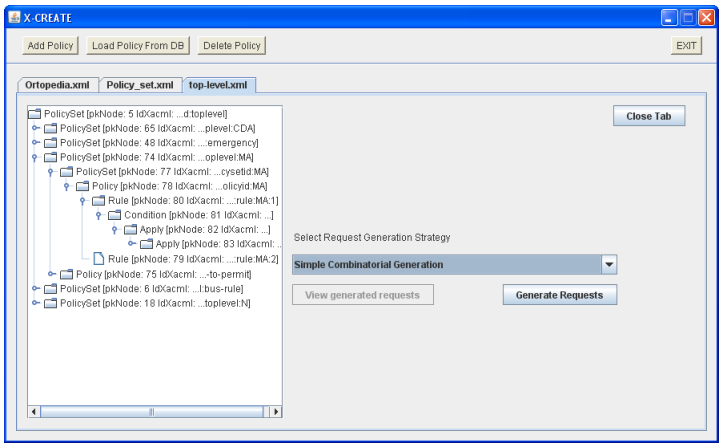

Figura 4.9: Esempio con più politiche.

teri dell'identificatore del nodo all'interno della politica XACML. Tale rappresentazione è molto utile nella fase di verifica della consistenza della basa di dati e della verifica della consistenza della gerarchia della politica. Inoltre permette all'utente di navigare tutta politica selezionando eventualmente un sotto albero radicato per la generazione dei casi di test per una particolare parte della politica;

- un *ComboBox* che permette di selezionare la strategia di generazione che si vuole adottare, e due pulsanti che offrono la possibilità di eseguire la strategia selezionata e visualizzare i risultati di tale esecuzione.

Capitolo 5

Conclusioni

Il tirocinio, condotto all'interno del Laboratorio di Ingegneria del Software dell'ISTI appartenente al CNR di Pisa, ha previsto la realizzazione di un framework per la generazione automatica e sistematica dei casi di test di politiche di controllo degli accessi. I casi di test sono le richieste che un ipotetico utente deve formulare per richiedere l'accesso ad una risorsa.

L'attività di tirocinio si è conclusa con la realizzazione dell'applicazione chiamata X-CREATE che permette di generare i casi di test a partire dal Context Schema al quale tutte le richieste devono essere conformi.

X-CREATE rappresenta il primo tentativo volto ad analizzare ed esplorare le potenzialità del Context Schema senza fare riferimento esplicito allo schema delle politiche. I casi di test sono derivati una volta per tutte sotto forma di strutture conformi allo schema del contesto di richieste prive di valori. Solo quando si vogliono istanziare tali richieste ad un dominio di una politica si analizza tale politica per estrarre dei valori significati ed assegnarli alle richieste in modo da avere delle richieste finali eseguibili.

Durante il tirocinio, soprattutto nella fase iniziale, è stato possibile raffinare alcuni requisiti che erano inconsistenti con le specifiche del linguaggio XACML. Inoltre, è stato possibile migliorare la strategia di generazione dei casi test. In particolare il contributo significativo è quello di permettere la generazione dei casi di test non soltanto volti al testing dell'intera politica ma volti anche ad una particolare sezione della politica. Questo si rende necessario nella fase di aggiornamento di una politica, modifica od estensione di una politica aggiungendo o rimuovendo vincoli.

In questa relazione sono stati presentati i principali aspetti progettuali e implementativi con particolare enfasi ai problemi che si sono affrontati relativamente ai requisiti generali del prodotto.

Sono stati discussi i problemi più importanti che si sono incontrati alcuni dei quali sono stati risolti e altri ancora aperti. Tali problemi potrebbero essere affrontati come sviluppi futuri dell'applicazione.

Altre attività svolte

Il tirocinio è iniziato affrontando un'altro problema: realizzare un'applicazione per la trasformazione di schemi XML in altri schemi XML. Tale applicazione doveva essere realizzata tramite l'integrazione di tool esistenti per la trasformazione di modelli sviluppati sulla piattaforma eclipse e concepire una metrica di valutazione della bontà delle trasformazioni.

Dopo una fase iniziale nella quale si è ricercato tali tool e dopo la valutazione e lo studio di un linguaggio per la trasformazione di modelli (il linguaggio ATL, Atlas Transformation Language) si è giunti alla conclusione che bisognava affrontare il problema da zero.

Dopo un attento studio del problema si è individuata l'area di ricerca per il trattamento di documenti XML che affronta il problema della trasformazione tra schemi XML: lo Schema Matching. Questo ha portato ad uno studio di fattibilità molto complesso e lungo terminato con la redazione di un documento che illustrava ad altissimo livello i principali requisiti che il sistema da realizzare deve rispettare.

L'analisi e la discussione di tale documento hanno permesso di giungere alla conclusione che lo studio di fattibilità ha portato fallimento del progetto per causa di una non chiare formulazione del problema e della indisponibilità di tempo: per poter risolvere il problema sarebbe necessario molto più tempo e molte più risorse umane. Il tempo e i vincoli di un tirocinio sarebbero bastati solo a definire i requisiti e forse ad condurre una progettazione ad altissimo livello. Quindi si è deciso di dirottare il tirocinio su un altro problema più chiaro e soprattutto fattibile per un tirocinio da 18 CFU.
Il progetto sul quale si è deciso di investire l'attività di tirocinio si manifesta nella realizzazione X-CREATE.

