



UNIVERSITÀ  
DEGLI STUDI  
FIRENZE

Scuola di Scienze Matematiche, Fisiche e Naturali  
Corso di Laurea in Informatica

Tesi di Laurea

ESPRIMERE IN FACPL POLITICHE DI  
CONTROLLO DEGLI ACCESSI BASATE SUL  
COMPORTAMENTO PASSATO

EXPRESSING ACCESS CONTROL POLICIES  
BASED ON PAST BEHAVIOR IN FACPL

FEDERICO SCHIPANI

Relatore: *Rosario Pugliese*  
Correlatore: *Andrea Margheri*

Anno Accademico 2014-2015



---

## INDICE

---

1	INTRODUZIONE	9
2	ACCESS CONTROL E USAGE CONTROL	11
2.1	Access Control	11
2.2	Usage Control	17
2.2.1	Caso di studio 1: Gestione lettura e scrittura di file	18
2.2.2	Caso di studio 2: Noleggio e acquisto di contenuti	20
3	FORMAL ACCESS CONTROL POLICY LANGUAGE	23
3.1	Tool	23
3.2	Processo di valutazione	24
3.3	Sintassi	26
3.4	Semantica	28
3.5	Esempi di politiche	29
4	IMPLEMENTARE USAGE CONTROL IN FACPL	33
4.1	Estensione del processo di Valutazione	33
4.2	Estensione Linguistica	35
4.3	Semantica	36
4.4	Formalizzazione dei case study	37
4.4.1	Accesso ai file	38
4.4.2	Noleggio e acquisto di contenuti	41
5	ESTENSIONE DELLA LIBRERIA FACPL	47
5.1	Implementazione Stato	47
5.1.1	Status Attribute	49
5.1.2	Funzioni su Status Attribute	49
5.1.3	Estensione del PEP	50
5.2	Estensione delle politiche	52
5.2.1	Funzioni di controllo su status	54
5.2.2	Obligation	55
5.3	Plugin Eclipse	57
5.4	Esempi	57
5.4.1	Primo case study	57
5.4.2	Secondo case study	60
6	CONCLUSIONI	61
6.1	Sviluppi futuri	62

## 2    Indice

A	CODICE COMPLETO	67
A.1	Codice del Capitolo 4	67
A.2	Codice del Capitolo 5	71

---

## ELENCO DELLE FIGURE

---

Figura 1	Access Control List (ACL) in OS X	13
Figura 2	Gruppo in OS X	14
Figura 3	Scenario ABAC base	15
Figura 4	Insieme dei componenti di $U\text{CON}_{ABC}$	18
Figura 5	Diagramma di flusso del case study sull'accesso dei file	19
Figura 6	Diagramma di flusso del case study sul noleggio e acquisto di contenuti	21
Figura 7	ToolChain di FACPL	24
Figura 8	Il processo di valutazione di FACPL	25
Figura 9	Valutazione dopo lo stato	34
Figura 10	Grafico UML delle classi Status e StatusAttribute	48
Figura 11	Grafico UML per la gerarchia di funzioni aritmetiche	51
Figura 12	Contesto	53
Figura 13	Comparatori	54



---

## LISTA DI CODICI

---

3.1	Esempio di politica in FACPL . . . . .	29
3.2	Richieste per Codice 3.1 . . . . .	30
4.1	Esempio per la sintassi . . . . .	37
4.2	Target PolicySet . . . . .	38
4.3	Policy Write . . . . .	38
4.4	Policy StopWrite . . . . .	39
4.5	Richieste del primo esempio . . . . .	39
4.6	Secondo Esempio . . . . .	41
4.7	Secondo Esempio . . . . .	43
4.8	Richieste del Secondo Esempio . . . . .	43
5.1	Stralcio della classe Status . . . . .	47
5.2	Set Attribute della classe Status . . . . .	48
5.3	Classe Status Attribute . . . . .	49
5.4	Interfaccia per le operazioni . . . . .	50
5.5	Add Status . . . . .	50
5.6	Discharge delle Fulfilled Obligation di stato . . . . .	52
5.7	Discharge delle Fulfilled Obligation di stato . . . . .	53
5.8	Parte rifattorizzata del metodo che si occupa del fulfilling . . . . .	55
5.9	CreateObligation nelle status . . . . .	55
5.10	CreateObligation nelle normali . . . . .	55
5.11	Peculiarità della classe FulfilledObligationStatus . . . . .	56
5.12	Discharge delle Fulfilled Obligation di stato . . . . .	57
5.13	Esempio di richiesta . . . . .	57
5.14	Esempio di stato . . . . .	58
5.15	Policy Write . . . . .	59
A.1	Policy set completo di 4.4.1 . . . . .	67
A.2	Policy set completo di 4.4.2 . . . . .	68
A.3	Richiesta di Sezione 5.4.1 . . . . .	71
A.4	Richiesta di Sezione 5.4.1 . . . . .	72
A.5	Richiesta di Sezione 5.4.1 . . . . .	73
A.6	Richiesta di Sezione 5.4.1 . . . . .	74
A.7	Richiesta di Sezione 5.4.1 . . . . .	75

## 6    Lista di Codici

A.8	Richiesta di Sezione 5.4.1 . . . . .	76
A.9	Richiesta di Sezione 5.4.1 . . . . .	77
A.10	Main di Sezione 5.4.1 . . . . .	78
A.11	Policy di Sezione 5.4.1 . . . . .	79
A.12	Status 5.4.1 . . . . .	84



*Ezechiele 25:17. "Il cammino dell'uomo timorato è minacciato da ogni parte dalle iniquità degli esseri egoisti e dalla tirannia degli uomini malvagi. Benedetto sia colui che nel nome della carità e della buona volontà conduce i deboli attraverso la valle delle tenebre, perché egli è in verità il pastore di suo fratello e il ricercatore dei figli smarriti. E la mia giustizia calerà sopra di loro con grandissima vendetta e furiosissimo sdegno su coloro che si proveranno ad ammorbare e infine a distruggere i miei fratelli. E tu saprai che il mio nome è quello del Signore quando farò calare la mia vendetta sopra di te."*  
— Jules Winnfield [Voce di Luca Ward]

*"Stay hungry, stay hungry"*  
— Paolo Bitta, l'uomo chiamato contratto



---

## INTRODUZIONE

---

I sistemi informatici si sono diffusi molto rapidamente, e grazie all'avvento di nuove tecnologie, come la rete, la condivisione di dati e risorse è diventata alla portata di tutti. Proteggere queste risorse da accessi indesiderati è diventato molto importante, motivo per cui negli ultimi decenni la questione del *Access Control* è diventata sempre più rilevante. In particolare, a partire dal 1970, sono stati proposti vari modelli (come ACL, Role Based Access Control (RBAC), Attribute Based Access Control (ABAC) e Policy Based Access Control (PBAC)), ognuno dei quali ha i suoi pro e contro.

Il primo modello proposto è ACL ed ha come obiettivo il controllo delle risorse di un sistema operativo. Questo modello si basa su liste di accesso associate ad ogni risorsa che definiscono le regole di accesso; è un modello molto semplice da implementare, ma quando i dati sono eccessivi diventa poco efficiente. Successivamente è stato introdotto un modello basato su ruoli, ovvero dei gruppi di utenti suddivisi secondo caratteristiche comuni, chiamato RBAC, il quale associa a degli insiemi di risorse dei diritti di accesso. Un punto debole di questo modello è l'assenza di costrutti che permettono la definizione delle regole che rende difficile assegnare permessi particolari a singole risorse. A seguire è stato utilizzato un modello basato su attributi chiamato ABAC, dove le decisioni vengono prese valutando caratteristiche del richiedente, della risorsa e dell'ambiente in cui è stata fatta la richiesta. Quest'ultimo modello tuttavia non offre una buona scalabilità. Per porre rimedio a questo difetto è stato introdotto un ulteriore modello, che si chiama PBAC il quale risulta essere uno dei più utilizzati ed è basato su politiche, ovvero un insieme di regole su attributi.

Recentemente, Sandhu e Park [13] hanno introdotto *Usage Control* che permette di prendere decisioni durante l'accesso e di basarsi sul comportamento passato in fase di valutazione di una richiesta. Questa caratteristica lo rende molto adatto ad ambienti come il Web, il Cloud o

in generale legati in qualche modo alla rete.

Formal Access Control Policy Language (FACPL) è un linguaggio basato su eXtensible Access Control Markup Language (XACML), supportato da una libreria scritta in Java. Rispetto a XACML la sintassi di FACPL è molto più semplice e concisa, quindi permette di formalizzare in modo facile e rapido politiche di *access control*. Tuttavia, FACPL non gode di caratteristiche per eseguire richieste ed ottenere risposte valutando anche il comportamento passato, quindi l'obiettivo di questa tesi è stato quello di estenderlo in modo da implementare nuove e fondamentali funzionalità che permettono di sfruttare caratteristiche tipiche di *Usage Control*.