5.1 Sviluppi Futuri

X-CREATE è un prodotto facilmente estendibile grazie alla particolare attenzione effettuata sia nella fase di progettazione sia nella fase di implementazione. I problemi ancora aperti si possono risolvere intervenendo in piccole parti circoscritte nel codice sorgente. In particolare le strategia *Combinatorial Generation from Intermediate Requests* e la omonima gerarchica possono essere implementate nella classe *RequestsGeneratorFromIntermediateRequests* presente nel package *filler*.

L'interfaccia grafica può essere estesa con nuove funzionalità per la gestione delle richieste finali in modo da permetterne la visualizzazione ed eventualmente la modifica.

La possibilità di includere un PDP per eseguire i casi test. La Sun fornisce un'implementazione Java di un PDP [41] conforme alle specifiche XACML. Tale PDP può essere invocato da X-CREATE passandogli come parametri la

politica sotto test e la cartella dove risiedono i casi di test ed eventualmente una cartella dove memorizzare i risultati. Questo può essere fatto semplicemente aggiungendo un pulsante all'interfaccia grafica e attribuire al suo gestore la responsabilità di eseguire la test suite.

Si potrebbe includere anche un Oracolo per confrontare i risultati ottenuti con quelli attesi in modo da valutare la bontà delle test suite generate.

I valori casuali potrebbero essere sostituiti con valori appartenenti al dominio in cui è istanziato il sistema di controllo degli accessi, per esempio possono essere reperiti in una base di dati.

5.2 Competenze acquisite

Le competenze acquisite durante il tirocinio sono relative alla progettazione e implementazione di applicazioni in grado di automatizzare il processo di testing di sistemi basati su XML.

L'apprendimento del linguaggio XML e una buona parte della famiglia XML ha permesso di raggiungere gli obiettivi prefissati. In particolare, si è appreso l'XML Schema, il modello dei dati XML (XDM), il linguaggio XPath che permette la navigazione di documenti espressi in XML, il linguaggio XQuery con il quale è possibile interrogare fonti XML, eXist-db un sistema di gestione di base di dati nativo XML.

L'apprendimento del linguaggio XACML ha permesso di constatare come XML sia potente e flessibile.

Le attività condotte hanno portato a raffinate le tecniche di analisi e progettazione di un prodotto software, l'analisi e valutazione di tecnologie al fine del riutilizzo e infine una migliore conoscenza dell'ambiente di sviluppo Eclipse.

La prima fase dell'attività di tirocinio ha permesso di vedere molto da vicino il mondo della ricerca nel campo informatico e nella fattispecie dell'ingegneria del software.

Concludendo, l'esperienza di tirocinio è stata molto formativa sia a livello professionale sia a livello umano.

Appendice A

La sintassi XACML

A.1 Policy Schema

A.1.1 Elemento <PolicySet>

```
<xs:element name="PolicySet" type="xacml:PolicySetType"/>
<xs:complexType name="PolicySetType">
 <xs:sequence>
  <xs:element ref="xacml:Description" minOccurs="0"/>
  <xs:element ref="xacml:PolicySetDefaults" minOccurs="0"/>
  <xs:element ref="xacml:Target"/>
  <xs:choice minOccurs="0" maxOccurs="unbounded">
   <xs:element ref="xacml:PolicySet"/>
   <xs:element ref="xacml:Policy"/>
   <xs:element ref="xacml:PolicySetIdReference"/>
   <xs:element ref="xacml:PolicyIdReference"/>
   <xs:element ref="xacml:CombinerParameters"/>
   <xs:element ref="xacml:PolicyCombinerParameters"/>
   <xs:element ref="xacml:PolicySetCombinerParameters"/>
  </xs:choice>
  <xs:element ref="xacml:Obligations" minOccurs="0"/>
 </xs:sequence>
 <xs:attribute name="PolicySetId" type="xs:anyURI" use="required"/>
 <xs:attribute name="Version" type="xacml:VersionType" default="1.0"/>
 <xs:attribute name="PolicyCombiningAlgId" type="xs:anyURI" use="
     required"/>
</xs:complexType>
```

Codice A.1: Elemento <PolicySet>.

A.1.2 Elemento <Target>

```
<xs:element name="Target" type="xacml:TargetType"/>
  <xs:complexType name="TargetType">
    <xs:sequence>
      <xs:element ref="xacml:Subjects" minOccurs="0"/>
      <xs:element ref="xacml:Resources" minOccurs="0"/>
      <xs:element ref="xacml:Actions" minOccurs="0"/>
      <xs:element ref="xacml:Environments" minOccurs="0"/>
    </xs:sequence>
```

```
9   </xs:complexType>
```

Codice A.2: Elemento <Target>.

L'elemento `<Target>` definisce un insieme di

A.1.3 Elemento <Subjects>

```
1 <xs:element name="Subjects" type="xacml:SubjectsType"/>
2   <xs:complexType name="SubjectsType">
3     <xs:sequence>
4       <xs:element ref="xacml:Subject" maxOccurs="unbounded"/>
5     </xs:sequence>
6   </xs:complexType>
```

Codice A.3: Elemento <Subjects>.