La tesi, dopo questa breve introduzione, è organizzata in questo modo:

- Nel Capitolo 2 vengono presentati in maggior dettaglio tutti i modelli di *Access Control* ed inoltre viene dedicata una sezione a *Usage Control* dove vengono anche introdotti due esempi che verranno portati avanti durante gli altri capitoli.
- Nel Capitolo 3 è descritto FACPL e vengono riportati alcuni esempi che mostrano le problematiche per cui non è possibile farne un uso a livello di *Usage Control*.
- Nel Capitolo 4 viene trattata l'estensione di FACPL ad un livello sintattico e semantico, spiegando quali nuove componenti sono state introdotte ed analizzandone il loro significato ed utilizzo attraverso dei casi di studio.
- Nel Capitolo 4 sono trattati gli argomenti visti in quello precedente ma dal punto di vista dell'estensione della libreria Java.
- Nel Capitolo 6 viene riassunto tutto il lavoro svolto e sono proposte idee per gli sviluppi futuri.
- In Appendice A viene proposto il codice completo.

---

## ACCESS CONTROL E USAGE CONTROL

---

I sistemi informatici moderni sono capaci di condividere dati e risorse computazionali, ed impedire accessi non autorizzati è diventata una priorità inderogabile. I motivi che portano a questa decisione possono essere legati alla privacy o alla consistenza dei dati durante una computazione. Ad esempio, molte informazioni personali possono essere raccolte durante alcune attività quotidiane, dunque diventa necessario proteggere questi dati da malintenzionati. Per questo motivo esistono sistemi di Access Control, ovvero dei sistemi definiti da un insieme di condizioni che permettono di creare una prima linea difensiva contro accessi indesiderati.

Nella prima parte di questo capitolo vengono trattati in maggior dettaglio i vari modelli di controllo all'accesso proposti negli ultimi decenni, infine viene analizzato anche un nuovo modello chiamato *Usage Control*. Infine nell'ultima sezione vengono proposti due case study.

### 2.1 ACCESS CONTROL

Negli anni sono stati proposti diversi approcci per cercare di definire un modello efficiente e scalabile. Andremo ora ad eseguire una classificazione dei modelli di Access Control, seguendo la catalogazione del NIST [10].

- Access Control List (ACL) è il primo di questi modelli ed è stato proposto intorno al 1970 spinto dall'avvento dei primi sistemi multi utente.
- Role Based Access Control (RBAC) è nato successivamente ad ACL e ne modifica alcuni aspetti in modo da rimuovere molte delle limitazioni di quest'ultimo.

- Attribute Based Access Control (ABAC) è nato dopo RBAC e fornisce un paradigma dinamico che aiuta a sviluppare policy che si adattano all'ambiente.
- Policy Based Access Control (PBAC) è molto simile ad ABAC ma migliora e standardizza quest ultimo modello.

### *Access Control Lists*

ACL è il primo modello di controlli agli accessi, è stato introdotto intorno al 1970 grazie all'avvento dei sistemi multi utente allo scopo di limitare l'accesso a file e dati appartenenti a diversi utenti, infatti i primi sistemi ad utilizzare questo modello sono stati sistemi di tipo UNIX. Con la comparsa della multiutenza per sistemi ad uso personale lo standard ACL è stato implementato in molte più ambienti come sistemi UNIX-Like e Windows.

Il concetto dietro ACL è uno dei più semplici, in quanto ogni risorsa del sistema che deve essere controllata ha una sua lista, che ad ogni soggetto associa le azioni che può effettuare sulla risorsa, ed il sistema operativo quando viene fatta richiesta decide in base alla lista se dare il permesso o meno.

Per esempio in Figura 1, si può vedere come *test\_folder* sia la risorsa da controllare, *federicoschipani*, *staff* e *everyone* siano i soggetti e le azioni associate sono, in questo caso, *Read & Write* al primo soggetto e *Read only* agli altri due.

La semplicità di questo modello non richiede grandi infrastrutture sottostanti, la sua implementazione dal punto di vista applicativo risulta abbastanza semplice attraverso l'uso di linguaggi ad alto livello come Python o Java, poiché le strutture che servono per implementare questo standard sono già definite. Nonostante negli anni sono stati sviluppati modelli più complessi, come ABAC, PBAC e RBAC, il sistema descritto in questa sezione viene comunque usato nei sistemi operativi recenti. Come si può vedere in Figura 1 OS X sfrutta questo standard per la gestione dei permessi sul filesystem.

Un aspetto negativo si manifesta quando si trattano grandi quantità di risorse. Ogni volta che viene richiesto l'accesso ad una risorsa da parte di un entità, utente o applicazione che sia, è necessario verificare nella lista associata, il che lo rende abbastanza oneroso dal punto di vista computazionale.

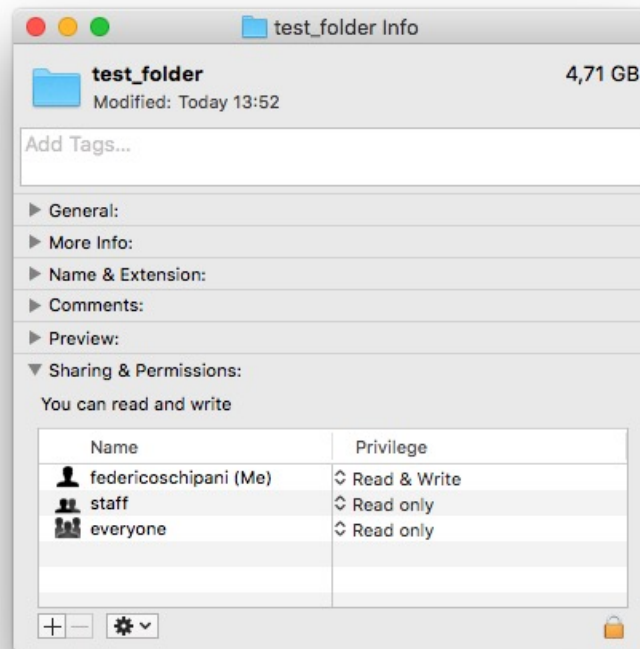


Figura 1.: Access Control List (ACL) in OS X

Un altro lato negativo emerge quando bisogna effettuare modifiche ai permessi di una determinata risorsa, in quanto è necessario andare ad operare sulla lista di quest'ultima, il che rende questo compito incline ad errori ed oneroso dal punto di vista del tempo.

### *Role-based Access Control*

RBAC è l'evoluzione di ACL, in quanto tende a correggerne alcuni difetti come la limitata scalabilità.

A differenza di ACL è presente il concetto il ruolo del richiedente, ovvero un titolo che definisce un livello di autorità, che raggrupperà diversi utenti in una categoria. RBAC attraverso questa nuova caratteristica riesce a porre rimedio ai difetti di ACL in quanto questo tipo di gestione offre il vantaggio di facilitare l'assegnazione dei permessi, poiché per ogni risorsa non si devono più gestire tutti i singoli utenti, ma basta gestire i permessi associati a queste nuove categorie.

Un utente può anche far parte di più gruppi, per esempio un contabile di un'azienda può far parte del gruppo *impiegati* e *contabili* in modo da permettergli l'accesso sia ai documenti riservati ai soli impiegati che quelli riservati ai soli contabili. Come si può vedere in Figura 2 il concetto di gruppo è implementato nei sistemi operativi moderni, in particolare in OS X, Windows e sistemi UNIX-Like.

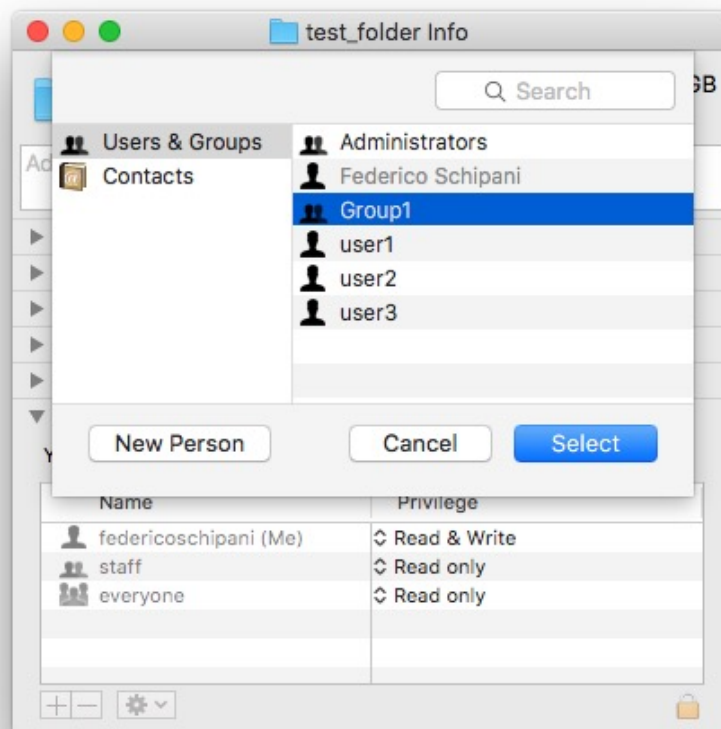


Figura 2.: Gruppo in OS X

RBAC però ha i suoi difetti, uno dei più evidenti è l'impossibilità di gestire le autorizzazioni a livello di singola persona, ed è quindi necessario creare diversi gruppi o trovare altri escamotage per autorizzare, o vietare, singoli utenti appartenenti a determinati gruppi.