A.1.4 Elemento <Subject>

```
1 <xs:element name="Subject" type="xacml:SubjectType"/>
2   <xs:complexType name="SubjectType">
3     <xs:sequence>
4       <xs:element ref="xacml:SubjectMatch" maxOccurs="unbounded"/>
5     </xs:sequence>
6   </xs:complexType>
```

Codice A.4: Elemento <Subject>.

A.1.5 Elemento <SubjectMatch>

```
1  <xs:element name="SubjectMatch" type="xacml:SubjectMatchType"/>
2    <xs:complexType name="SubjectMatchType">
3      <xs:sequence>
4        <xs:element ref="xacml:AttributeValue"/>
5        <xs:choice>
6          <xs:element ref="xacml:SubjectAttributeDesignator"/>
7          <xs:element ref="xacml:AttributeSelector"/>
8        </xs:choice>
9      </xs:sequence>
10     <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
11   </xs:complexType>
```

Codice A.5: Elemento <SubjectMatch>.

A.1.6 Elemento <Resources>

```
1 <xs:element name="Resources" type="xacml:ResourcesType"/>
2   <xs:complexType name="ResourcesType">
3     <xs:sequence>
4       <xs:element ref="xacml:Resource" maxOccurs="unbounded"/>
5     </xs:sequence>
6   </xs:complexType>
```

Codice A.6: Elemento <Resources>.

A.1.7 Elemento <Resource>

```
<xs:element name="Resource" type="xacml:ResourceType"/>
  <xs:complexType name="ResourceType">
    <xs:sequence>
      <xs:element ref="xacml:ResourceMatch" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
```

Codice A.7: Elemento <Resource>.

A.1.8 Elemento <ResourceMatch>

```
<xs:element name="ResourceMatch" type="xacml:ResourceMatchType"/>
  <xs:complexType name="ResourceMatchType">
    <xs:sequence>
      <xs:element ref="xacml:AttributeValue"/>
      <xs:choice>
        <xs:element ref="xacml:ResourceAttributeDesignator"/>
        <xs:element ref="xacml:AttributeSelector"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
  </xs:complexType>
```

Codice A.8: Elemento <ResourceMatch>.

A.1.9 Elemento <Actions>

```
<xs:element name="Actions" type="xacml:ActionsType"/>
  <xs:complexType name="ActionsType">
    <xs:sequence>
      <xs:element ref="xacml:Action" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
```

Codice A.9: Elemento <Actions>.

A.1.10 Elemento <Action>

```
<xs:element name="Action" type="xacml:ActionType"/>
  <xs:complexType name="ActionType">
    <xs:sequence>
      <xs:element ref="xacml:ActionMatch" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
```

Codice A.10: Elemento <Action>.

A.1.11 Elemento <ActionMatch>

```
<xs:element name="ActionMatch" type="xacml:ActionMatchType"/>
  <xs:complexType name="ActionMatchType">
    <xs:sequence>
      <xs:element ref="xacml:AttributeValue"/>
      <xs:choice>
        <xs:element ref="xacml:ActionAttributeDesignator"/>
        <xs:element ref="xacml:AttributeSelector"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
  </xs:complexType>
```

Codice A.11: Elemento <ActionMatch>.

A.1.12 Elemento <Environments>

```
<xs:element name="Environments" type="xacml:EnvironmentsType"/>
  <xs:complexType name="EnvironmentsType">
    <xs:sequence>
      <xs:element ref="xacml:Environment" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
```

Codice A.12: Elemento <Environments>.

A.1.13 Elemento <Environment>

```
<xs:element name="Environment" type="xacml:EnvironmentType"/>
  <xs:complexType name="EnvironmentType">
    <xs:sequence>
      <xs:element ref="xacml:EnvironmentMatch" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
```

Codice A.13: Elemento <Environment>.

A.1.14 Elemento <EnvironmentMatch>

```
<xs:element name="EnvironmentMatch" type="xacml:EnvironmentMatchType"/
    >
  <xs:complexType name="EnvironmentMatchType">
    <xs:sequence>
      <xs:element ref="xacml:AttributeValue"/>
      <xs:choice>
        <xs:element ref="xacml:EnvironmentAttributeDesignator"/>
        <xs:element ref="xacml:AttributeSelector"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute name="MatchId" type="xs:anyURI" use="required"/>
  </xs:complexType>
```

Codice A.14: Elemento <EnvironmentMatch>.