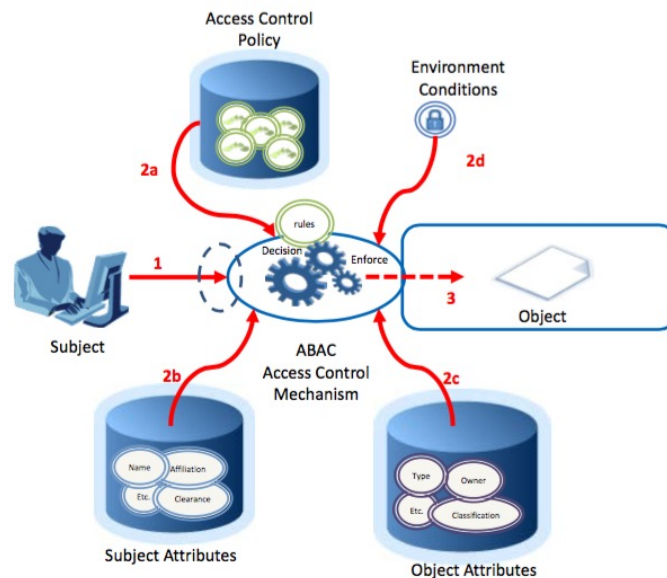


Figura 3.: Scenario ABAC base

### *Attribute-based Access Control*

ABAC è un modello di controllo all'accesso nel quale le decisioni sono prese in base ad un insieme di attributi, associazioni con il richiedente, ambiente e risorsa stessa. In questa nuova architettura è presente un nuovo componente chiamato Policy Decision Point (PDP) il quale si occuperà della valutazione delle richieste prima di fornire una decisione finale. Ogni attributo è un campo distinto dagli altri che il PDP compara con un insieme di valori per determinare o meno l'accesso alla risorsa.

In Figura 3 viene mostrato il funzionamento di ABAC. Di particolare rilevanza è il secondo step, che porta alla raccolta di tutte le informazioni per produrre una decisione finale. Inizialmente il sistema richiede al Access Control Policy (ACS) le policy, successivamente effettua la raccolta degli attributi dalle varie fonti. Questi attributi possono provenire da disparate fonti ed essere di svariati tipi. Per esempio nella valutazione di una richiesta possono essere considerati attributi come la data di assunzione di un dipendente ed il suo grado all'interno dell'azienda.

Un vantaggio di ABAC è che non c'è la necessità che il richiedente conosca in anticipo la risorsa o il sistema a cui dovrà accedere. Fino a quando gli attributi che il richiedente fornisce coincidono con i requisiti dettati dalle policy l'accesso sarà garantito. ABAC perciò è utilizzato in situazioni in cui i proprietari delle risorse vogliono far accedere utenti che

non conoscono direttamente a patto che però rispettino i criteri preposti, il che rende il tutto molto più dinamico rispetto ad ACL e RBAC

Diversamente da ACL e RBAC questo tipo di controllo agli accessi non è implementato nei sistemi operativi, ma è largamente usato a livello applicativo. Spesso si usano applicazioni intermedie per mediare gli accessi da parte degli utenti a specifiche risorse. Implementazioni semplici di questo modello non richiedono grandi *database* o altre infrastrutture, tuttavia in ambienti dove non basta una semplice applicazione c'è necessità di grandi banche di dati.

Una limitazione di ABAC è che in grandi ambienti, con tante risorse, individui e applicazioni ci saranno molte regole, ed organizzarle in maniera efficiente diventa un compito oneroso.

### *Policy-based Access Control (PBAC)*

PBAC è stato sviluppato per far fronte alle problematiche di organizzazione delle regole di ABAC, infatti è una sua naturale evoluzione e tende ad uniformare ed armonizzare il sistema di controllo accessi. Questo modello cerca di aiutare le imprese a indirizzarsi verso la necessità di implementare un sistema di controllo agli accessi basato su policy.

PBAC combina attributi dalle risorse, dall'ambiente e dal richiedente con informazioni su determinate circostanze sotto le quali la richiesta è stata effettuata ed inoltre si serve di ruoli per determinare quando garantire l'accesso.

Nei sistemi ABAC gli attributi richiesti per avere accesso ad una particolare risorsa sono determinati a livello locale e possono variare da organizzazione ad organizzazione. Per esempio, un'unità organizzativa può determinare che l'accesso ad un archivio di documenti sensibili è semplicemente soggetto a richiesta di credenziali e ruolo particolare. Un'altra unità invece, oltre a richiedere credenziali e ruolo, richiede anche un certificato. Se un documento viene trasferito dal secondo al primo archivio perde la protezione fornita da quest'ultimo e sarà soggetto solo alla richiesta di credenziali e ruolo. Con PBAC invece si ha un solo punto dove vengono gestite le policy, e queste policy verranno valutate ad ogni tentativo di accedere alla risorsa.

PBAC quindi è un sistema molto più complicato di ABAC e perciò richiede il dislocamento di infrastrutture molto più onerose dal punto di vista economico che includono *database*, *directory service* e altri applicativi di mediazione e gestione. PBAC non richiede solo un'applicazione per gestire la valutazione delle policy, ma anche un sistema per la scrittura di

queste ultime in modo che non risultino ambigue. Uno standard basato su eXtensible Markup Language (XML), e che si chiama XACML, è sviluppato in modo tale da creare policy facilmente leggibili da una macchina.

Sfortunatamente però, queste policy non sono facili da scrivere e l'uso di XACML non necessariamente rende facile il processo di creazione, specifica e valutazione corretta di una policy.

Ci vorrebbe anche un modo per assicurarsi che tutti gli utenti di un sistema utilizzino lo stesso insieme di attributi, piuttosto arduo da realizzare. Gli attributi dovrebbero essere forniti da un'entità chiamata Authoritative Attribute Source (AAS) che, oltre a fare da sorgente per gli attributi, deve anche occuparsi della loro consistenza. In più bisogna instaurare un meccanismo per verificare che questi attributi provengano realmente dall'AAS. Può sembrare facile fare una cosa del genere, ma bisogna considerare il caso in cui più aziende lavorano insieme e devono implementare un sistema di controllo degli accessi in comune. Un problema si può verificare quando un'azienda valuta la gestione dell'AAS tramite una particolare *repository*, ma un'altra azienda non è d'accordo a questo tipo di soluzione

## 2.2 USAGE CONTROL

Oggi sono presenti differenti tipi di sistemi diversi che richiedono un modello più flessibile e continuativo per gestire la sicurezza. Questa sezione parlerà di un nuovo sistema, chiamato *Usage Control* [7].

*Usage Control* si propone come un nuovo e promettente approccio per il controllo degli accessi. In particolare verrà trattato il modello, inizialmente proposto da Sandhu e Park[7], chiamato  $U\text{CON}_{ABC}$ .

$U\text{CON}_{ABC}$  utilizza un approccio diverso rispetto a *Access Control* in quanto, rispetto a quest'ultimo, si riesce ad avere una continuità delle decisioni sull'accesso. Ciò vuol dire che le decisioni non vengono più prese decisioni solo a priori, ma anche durante l'accesso. Quindi, se durante l'utilizzo, qualche attributo di stato cambia e la *policy* non è più soddisfatta viene revocato l'accesso. Di conseguenza è richiesto un componente che modella lo stato del sistema, in modo tale da effettuare valutazioni in base a quelle effettuate in precedenza.

Il vantaggio di *Usage Control* è la sua capacità di adattarsi a vari casi di utilizzo, riuscendo così a includere e migliorare sistemi come ACL, RBAC, ABAC e PBAC descritti in 2.1. Il passaggio da Access Control a Usage Control è importante soprattutto quando si va a considerare ambienti legati alla rete, come possono essere il web o il cloud.