A.1.15 Elemento <Policy>

```
<xs:element name="Policy" type="xacml:PolicyType"/>
  <xs:complexType name="PolicyType">
    <xs:sequence>
      <xs:element ref="xacml:Description" minOccurs="0"/>
      <xs:element ref="xacml:PolicyDefaults" minOccurs="0"/>
      <xs:element ref="xacml:CombinerParameters" minOccurs="0"/>
      <xs:element ref="xacml:Target"/>
      <xs:choice maxOccurs="unbounded">
        <xs:element ref="xacml:CombinerParameters" minOccurs="0"/>
        <xs:element ref="xacml:RuleCombinerParameters" minOccurs="0"/>
        <xs:element ref="xacml:VariableDefinition"/>
        <xs:element ref="xacml:Rule"/>
      </xs:choice>
      <xs:element ref="xacml:Obligations" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="PolicyId" type="xs:anyURI" use="required"/>
    <xs:attribute name="Version" type="xacml:VersionType" default="1.0
        "/>
    <xs:attribute name="RuleCombiningAlgId" type="xs:anyURI" use="
        required"/>
  </xs:complexType>
```

Codice A.15: Elemento <Policy>.

A.1.16 Elemento <Rule>

```
<xs:element name="Rule" type="xacml:RuleType"/>
  <xs:complexType name="RuleType">
    <xs:sequence>
      <xs:element ref="xacml:Description" minOccurs="0"/>
      <xs:element ref="xacml:Target" minOccurs="0"/>
      <xs:element ref="xacml:Condition" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="RuleId" type="xs:string" use="required"/>
    <xs:attribute name="Effect" type="xacml:EffectType" use="required"
        />
  </xs:complexType>
```

Codice A.16: Elemento <Rule>.

`<Rule> <>`

A.1.17 Elemento <VariableDefinition>

```
<xs:element name="VariableDefinition" type="
    xacml:VariableDefinitionType"/>
  <xs:complexType name="VariableDefinitionType">
    <xs:sequence>
      <xs:element ref="xacml:Expression"/>
    </xs:sequence>
    <xs:attribute name="VariableId" type="xs:string" use="required"/>
  </xs:complexType>
```

Codice A.17: Elemento <VariableDefinition>.

A.1.18 Elemento <VariableReference>

```
<xs:element name="VariableReference" type="xacml:VariableReferenceType
    " substitutionGroup="xacml:Expression"/>
  <xs:complexType name="VariableReferenceType">
    <xs:complexContent>
      <xs:extension base="xacml:ExpressionType">
        <xs:attribute name="VariableId" type="xs:string" use="required
            "/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

Codice A.18: Elemento <VariableReference>.

A.1.19 Elemento <Expression>

```
<xs:element name="Expression" type="xacml:ExpressionType" abstract="
    true"/>
  <xs:complexType name="ExpressionType" abstract="true"/>
```

Codice A.19: Elemento <Expression>.

A.1.20 Elemento <Condition>

```
<xs:element name="Condition" type="xacml:ConditionType"/>
  <xs:complexType name="ConditionType">
    <xs:sequence>
      <xs:element ref="xacml:Expression"/>
    </xs:sequence>
  </xs:complexType>
```

Codice A.20: Elemento <Condition>.

A.1.21 Elemento <Apply>

```
<xs:element name="Apply" type="xacml:ApplyType" substitutionGroup="
    xacml:Expression"/>
  <xs:complexType name="ApplyType">
    <xs:complexContent>
      <xs:extension base="xacml:ExpressionType">
        <xs:sequence>
          <xs:element ref="xacml:Expression" minOccurs="0" maxOccurs="
              unbounded"/>
        </xs:sequence>
        <xs:attribute name="FunctionId" type="xs:anyURI" use="required
            "/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

Codice A.21: Elemento <Apply>.

A.1.22 Elemento <Function>

```
1 <xs:element name="Function" type="xacml:FunctionType"
      substitutionGroup="xacml:Expression"/>
2   <xs:complexType name="FunctionType">
3     <xs:complexContent>
4       <xs:extension base="xacml:ExpressionType">
5         <xs:attribute name="FunctionId" type="xs:anyURI" use="required
              "/>
6       </xs:extension>
7     </xs:complexContent>
8   </xs:complexType>
```

Codice A.22: Elemento <Function>.

A.1.23 Elemento <SubjectAttributeDesignator>

```
1 <xs:element name="SubjectAttributeDesignator" type="
      xacml:SubjectAttributeDesignatorType" substitutionGroup="
      xacml:Expression"/>
2   <xs:complexType name="SubjectAttributeDesignatorType">
3     <xs:complexContent>
4       <xs:extension base="xacml:AttributeDesignatorType">
5         <xs:attribute name="SubjectCategory" type="xs:anyURI" use="
              optional" default="urn:oasis:names:tc:xacml:1.0:subject-
              category:access-subject"/>
6       </xs:extension>
7     </xs:complexContent>
8   </xs:complexType>
```

Codice A.23: Elemento <SubjectAttributeDesignator>.

A.1.24 Elemento <ResourceAttributeDesignator>

```
1   <xs:element name="ResourceAttributeDesignator" type="
        xacml:AttributeDesignatorType" substitutionGroup="
        xacml:Expression"/>
```

Codice A.24: Elemento <ResourceAttributeDesignator>.

A.1.25 Elemento <ActionAttributeDesignator>

```
1 <xs:element name="ActionAttributeDesignator" type="
      xacml:AttributeDesignatorType" substitutionGroup="xacml:Expression
      "/>
```

Codice A.25: Elemento <ActionAttributeDesignator>.

A.1.26 Elemento <EnvironmentAttributeDesignator>

```
<xs:element name="EnvironmentAttributeDesignator" type="
    xacml:AttributeDesignatorType" substitutionGroup="xacml:Expression
    "/>
```

Codice A.26: Elemento <EnvironmentAttributeDesignator>.

A.1.27 Elemento <AttributeSelector>

```
<xs:element name="AttributeSelector" type="xacml:AttributeSelectorType
    " substitutionGroup="xacml:Expression"/>
  <xs:complexType name="AttributeSelectorType">
    <xs:complexContent>
      <xs:extension base="xacml:ExpressionType">
        <xs:attribute name="RequestContextPath" type="xs:string" use="
            required"/>
        <xs:attribute name="DataType" type="xs:anyURI" use="required"/
            >
        <xs:attribute name="MustBePresent" type="xs:boolean" use="
            optional" default="false"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

Codice A.27: Elemento <AttributeSelector>.

A.1.28 Elemento <AttributeValue>

```
<xs:element name="AttributeValue" type="xacml:AttributeValueType"
    substitutionGroup="xacml:Expression"/>
  <xs:complexType name="AttributeValueType" mixed="true">
    <xs:complexContent mixed="true">
      <xs:extension base="xacml:ExpressionType">
        <xs:sequence>
          <xs:any namespace="##any" processContents="lax" minOccurs="0
              " maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="DataType" type="xs:anyURI" use="required"/
            >
        <xs:anyAttribute namespace="##any" processContents="lax"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
```

Codice A.28: Elemento <AttributeValue>.

A.2 Context Schema

A.2.1 Elemento <Request>

```
<xs:element name="Request" type="xacml-context:RequestType" />
    <xs:complexType name="RequestType">
        <xs:sequence>
            <xs:element ref="xacml-context:Subject" maxOccurs="
                unbounded" />
            <xs:element ref="xacml-context:Resource" maxOccurs="
                unbounded" />
            <xs:element ref="xacml-context:Action" />
            <xs:element ref="xacml-context:Environment" />
        </xs:sequence>
    </xs:complexType>
```

Codice A.29: Elemento <Request>.

A.2.2 Elemento <Subject>

```
<xs:element name="Subject" type="xacml-context:SubjectType" />
    <xs:complexType name="SubjectType">
        <xs:sequence>
            <xs:element ref="xacml-context:Attribute" minOccurs="0"
                maxOccurs="unbounded" />
        </xs:sequence>
        <xs:attribute name="SubjectCategory" type="xs:anyURI" default=
            "urn:oasis:names:tc:xacml:1.0:subject-category:access-
            subject" />
    </xs:complexType>
```

Codice A.30: Elemento <Subject>.

A.2.3 Elemento <Resource>

```
<xs:element name="Resource" type="xacml-context:ResourceType" />
    <xs:complexType name="ResourceType">
        <xs:sequence>
            <xs:element ref="xacml-context:ResourceContent" minOccurs=
                "0" />
            <xs:element ref="xacml-context:Attribute" minOccurs="0"
                maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
```

Codice A.31: Elemento <Resource>.

A.2.4 Elemento <ResourceContent>

```
<xs:element name="ResourceContent" type="xacml-
    context:ResourceContentType" />
    <xs:complexType name="ResourceContentType" mixed="true">
        <xs:sequence>
            <xs:any minOccurs="0" maxOccurs="unbounded" namespace="##
                any" processContents="lax" />
        </xs:sequence>
        <xs:anyAttribute namespace="##any" processContents="lax" />
    </xs:complexType>
```

Codice A.32: Elemento <ResourceContent>.

A.2.5 Elemento <Action>

```
<xs:element name="Action" type="xacml-context:ActionType" />
    <xs:complexType name="ActionType">
        <xs:sequence>
            <xs:element ref="xacml-context:Attribute" minOccurs="0"
                maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
```

Codice A.33: Elemento <Action>.

A.2.6 Elemento <Environment>

```
 <xs:element name="Environment" type="xacml-context:EnvironmentType" /
     >
    <xs:complexType name="EnvironmentType">
        <xs:sequence>
            <xs:element ref="xacml-context:Attribute" minOccurs="0"
                maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
```

Codice A.34: Elemento <Environment>.

A.2.7 Elemento <Attribute>

```
<xs:element name="Attribute" type="xacml-context:AttributeType" />
    <xs:complexType name="AttributeType">
        <xs:sequence>
            <xs:element ref="xacml-context:AttributeValue" maxOccurs="
                unbounded" />
        </xs:sequence>
        <xs:attribute name="AttributeId" type="xs:anyURI" use="
            required" />
        <xs:attribute name="DataType" type="xs:anyURI" use="required"
            />
        <xs:attribute name="Issuer" type="xs:string" use="optional" />
    </xs:complexType>
```

Codice A.35: Elemento <Attribute>.

A.2.8 Elemento <AttributeValue>

```
    <xs:element name="AttributeValue" type="xacml-
        context:AttributeValueType" />
    <xs:complexType name="AttributeValueType" mixed="true">
        <xs:sequence>
```