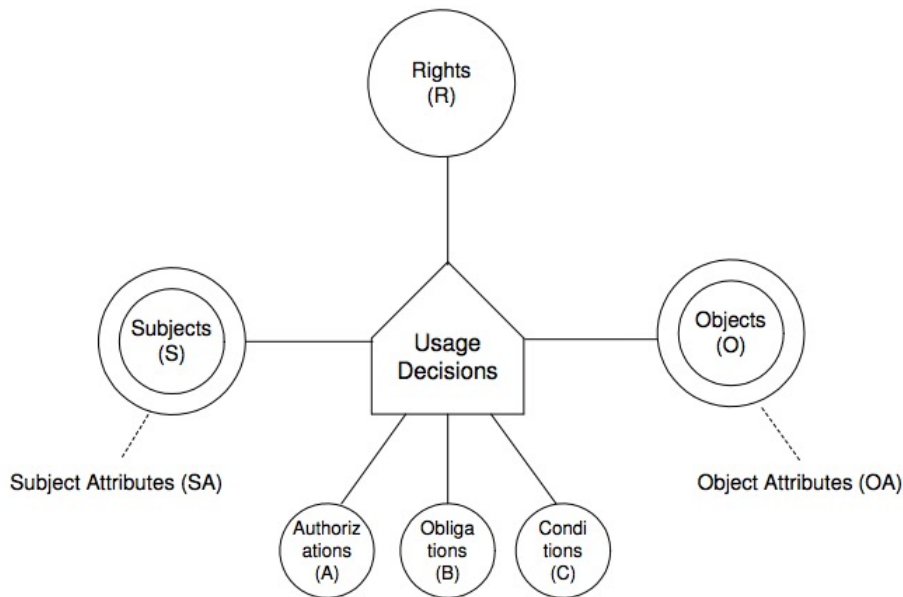


Figura 4.: Insieme dei componenti di  $UCON_{ABC}$

Il processo decisionale in Usage Control è diviso in due fasi [6].

- La prima fase è una fase di *pre decision* che fondamentalemente è la classica decisione presa in *Access Control*, questa decisione viene presa al momento in cui è effettuata la prima richiesta per produrre la decisione di accesso.
- La seconda fase è chiamata *ongoing decision*, ed è un processo che implementa il concetto di continuità in quanto le decisioni vengono prese durante l'accesso.

I componenti necessari a questo tipo di processo decisionale sono dei predicati, che possono essere di tre tipi, *authorizations*, *conditions* oppure *rights*, delle azioni chiamate *obligations* che devono essere eseguite durante l'accesso ed infine uno stato. Come questi componenti contribuiscono a prendere una decisione viene mostrato in Figura 4.

### 2.2.1 Caso di studio 1: Gestione lettura e scrittura di file

Dentro ad un sistema ci sono vari file, ai quali per questione di consistenza, bisogna limitare l'accesso. La regola è: "per un determinato file un massimo di due persone possono accedere in lettura oppure solo una persona può

*accedere in scrittura*". In Figura 5 viene mostrato un diagramma di flusso che sintetizza, e permette di capire meglio, la regola di accesso. Quando arriva la richiesta viene verificata la presenza nel database del richiedente e del file richiesto, se entrambe danno esito positivo si passa all'analisi dei requisiti per soddisfare la richiesta.

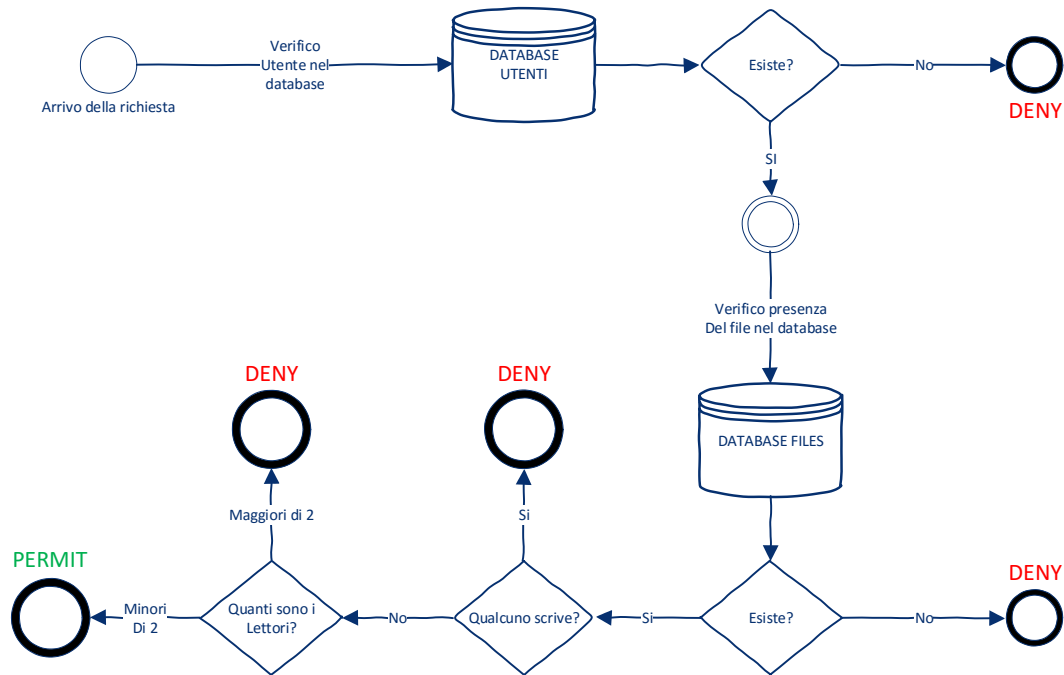


Figura 5.: Diagramma di flusso del case study sull'accesso dei file

Se in un primo momento nessuno sta visualizzando o scrivendo un determinato file, ed un utente generico chiederà l'accesso in lettura per questo file, ovviamente il responso sarà positivo in quanto non viola la regola preposta prima.

Dopo un po' di tempo, mentre il primo sta ancora leggendo, un altro utente chiede l'accesso in scrittura, che gli viene negato. In un istante di tempo successivo il primo utente sta continuando a leggere, ed anche il secondo utente vuole leggere. In questo caso viene dato responso positivo.

Infine, entrambi gli utenti smettono di leggere, ma uno di loro vuole apportare una modifica, allora richiede l'accesso in scrittura, che questa volta gli viene consentito poiché nessuno sta leggendo.

### 2.2.2 Caso di studio 2: Noleggio e acquisto di contenuti

Un altro utilizzo possibile di *Usage Control* riguarda l'analisi del comportamento passato. Un'azienda fornisce ai propri clienti la possibilità di effettuare noleggi o acquisti di contenuti multimediali (musica, video, film, serie tv e via scorrendo).

In caso il contenuto fosse stato acquistato, l'acquirente potrà ottenere l'accesso infinite volte per infinito tempo. Nel caso di noleggio invece saranno presenti delle condizioni, come per esempio un numero massimo di visioni che quando superato non permette più di fruire del contenuto o una data di scadenza che, una volta oltrepassata, impedirà l'ulteriore visione del file noleggiato in precedenza.

Come nell'esempio precedente viene mostrato un diagramma di flusso, proposto Figura 6, che permette di capire meglio il funzionamento questo sistema di *Usage Control*. Innanzitutto viene verificata la presenza nei due relativi database dell'utente e del file richiesto, successivamente viene analizzata la richiesta, che può essere di tre tipi.

- Visione: nel caso la richiesta fosse di visione viene verificato se realmente l'utente ha diritto ad avere accesso a quella risorsa, e di conseguenza viene presa una decisione.
- Acquisto: nel caso la richiesta fosse di acquisto verrà accreditato l'acquisto all'utente che ha effettuato la richiesta.
- Noleggio: per questa forma ci sono due diverse tipologie, il noleggio a tempo e il noleggio a numero di visualizzazioni. Nel primo caso all'utente sarà concesso di vedere il file per un determinato periodo di tempo, mentre nel secondo il richiedente potrà visionare il file per un numero limitato di volte.

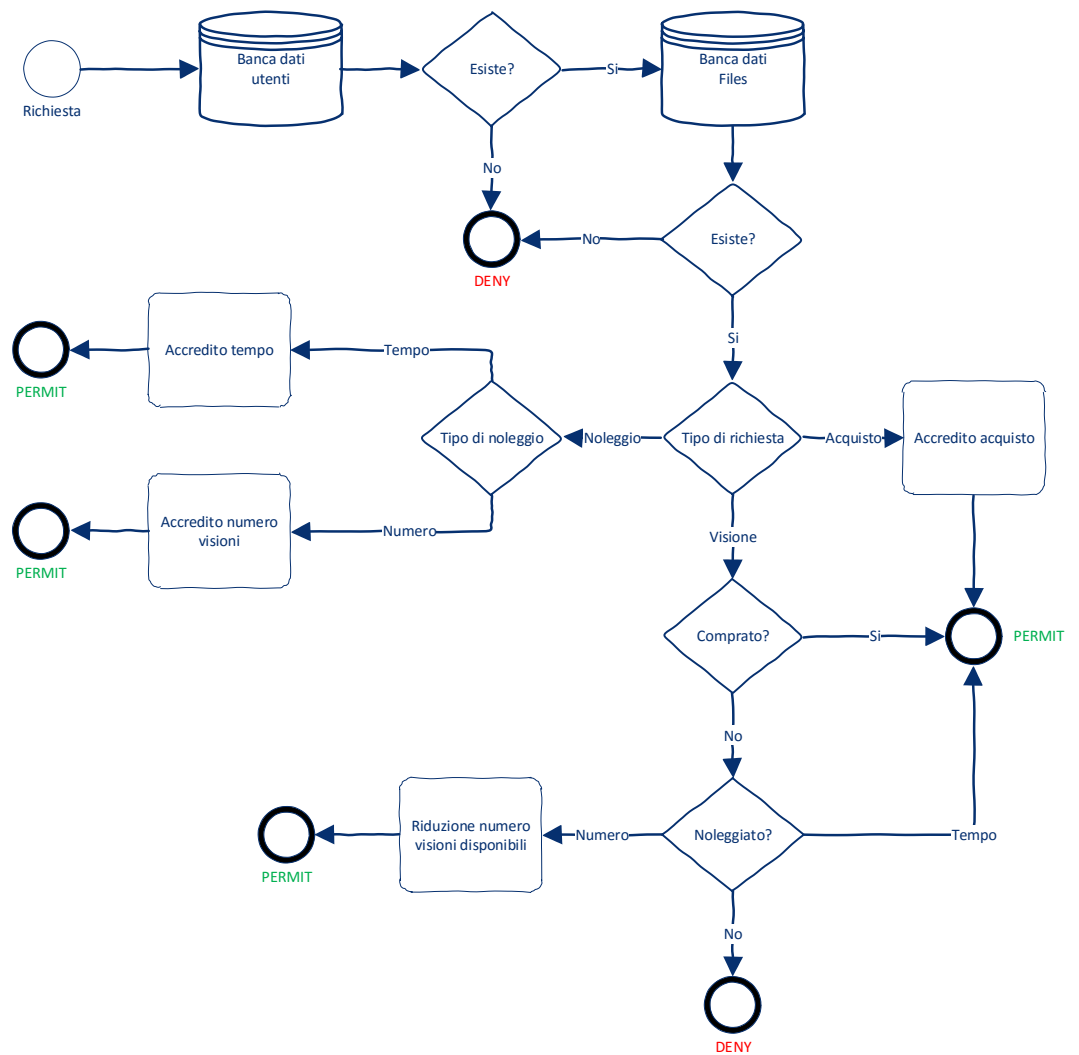


Figura 6.: Diagramma di flusso del case study sul noleggio e acquisto di contenuti





---

## FORMAL ACCESS CONTROL POLICY LANGUAGE

---

Negli anni molti linguaggi sono stati proposti per definire policy di access control. Uno di questi è eXtensible Access Control Markup Language (XACML) di OASIS, e la sua prima versione risale al 2003. Questo linguaggio ha una sintassi basata su XML e fornisce caratteristiche avanzate per l'*Access Control*. Il problema fondamentale di XACML è che non ha una sintassi facile da leggere e da scrivere.

Formal Access Control Policy Language (FACPL) è un linguaggio per formalizzare policy di access control supportato da una solida libreria Java ed utilizzabile attraverso un plugin per il famoso ambiente di sviluppo Eclipse. L'obiettivo di FACPL è definire una sintassi alternativa per XACML in modo da renderlo più agevole da usare. FACPL quindi è parzialmente ispirato a XACML, ma oltre ad introdurre una nuova sintassi ridefinisce alcuni aspetti aggiungendo nuove caratteristiche. Il suo scopo però non è sostituire XACML, ma fornire un linguaggio compatto ed espressivo per facilitare le tecniche di analisi attraverso tool specifici.

In Sezione 3.1 sono descritti i tool necessari per l'utilizzo di FACPL. Nella Sezione 3.2 è effettuata una disamina sul processo di valutazione di FACPL, nella quale sono presentati i componenti principali e la descrizione dell'interazione tra di essi. Nelle Sezioni 3.3 e 3.4 viene analizzata rispettivamente la sintassi e la semantica di FACPL. In Sezione 3.5 è proposto un esempio di politica con FACPL ed inoltre è spiegato in maggior dettaglio perché non è possibile prendere decisioni basate sul comportamento passato.

### 3.1 TOOL

Il plugin per Eclipse di FACPL è stato scritto con l'ausilio di un *Framework* chiamato *Xtext*. Quest'ultimo a sua volta utilizza *Xtend* poiché permette di scrivere regole di traduzione per generare automaticamente codice Java.

Attraverso il plugin si riesce a rendere Eclipse un vero e proprio ambiente di sviluppo per FACPL in quanto si ottengono funzioni come l'autocompletamento o l'highlighting del codice.

L'ambiente di sviluppo con il relativo plugin, le regole di traduzione, e la libreria Java formano insieme la *toolchain* di FACPL, mostrata in Figura 7.

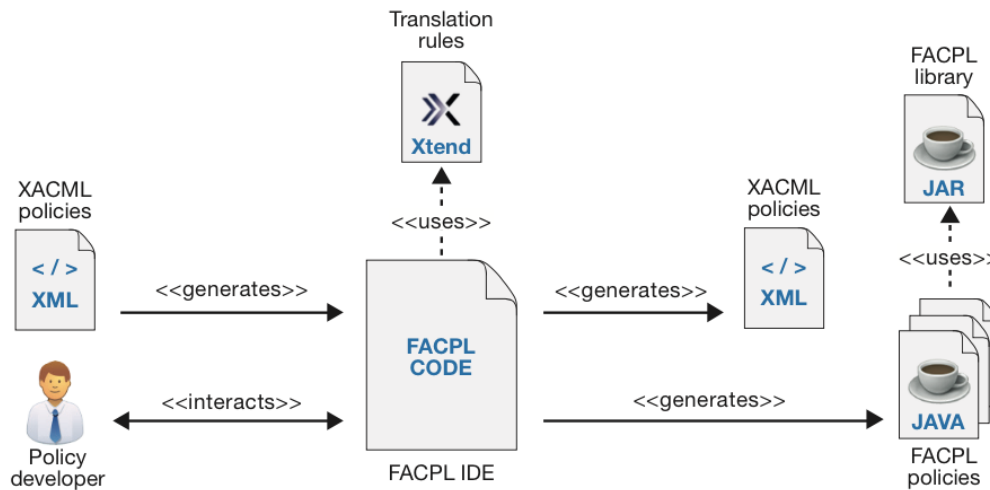


Figura 7.: ToolChain di FACPL

Lo sviluppatore che utilizza l'Integrated Development Environment (IDE) di FACPL può generare codice Java o XACML a partire da politiche scritte in FACPL. La traduzione da codice FACPL a Java o XACML è effettuata attraverso le regole di traduzione scritte in Xtend.

### 3.2 PROCESSO DI VALUTAZIONE

In figura 8 è mostrato il processo di valutazione delle policy definite in FACPL. I componenti principali sono tre:

- Policy Repository (PR)
- Policy Decision Point (PDP)
- Policy Enforcement Point (PEP)

Le policy sono memorizzate nel PR, il quale le rende disponibili al PDP che deciderà, successivamente, se garantire l'accesso o meno (Primo step). Nello step 2, quando il PEP riceve una richiesta, le credenziali di quest'ultima vengono codificate in una sequenza di attributi (ogni

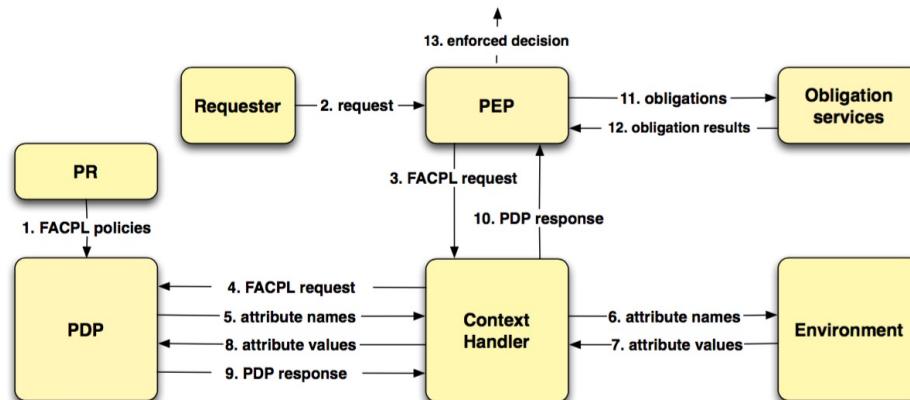


Figura 8.: Il processo di valutazione di FACPL

attributo è una coppia stringa valore) che, nello step 3, andranno a loro volta a formare una FACPL Request. Al quarto step il *context handler* aggiungerà attributi di ambiente (per esempio l'ora di ricezione della richiesta) e manderà la richiesta al PDP. A questo punto il PDP, tra il quinto e l'ottavo step, valuterà la richiesta e fornirà un risultato, il quale può eventualmente contenere delle *obligations*. La decisione del PDP può essere di quattro tipi:

- permit
- deny
- not-applicable
- indeterminate.