```
4             <xs:any minOccurs="0" maxOccurs="unbounded" namespace="##
                  any" processContents="lax" />
5         </xs:sequence>
6         <xs:anyAttribute namespace="##any" processContents="lax" />
7     </xs:complexType>
```

Codice A.36: Elemento <AttributeValue>.

A.2.9 Elemento <Response>

```
1     <xs:element name="Response" type="xacml-context:ResponseType" />
2     <xs:complexType name="ResponseType">
3         <xs:sequence>
4             <xs:element ref="xacml-context:Result" maxOccurs="
                  unbounded" />
5         </xs:sequence>
6     </xs:complexType
```

Codice A.37: Elemento <Response>.

A.2.10 Elemento <Result>

```
1     <xs:element name="Result" type="xacml-context:ResultType" />
2     <xs:complexType name="ResultType">
3         <xs:sequence>
4             <xs:element ref="xacml-context:Decision" />
5             <xs:element ref="xacml-context:Status" minOccurs="0" />
6             <xs:element ref="xacml:Obligations" minOccurs="0" />
7         </xs:sequence>
8         <xs:attribute name="ResourceId" type="xs:string" use="optional
              " />
9     </xs:complexType>
```

Codice A.38: Elemento <Result>.

A.2.11 Elemento <Decision>

```
1     <xs:element name="Decision" type="xacml-context:DecisionType" />
2     <xs:simpleType name="DecisionType">
3         <xs:restriction base="xs:string">
4             <xs:enumeration value="Permit" />
5             <xs:enumeration value="Deny" />
6             <xs:enumeration value="Indeterminate" />
7             <xs:enumeration value="NotApplicable" />
8         </xs:restriction>
9     </xs:simpleType>
```

Codice A.39: Elemento <Decision>.

Appendice B

Esempi

```
<?xml version="1.0" encoding="utf-8"?>
<Request ... >
 <Subject SubjectCategory="">
  <Attribute AttributeId="" DataType="" Issuer="">
   <AttributeValue> </AttributeValue>
  </Attribute>
  <Attribute AttributeId="" DataType="" Issuer="">
   <AttributeValue> </AttributeValue>
  </Attribute>
 </Subject>
 <Resource>
  <Attribute AttributeId="" DataType="" Issuer="">
   <AttributeValue> </AttributeValue>
  </Attribute>
  <Attribute AttributeId="" DataType="" Issuer="">
   <AttributeValue> </AttributeValue>
  </Attribute>
  <Attribute AttributeId="" DataType="" Issuer="">
   <AttributeValue> </AttributeValue>
 </Resource>
 <Action>
  <Attribute AttributeId="" DataType="" Issuer="">
   <AttributeValue> </AttributeValue>
  </Attribute>
  <Attribute AttributeId="" DataType="" Issuer="">
   <AttributeValue> </AttributeValue>
  </Attribute>
  <Attribute AttributeId="" DataType="" Issuer="">
   <AttributeValue> </AttributeValue>
  </Attribute>
 </Action>
 <Environment>
  <Attribute AttributeId="" DataType="" Issuer="">
   <AttributeValue> </AttributeValue>
  </Attribute>
  <Attribute AttributeId="" DataType="" Issuer="">
   <AttributeValue> </AttributeValue>
  </Attribute>
  <Attribute AttributeId="" DataType="" Issuer="">
   <AttributeValue> </AttributeValue>
  </Attribute>
  <Attribute AttributeId="" DataType="" Issuer="">
   <AttributeValue> </AttributeValue>
```

```
44    </Attribute>
45   </Environment>
46  </Request>
```

Codice B.1: Richiesta intermedia 1.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <Request ... >
3   <Subject SubjectCategory="">
4    <Attribute AttributeId="" DataType="" Issuer="">
5     <AttributeValue> </AttributeValue>
6    </Attribute>
7    <Attribute AttributeId="" DataType="" Issuer="">
8     <AttributeValue> </AttributeValue>
9    </Attribute>
10   <Attribute AttributeId="" DataType="" Issuer="">
11    <AttributeValue> </AttributeValue>
12   </Attribute>
13  </Subject>
14  <Resource>
15   <Attribute AttributeId="" DataType="" Issuer="">
16    <AttributeValue> </AttributeValue>
17   </Attribute>
18   <Attribute AttributeId="" DataType="" Issuer="">
19    <AttributeValue> </AttributeValue>
20   </Attribute>
21  </Resource>
22  <Action>
23   <Attribute AttributeId="" DataType="" Issuer="">
24    <AttributeValue> </AttributeValue>
25   </Attribute>
26   <Attribute AttributeId="" DataType="" Issuer="">
27    <AttributeValue> </AttributeValue>
28   </Attribute>
29   <Attribute AttributeId="" DataType="" Issuer="">
30    <AttributeValue> </AttributeValue>
31   </Attribute>
32   <Attribute AttributeId="" DataType="" Issuer="">
33    <AttributeValue> </AttributeValue>
34   </Attribute>
35  </Action>
36  <Environment>
37   <Attribute AttributeId="" DataType="" Issuer="">
38    <AttributeValue> </AttributeValue>
39   </Attribute>
40   <Attribute AttributeId="" DataType="" Issuer="">
41    <AttributeValue> </AttributeValue>
42   </Attribute>
43  </Environment>
44 </Request>
```

Codice B.2: Richiesta intermedia 2.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <Request ... >
3   <Subject SubjectCategory="">
4    <Attribute AttributeId="ruolo" DataType="string" Issuer="">
5     <AttributeValue>paziente</AttributeValue>
6    </Attribute>
7    <Attribute AttributeId="ruolo" DataType="string" Issuer="">
8     <AttributeValue>RandomValueS1</AttributeValue>
```

```
 </Attribute>
</Subject>
<Resource>
 <Attribute AttributeId="resourceID" DataType="string" Issuer="">
  <AttributeValue>cartellaClinica</AttributeValue>
 </Attribute>
 <Attribute AttributeId="resourceID" DataType="string" Issuer="">
  <AttributeValue>RandomValueR1</AttributeValue>
 </Attribute>
 <Attribute AttributeId="" DataType="" Issuer="">
  <AttributeValue> </AttributeValue>
</Resource>
<Action>
 <Attribute AttributeId="actionID" DataType="string" Issuer="">
  <AttributeValue>lettura</AttributeValue>
 </Attribute>
 <Attribute AttributeId="actionID" DataType="string" Issuer="">
  <AttributeValue>RandomValueA1</AttributeValue>
 </Attribute>
 <Attribute AttributeId="" DataType="" Issuer="">
  <AttributeValue> </AttributeValue>
 </Attribute>
</Action>
<Environment>
 <Attribute AttributeId="envID" DataType="dayTimeDuration" Issuer="">
  <AttributeValue>P7D</AttributeValue>
 </Attribute>
 <Attribute AttributeId="envID" DataType="dayTimeDuration" Issuer="">
  <AttributeValue>RandomValueE1</AttributeValue>
 </Attribute>
 <Attribute AttributeId="" DataType="" Issuer="">
  <AttributeValue> </AttributeValue>
 </Attribute>
 <Attribute AttributeId="" DataType="" Issuer="">
  <AttributeValue> </AttributeValue>
 </Attribute>
</Environment>
</Request>
```

Codice B.3: Richiesta finale 1.

```
<?xml version="1.0" encoding="utf-8"?>
<Request ... >
 <Subject SubjectCategory="">
  <Attribute AttributeId="ruolo" DataType="string" Issuer="">
   <AttributeValue>RandomValueS1</AttributeValue>
  </Attribute>
  <Attribute AttributeId="ruolo" DataType="string" Issuer="">
   <AttributeValue>paziente</AttributeValue>
  </Attribute>
  <Attribute AttributeId="" DataType="" Issuer="">
   <AttributeValue> </AttributeValue>
  </Attribute>
 </Subject>
 <Resource>
  <Attribute AttributeId="resourceID" DataType="string" Issuer="">
   <AttributeValue>RandomValueR1</AttributeValue>
  </Attribute>
  <Attribute AttributeId="resourceID" DataType="string" Issuer="">
   <AttributeValue>cartellaClinica</AttributeValue>
  </Attribute>
 </Resource>
```