Il significato delle prime due decisioni è facilmente intuibile, mentre per le ultime due vuol dire che c'è stato un errore durante la valutazione. Gli errori possono essere di diverso tipo, e vengono gestiti attraverso algoritmi che combinano le decisioni delle varie policy per ottenere un risultato finale. Le *obligations* sono azioni, eseguite dal PEP, correlate al sistema di controllo degli accessi. Queste azioni possono essere di svariati tipi, come per esempio generare un file di log, o mandare una mail. Allo step 13, sulla base del risultato delle *obligations*, il acPEP esegue un processo chiamato *Enforcement* il quale restituirà un'altra decisione. Quest'ultima decisione corrisponde alla decisione finale del sistema e può differire da quella del PDP.

Tabella 1.: Sintassi di FACPL

<b>Policy Authorisation Systems</b>	$PAS ::= (pep : EnfAlg \text{ } pdp : PDP)$
<b>Enforcement algorithms</b>	$EnfAlg ::= base \mid deny\text{-}biased \mid permit\text{-}biased$
<b>Policy Decision Points</b>	$PDP ::= \{Alg \text{ } policies : Policy^+\}$
<b>Combining algorithms</b>	$Alg ::= p\text{-}over \mid d\text{-}over \mid d\text{-}unless\text{-}p \mid p\text{-}unless\text{-}d$ $\mid first\text{-}app \mid one\text{-}app \mid weak\text{-}con \mid strong\text{-}con$
<b>Policies</b>	$Policy ::= (Effect \text{ } target : Expr \text{ } obl : Obligation^*)$ $\mid \{Alg \text{ } target : Expr \text{ } policies : Policy^+ \text{ } obl : Obligation^*\}$
<b>Effects</b>	$Effect ::= permit \mid deny$
<b>Obligations</b>	$Obligation ::= [Effect \text{ } ObType \text{ } PepAction(Expr^*)]$
<b>Obligation Types</b>	$ObType ::= M \mid O$
<b>Expressions</b>	$Expr ::= Name \mid Value$ $\mid and(Expr, Expr) \mid or(Expr, Expr) \mid not(Expr)$ $\mid equal(Expr, Expr) \mid in(Expr, Expr)$ $\mid greater\text{-}than(Expr, Expr) \mid add(Expr, Expr)$ $\mid subtract(Expr, Expr) \mid divide(Expr, Expr)$ $\mid multiply(Expr, Expr)$
<b>Attribute Names</b>	$Name ::= Identifier/Identifier$
<b>Literal Values</b>	$Value ::= true \mid false \mid Double \mid String \mid Date$
<b>Requests</b>	$Request ::= (Name, Value)^+$

### 3.3 SINTASSI

La sintassi di FACPL è definita nella tabella 1. La sintassi è fornita come una grammatica di tipo Extended Backus Naur form (EBNF), dove il simbolo ? corrisponde ad un elemento opzionale, il simbolo \* corrisponde ad una sequenza con un numero arbitrario di elementi (anche 0), ed il simbolo + corrisponde ad una sequenza non vuota con un numero arbitrario di elementi.

Al livello più alto c'è il Policy Authorisation System (PAS), il quale definisce le specifiche del PEP e del PDP. Il PEP è definito semplicemente come un *enforcing algorithm* che sarà applicato per decidere su quali decisioni verrà eseguito il processo di *enforcement*.

Tabella 2.: Sintassi ausiliaria per le risposte

<b>PDP Responses</b>	$PDPResponse ::= \langle Decision \ FObligation^* \rangle$
<b>Decisions</b>	$Decision ::= permit \mid deny \mid not\text{-}app \mid indet$
<b>Fulfilled obligations</b>	$FObligation ::= [ObType \ PepAction(Value^*)]$

Il PDP invece è definito come una sequenza (non vuota) di *Policy*, ed un algoritmo di combining che combinerà i risultati di queste *policy* per ottenere un unico risultato finale.

Una *policy* può essere una semplice *rule* o una *policy set*, quest'ultima avrà al suo interno altre *policy set* o *rule*, ed in questo modo viene formata una gerarchia di *policy*.

Un *policy set* individua un target, che è una espressione che indica il set di richieste di accesso alla quale si applica la *policy*, una lista di *obligations*, che definiscono azioni obbligatorie o opzionali che devono essere eseguite nel processo di *enforcement*, una sequenza di altre *policy*, ed un algoritmo per combinarle.

Una *rule* includerà un *effect*, che sarà permit o deny quando la regola è valutata correttamente, un target ed una lista di *obligations*.

Le *Expressions* sono formate da *attribute names* e valori (per esempio boolean, double, strings, date).

Un *Attribute Name* indica il valore di un attributo che può essere contenuto nella richiesta o nel contesto. FACPL usa per gli *Attribute Name* una forma del tipo *Identifier / Identifier*, dove il primo Identifier indica la categoria, ed il secondo il nome dell'attributo. Per esempio *Action / ID* rappresenta il valore di un attributo ID di categoria Action.

I *Combining Algorithm* implementano diverse strategie che servono per risolvere conflitti tra le varie decisioni, restituendo alla fine un'unica decisione finale.

Una *obligation* ha al suo interno un effect, un tipo, ed una azione eseguita dal PEP con la relativa *Expression*.

Una *request* consiste di una sequenza di attributi organizzati in categorie.

La risposta ad una valutazione di una richiesta FACPL è scritta usando la sintassi riportata in Tabella 2. La valutazione in due step, descritta precedentemente in Sezione 3.2, produce due tipi di risultati. Il primo è la risposta del PDP, il secondo è una decisione, ovvero una risposta del PEP.

La decisione del PDP, nel caso in cui ritorni *permit* o *deny*, viene associata ad una lista, anche vuota, di *fulfilled obligation*.

Una *fulfilled obligation* è una semplice coppia formata da un tipo (M o O) ed una azione i quali argomenti sono ottenuti dalla valutazione del PDP. Rappresenta una *obligation* valutata dal PEP

### 3.4 SEMANTICA

Molteplici sono le componenti di FACPL, e la semantica ora verrà informalmente analizzata. La semantica formale è presente in [9]. Prima è presentato il processo di decisione del PDP, successivamente quello PEP.

Quando il PDP riceve una richiesta, per prima cosa la valuta sulle basi delle *policy* disponibili, successivamente determinerà un risultato combinando le decisioni ritornate da queste *policy* attraverso degli algoritmi di combining.

La valutazione della *policy* rispetto alla richiesta comincia verificando l'applicabilità alla richiesta, che è fatta valutando un'espressione definita *target*.

Supponiamo che l'applicabilità dia esito positivo, nel caso ci sia una *rule* sarà ritornato il valore risultante dalla valutazione di quest'ultima, mentre se c'è un *policy set* il risultato è ottenuto valutando le *policy* contenute all'interno, e combinando i loro valori con uno specifico algoritmo. Successivamente a queste valutazioni viene effettuato il *fulfillment* delle *obligation* contenute all'interno delle *policy*.

Supponiamo ora che l'applicabilità non dia esito positivo, ovvero la valutazione del *target* restituisca *false*. In questo caso il risultato della *policy* sarà *not-app*. Mentre se il *target* restituisce un valore non booleano o ritorna un errore il risultato della *policy* sarà *indet*.

Valutare le espressioni corrisponde ad applicare degli operatori e risolvere i nomi degli attributi che contengono, e di conseguenza ricavarne un valore.

La valutazione di una *policy* termina con il *fulfillment* di tutte le *obligations* che hanno il valore di applicabilità coincidente con quello ritornato dalla valutazione della *policy*. Quest'operazione consiste nel valutare tutte le espressioni presenti al interno delle *obligations* coinvolte nel processo. Se ci sarà un errore nel processo di *fulfillment* allora il risultato della *policy* sarà *indet*, altrimenti il risultato del *fulfillment* sarà uguale a quello della valutazione del PDP.

Gli algoritmi di combining hanno lo scopo di combinare le decisioni risultanti dalla valutazione delle richieste in accordo con le policy. Un'altra funzione che hanno è, nel caso nel caso in cui la valutazione finale risulti permit o deny, ritornare le *obligations* coerenti con il risultato della decisione. Come ultimo step il risultato del PDP viene mandato al PEP per l'enforcement. Il PEP per effettuare questo processo deve eseguire l'azione all'interno di ogni *fulfilled obligation* e decidere come comportarsi per le decisioni di tipo not-app e indet.

Per fare questo processo il PEP usa delle strategie. In particolare, l'algoritmo deny-biased (rispettivamente, permit-based) effettua l'enforcement dei permit (rispettivamente deny) solo quando tutte le corrispondenti obligations sono correttamente eseguite, mentre effettua l'enforcement dei deny (rispettivamente permit) in tutti gli altri casi. Invece, l'algoritmo di base lascia tutte le decisioni non cambiate ma, in caso di decisioni permit e deny, effettua l'enforcement di indet se ci sarà un errore durante l'esecuzione delle obligations. Questo evidenzia che le obligations non solo influenzano il processo di autorizzazione, ma anche l'enforcement. Gli errori causati dalle obligations con tipo O vengono ignorati.