```
22  <Action>
23   <Attribute AttributeId="actionID" DataType="string" Issuer="">
24    <AttributeValue>RandomValueA1</AttributeValue>
25   </Attribute>
26   <Attribute AttributeId="actionID" DataType="string" Issuer="">
27    <AttributeValue>lettura</AttributeValue>
28   </Attribute>
29   <Attribute AttributeId="" DataType="" Issuer="">
30    <AttributeValue> </AttributeValue>
31   </Attribute>
32   <Attribute AttributeId="" DataType="" Issuer="">
33    <AttributeValue> </AttributeValue>
34   </Attribute>
35  </Action>
36  <Environment>
37   <Attribute AttributeId="envID" DataType="dayTimeDuration" Issuer="">
38    <AttributeValue>RandomValueE1</AttributeValue>
39   </Attribute>
40   <Attribute AttributeId="envID" DataType="dayTimeDuration" Issuer="">
41    <AttributeValue>P7D</AttributeValue>
42   </Attribute>
43  </Environment>
44 </Request>
```

Codice B.4: Richiesta finale 2.

Bibliografia

[1] MySQL ,RDBMS. http://dev.mysql.com/.

[2] Pairwise Testing, Combinatorial Test Case Generation. http://www.pairwise.org/tools.asp.

[3] Creating a gui with jfc/swing. http://download.oracle.com/javase/tutorial/uiswing/.

[4] Eclipse. http://www.eclipse.org/.

[5] Java technology. http://www.sun.com/java/.

[6] Jdbc(tm) database access. http://download.oracle.com/javase/tutorial/jdbc/index.html.

[7] Sgml, standard generalized markup language. http://www.w3.org/MarkUp/SGML/.

[8] Xml, extensible markup language. http://www.w3.org/XML/.

[9] Xml path language (xpath) 2.0. http://www.w3.org/TR/xpath20/.

[10] Xml schema. http://www.w3.org/XML/Schema.

[11] Xquery 1.0: An xml query language. http://www.w3.org/TR/xquery/.

[12] Xquery 1.0 and xpath 2.0 data model (xdm). http://www.w3.org/TR/xpath-datamodel/.

[13] P. Ammann, P. Black, and W. Majurski. Using model checking to generate tests from specifications. In *Proc. of ICFEM*, pages 46–54, 1998.

[14] A. Bertolino, J. Gao, E. Marchetti, and A. Polini. Automatic test data generation for XML schema-based partition testing. In *Proc. of AST*, May 2007.

[15] A. Bertolino, J. Gao, E. Marchetti, and A. Polini. TAXI – A Tool for XML-Based Testing. In *Proc. of ICSE Companion*, pages 53–54, 2007.

[16] A. Bertolino, F. Lonetti, and E. Marchetti. Systematic XACML request generation for testing purposes. *Technical report*, 2010.

[17] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG system: An approach to testing based on combinatiorial design. *IEEE Trans. on Soft. Eng.*, 23(7):437–444, July 1997.

[18] R. DeMillo, R. Lipton, and F. Sayward. Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41, 1978.

[19] K. Fisler, S. Krishnamurthi, L. Meyerovich, and M. Tschantz. Verification and change-impact analysis of access-control policies. In *Proc. of ICSE*, pages 196–205. ACM New York, NY, USA, 2005.

[20] A. Gargantini and C. Heitmeyer. Using model checking to generate tests from requirements specifications. In *In Proc. Joint 7th ESEC and 7th ACM SIGSOFT FSE*, pages 146–162, 1999.

[21] G. Hughes and T. Bultan. Automated verification of access control policies. *Technical Report 2004-22. Comp. Science Dep., Univ. of California, Santa Barbara, CA*, 2004.

[22] J. Hwang, E. Martin, T. Xie, and V. Hu. Policy-Based Testing, 2008. sub. for publ. in the Enc. of Soft. Eng., available at http://ase.csc.ncsu.edu/projects/policy/.

[23] Y. Le Traon, T. Mouelhi, A. Pretschner, and B. Baudry. Test-Driven Assessment of Access Control in Legacy Applications. In *Proc. of ICST*, pages 238–247, 2008.

[24] K. Li, L. Mounier, and R. Groz. Test generation from security policies specified in or-BAC. In *Proc. of COMPSAC*, pages 255–260, 2007.

[25] N. Li, J. Hwang, and T. Xie. Multiple-implementation testing for XACML implementations. In *Proc. of TAV-WEB*, pages 27–33, 2008.

[26] N. Li, J. Hwang, and T. Xie. Multiple-implementation testing for xacml implementations. In *Proc. of TAV-WEB*, pages 27–33, New York, NY, USA, 2008. ACM.

[27] W. Mallouli, J. Orset, A. Cavalli, N. Cuppens, and F. Cuppens. A formal approach for testing security rules. In *Proc. of ACMT*, page 132, 2007.

[28] E. Martin and T. Xie. Automated test generation for access control policies. In *Supplemental Proc. of ISSRE*, November 2006.

[29] E. Martin and T. Xie. Automated test generation for access control policies via change-impact analysis. In *Proc. of SESS*, pages 5–11, May 2007.

[30] E. Martin and T. Xie. A fault model and mutation testing of access control policies. In *Proc. of WWW*, pages 667–676, May 2007.

[31] E. Martin, T. Xie, and T. Yu. Defining and measuring policy coverage in testing access control policies. In *Proc. of ICICS*, pages 139–158, December 2006.

[32] A. Masood, A. Ghafoor, and A. Mathur. Test Generation for Access Control Systems that Employ RBAC Policies. Technical report, SERC-TR-283, Purdue University, 2005.

[33] A. Masood, A. Ghafoor, and A. Mathur. Scalable and effective test generation for access control systems that employ RBAC policies. Technical report, CERIAS TR 2006-24, Purdue University, 2006.

[34] W. Meier. exist: An open source native xml database. Darmstadt University of Technology.

[35] OASIS. extensible access control markup language (xacml) version 2.0. http://www.oasis-open.org/committees/xacml/, February 2003.

[36] T. Ostrand and M. Balcer. The category-partition method for specifying and generating functional tests. *Communications of ACM*, 31(6):676–686, June 1988.

[37] T. J. Ostrand and M. J. Balcer. The category-partition method for specifying and generating functional tests. *Commun. ACM*, 31(6):676–686, 1988.

[38] L. Pantieri and T. Gordini. L'arte di scrivere con LaTeX. http://www.guit.sssup.it/, Luglio 2010.

[39] A. Pretschner, T. Mouelhi, and Y. Le Traon. Model-based tests for access control policies. In *Proc. of ICST*, pages 338–347, 2008.

[40] C. Stanbridge and J. Ryan-Brown. Combotest - introducing combinatorial methods for automating the generation of test suites.

[41] Sun Microsystems. Sun's XACML Implementation. http://sunxacml.sourceforge.net/, 2006.

[42] Y. Traon, T. Mouelhi, and B. Baudry. Testing security policies: going beyond functional testing. In *Proc. of ISSRE*, pages 93–102, 2007.

[43] N. Zhang, M. Ryan, and D. Guelev. Evaluating access control policies through model checking. *LNCS*, 3650:446, 2005.