### 3.5 ESEMPI DI POLITICHE

In questa sezione è analizzata una semplice politica scritta in FACPL con delle possibili richieste di accesso, la sintassi delle richieste e delle politiche è leggermente diversa da quella riportata in Tabella 1 in quanto, per questioni di comodità e facilità di lettura del codice, è stata usata quella del plugin.

Codice 3.1: Esempio di politica in FACPL

---

```
PolicySet fileRule { permit-overrides
  target:
    equal("458", resource/resource-id)
  policies:
    Rule writeRule ( permit target:
      equal ("WRITE" , subject/action )
      && equal ("ADMINISTRATOR", subject/role)
    )
    Rule writePeronio ( permit target:
      equal("PERONIO", subject/id)
```

```

    )
    Rule denyRule ( deny target:
        equal("GUEST", subject/role) )
    obl:
    [ deny M action2 (subject / id )]
    [ permit M action1 (subject / id)]
}

```

---

Con il Codice 3.1 si vuole ottenere lo scopo di regolare l'accesso ad una risorsa chiamata 458. In questo caso gli utenti che hanno ruolo *GUEST* non possono accedere, mentre gli *ADMINISTRATOR* sì, fatta eccezione per l'utente Peronio, che qualunque ruolo abbia può accedere. Le richieste effettuate al sistema vengono mostrate in Codice 3.2, e sono tre. La prima proviene dall'utente Gianfabrizio che fa parte degli *ADMINISTRATOR*, la seconda e la terza rispettivamente dall'utente Gianpietro e Peronio che fanno entrambi parte dei *GUEST*.

Codice 3.2: Richieste per Codice 3.1

```

Request:{ Request1
    (subject/action , "WRITE")
    (subject/role , "ADMINISTRATOR")
    (resource/resource-id , "458")
    (subject/id, "GianFabrizio")
}
Request:{ Request2
    (subject/action , "WRITE")
    (subject/role , "GUEST")
    (resource/resource-id , "458")
    (subject/id, "GianPietro")
}
Request:{ Request3
    (subject/action , "WRITE")
    (subject/role , "GUEST")
    (resource/resource-id , "458")
    (subject/id, "PERONIO")
}

```

---



Tabella 3.: Risultati delle richieste

	<b>Risultato</b>	<b>Obligation</b>
<b>Richiesta 1</b>	<i>PERMIT</i>	PERMIT M action1([GIANFABRIZIO])
<b>Richiesta 2</b>	<i>DENY</i>	DENY M action2([GianPietro])
<b>Richiesta 3</b>	<i>PERMIT</i>	PERMIT M action1([PERONIO])

In Tabella 3 sono riassunti i risultati delle richieste. Ovviamente alla prima richiesta il risultato è permit, in quanto l'utente è un amministratore. Alla seconda richiesta il risultato è deny poiché l'utente è un ospite, mentre alla terza, nonostante l'utente faccia parte dello stesso gruppo del secondo riesce ad ottenere risultato permit per via della regola che considera il suo nome.

FACPL, come mostrato dall'esempio, permette di fare richieste ed ottenere delle risposte, ma queste richieste sono totalmente indipendenti l'una dall'altra, quindi l'ordine di esecuzione non avrebbe influenzato in alcun modo il risultato finale. In sezione 2.2 sono stati introdotti due esempi i quali non possono, per ora, essere implementati in FACPL poiché manca quest'aspetto che crea dipendenza tra le richieste. Per creare questa dipendenza tra richieste è necessario che il sistema di controllo agli accessi riesca a tenere traccia degli avvenimenti precedenti, perciò si inizia a parlare di un nuovo concetto che può essere assimilabile ad uno stato. Nel Capitolo 4 e 5 l'obiettivo è proprio permettere a FACPL questo tipo di valutazione.



---

## IMPLEMENTARE USAGE CONTROL IN FACPL

---

FACPL, per come è descritto nel Capitolo 3, non supporta Usage Control, di conseguenza non è possibile prendere decisioni basate sul comportamento passato. Grazie a delle nuove strutture, implementate insieme al mio collega Filippo Mamelì, è possibile prendere questo tipo di decisioni.

Questa estensione ha richiesto delle modifiche alla sintassi del linguaggio in modo da poter sfruttare facilmente le nuove funzionalità. Introdurre queste modifiche ha richiesto del lavoro sulla libreria, in quanto è stato necessario aggiungere nuove componenti e di conseguenza modificare il processo di valutazione delle policy.

In Sezione 4.1 viene analizzato il nuovo processo di valutazione alla luce delle modifiche introdotte in FACPL. Nella Sezione 4.2 invece viene discussa l'estensione dal punto di vista della sintassi, introducendo la nuova grammatica. Successivamente, in Sezione 4.3 viene spiegata la semantica dei nuovi costrutti implementati. Infine in Sezione 4.4 sono proposti in FACPL due case study già presentati in Sezione 2.2.1 e 2.2.2

### 4.1 ESTENSIONE DEL PROCESSO DI VALUTAZIONE

Il processo di valutazione, presentato in Sezione 3.2, è stato esteso per via delle modifiche introdotte. Rispetto al processo di valutazione standard, sono state aggiunte componenti al grafico, rendendolo così adatto allo *Usage Control*, e quindi assicurare un controllo continuativo basato sul comportamento passato.

Come si nota in Figura 9 è stato aggiunto un componente alla struttura della valutazione. Questo componente rappresenta lo *Status* (Stato), al quale il PDP e PEP ci accedono tramite attributi. Questi attributi vengono chiamati *Status Attribute*.

Analizziamo quindi, a scopo esemplificativo, come viene gestita la presenza dello stato. Inizialmente viene definito il sistema, che ora rispetto a quelli già citati in Sezione 3.2, ha un componente in più, ovvero lo Stato.

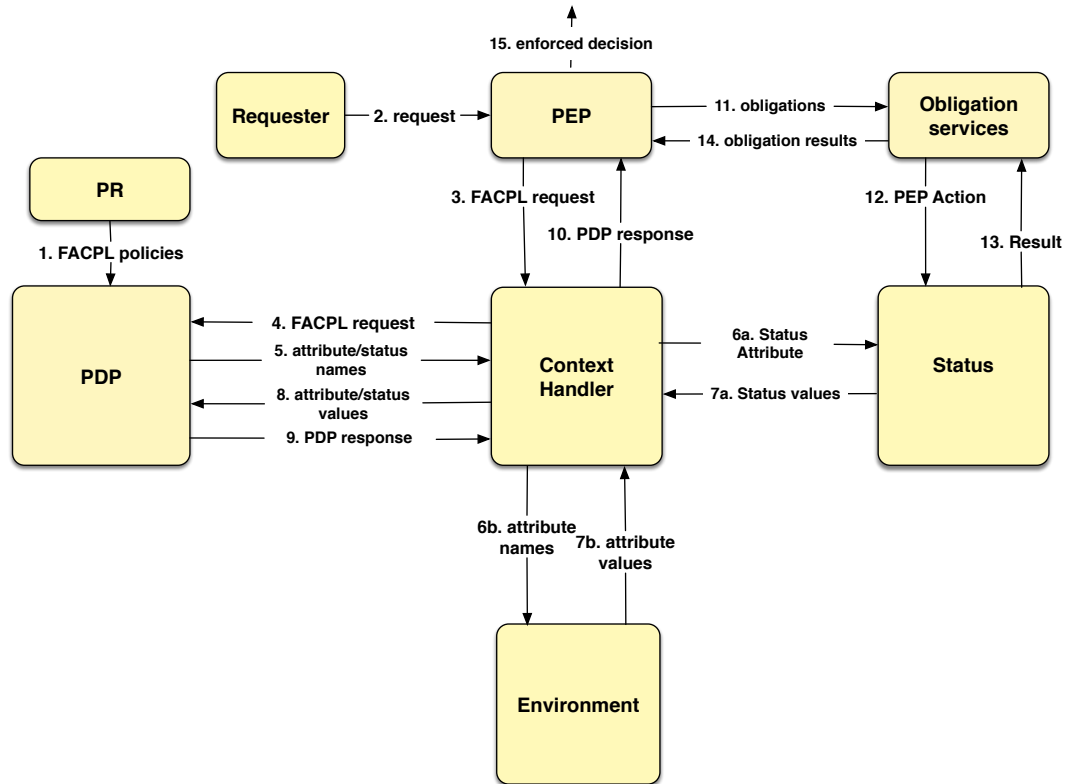


Figura 9.: Valutazione dopo lo stato

Fino al quarto step il comportamento è analogo a quello precedente, mentre cambia negli step successivi.

Al quinto step il *PDP* non necessiterà solo dei normali attributi d'ambiente, ma necessiterà anche degli *Status Attribute* coinvolti nella richiesta effettuata. Il *Context Handler* quindi non andrà solo a fare la ricerca all'interno dell'environment, ma andrà a cercare anche gli *Status Attribute* all'interno dello *Status*.

A questo punto, quando il *PDP* avrà tutte le informazioni necessarie si potrà passare alla vera e propria valutazione della richiesta che avviene come sempre.

Nel caso in cui viene restituito permit o deny è necessario fare l'enforcement della risposta del *PDP*. Questo processo differisce dal precedente poiché ora sono state implementate nuove azioni sullo stato che devono essere eseguite dal *PEP* (Passo 11-14). Le nuove azioni sono eseguite

attraverso un nuovo tipo di obligations, chiamate *ObligationStatus*. Quest'ultime vengono valutate dal PEP alla pari di una normale *Obligation*, ma la sostanziale differenza tra esse ed una normale *Obligation* è legata alla funzione che contengono. Mentre le normali *Obligation* conterranno generiche funzioni, come creare un log o mandare una mail, le *ObligationStatus* potranno eseguire azioni per modificare lo stato del sistema. Una volta effettuato l'enforcement viene restituita la decisione finale.

#### 4.2 ESTENSIONE LINGUISTICA

Per implementare queste nuove funzionalità è stata modificata anche la grammatica di FACPL. Nella grammatica estesa sono state aggiunte nuove regole di produzione e simboli terminali che codificano le nuove funzionalità.

Come è facilmente osservabile dalla consultazione della Tabella 4 le aggiunte rispetto alla tabella riporta in Sezione 3.3 sono state diverse, vediamo adesso quali sono.

La prima modifica è nel PAS, cioè lo *Status*, che è della forma

status : Attribute

ed è formato da uno o più *Attribute*.

Passiamo ora a descrivere *Attribute* che è della forma

(Type Identifier(= Value)<sup>?</sup>)

questo tipo particolare di attribute, che è lo *Status Attribute* descritto in precedenza, è formato innanzitutto da un *Type*, dopo il tipo è richiesta una generica stringa chiamata *Identifier*, che sarà un generico nome da dare all'attributo, infine viene richiesto un *Value*, ovvero un valore, che in questo caso è opzionale, all'atto pratico vuol dire che l'attributo di stato potrà essere inizializzato con un valore oppure potrà essere solamente definito, lasciando che il valore sia quello di default.

*Type* è il tipo che avrà l'attributo di stato, e potrà essere `int`, `boolean`, `date` o `double`.

La regola *PepAction* è stata modificata in modo tale che includesse nuove funzioni per operare sugli attributi di stato. Infine l'ultima regola

di produzione modificata è stata quella riguardante *Attribute Names*, in questo caso è stata semplicemente aggiunto, a fianco di *Identifier/Identifier*, una nuova produzione *Status/Identifier*. Questa nuova produzione serve semplicemente per permettere il confronto tra attributi di stato attraverso le già esistenti *Expression*. La sintassi delle risposte è rimasta invariata.

#### 4.3 SEMANTICA

La semantica di FACPL rimane molto simile a quella descritta in Sezione 3.4, quindi verranno di seguito descritte in modo informale solo le novità introdotte.

La prima di queste riguarda la valutazione delle richieste dal PDP. Il PDP ora non si deve più basare solo su richieste totalmente scollegate l'una dall'altra, e quindi è stato introdotto il concetto di *Status*. Lo stato permette di rappresentare il comportamento passato del sistema, e lo fa introducendo una nuova serie di attributi chiamati *Status Attribute*.

Nel linguaggio questo nuovo tipo di attributi viene considerato al pari di normali attributi, quindi si ha la possibilità di effettuare tutte le operazioni di confronto tra di essi, ma in più si deve avere la possibilità di modificarli e memorizzarli in modo da poterli sfruttare per *Usage Control*.

Per questo sono state aggiunte delle *Pep Action*, ovvero delle azioni eseguite dal PEP in seguito alla valutazione di *Obligations status*. La prima di queste è l'addizione, e permette, in seguito alla valutazione di una *Obligations*, l'aggiunta di un valore numerico, definito dallo sviluppatore, ad uno *Status Attribute* di tipo *double* o *int*.

```
obl:
    [permit M add(counter, 2)]
```

Per esempio l'esecuzione di questa *Obligation* su un'attributo, chiamato *counter*, inizializzato a 0, porterà l'attributo al valore 2.

L'operazione di somma è stata implementata anche per altri due tipi, *Date* e *String*. Oltre all'addizione sono presenti funzioni per la sottrazione, divisione e moltiplicazione che operano in modo analogo a questa appena descritta, ma sono definite soltanto su tipi numerici.

Un'altra operazione implementata modifica il valore originale di uno *Status Attribute* con il valore passatogli come secondo parametro.

```
obl:
    [permit M flag(isFoo, true)]
```

L'esecuzione con successo di questa *Obligation* porterà l'attributo *flag* ad avere un valore *true*. Questo tipo di operazione è stata definita anche per il tipo *Date* e *String*.

Vediamo ora un esempio di questa nuova estensione, prenderemo spunto dal primo caso trattato in precedenza nella Sezione 4.1.

Codice 4.1: Esempio per la sintassi

---

```
Policy example < permit-overrides
  target: equal("Bob",name/id) && equal("read", action/id)
  rules:
    Rule access (
      permit target: less-than(status/counter, 2))
    obl:
      [ permit M add(counter, 1)]
>

PAS {
  Combined Decision : false ;
  Extended Indeterminate : false ;
  Java Package : "example" ;
  Requests To Evaluate : Request_example ;
  pep: deny- biased
  pdp: deny- unless- permit
  status: [(int counter = 0)]
  include example
}
```

---

In questo esempio (Codice 4.1) si può vedere come nel PAS è stato definito uno stato, con al suo interno uno solo attributo inizializzato con valore 0. Successivamente si può notare nella *Rule* che viene fatto un controllo sul valore di quest'attributo. Infine nella *Obligation* si può notare come viene aggiornato lo stato dell'attributo in base al risultato della valutazione della *Rule*.

#### 4.4 FORMALIZZAZIONE DEI CASE STUDY

Queste nuove funzionalità introdotte servono allo scopo descritto in Sezione 2.2, ovvero l'implementazione di un nuovo modello chiamato *Usage Control*. In questi due semplici case study lo stato gioca un ruolo

fondamentale, in quanto il sistema di access control, per poter soddisfare requisiti di consistenza deve tener traccia del comportamento passato. Mostriamo ora l'implementazione in FACPL dei due esempi trattati in Sezione 2.2. Per comodità è usata la sintassi del plugin di Eclipse, che differisce leggermente da quella presentata in 4.2.

#### 4.4.1 Accesso ai file

Il primo esempio in Sezione 2.2 poneva una regola sull'accesso ai file, ovvero permetteva un massimo di due persone in contemporanea che potevano effettuare l'accesso in lettura oppure un massimo di una persona che poteva ottenere l'accesso in scrittura.

Tutte le policy che saranno mostrate in questa sezione sono incluse in un *PolicySet* che racchiude al suo interno un'espressione di tipo target, mostrata in Codice 4.2.

Codice 4.2: Target PolicySet

---

```
PolicySet ReadWrite_Policy { deny- unless- permit
  target: equal ( "Bob" , name / id ) && ("Alice, name/id") }
```

---

Il target del PolicySet ReadWrite\_Policy verifica che le richieste provengano da utenti che hanno nome *Alice* o *Bob*, in caso contrario il responso sarà Not Applicable.

Successivamente sono state scritte quattro policy per gestire le quattro operazioni possibili, ovvero read, write, stopRead ed infine stopWrite.

Prendiamo in considerazione la policy mostrata in Codice 4.3, ovvero quella per la write. Come prima è presente un target, che richiede questa volta due diverse condizioni, la prima riguarda l'id del file richiesto, la seconda invece richiede che l'azione sia write. Le parti interessanti di questa policy sono due, la prima riguarda la *Rule*, la seconda la *Obligation*.

La *Rule* restituisce permit se le due condizioni dell'equal sono vere, come si può facilmente notare l'operazione di confronto non viene fatta tra una stringa ed un normale attributo, ma tra una stringa ed uno *Status Attribute*.

L'ultima cosa da notare è l'unica *Obligation* presente per questa policy. Questo tipo particolare di *Obligation*, ha sempre al suo interno un'azione che verrà eseguita dal PEP, questa volta però non sarà una semplice azione come scrivere un log o mandare una mail, l'azione andrà a modificare lo stato del sistema, mettendo il valore true all'attributo *isWriting*.



Codice 4.3: Policy Write

---

```

PolicySet Write_Policy { deny- unless- permit
  target: equal("file", file/id) && ("write", action/id)
  policies:
    Rule write ( permit target:
      equal ( status / isWriting , false ) &&
      equal ( status / counterReadFile1, 0)
    )
  obl:
    [ permit M flagStatus(isWriting, true) ]
}

```

---

Successivamente si prende in considerazione la policy per l'operazione stopWrite, mostrata in Codice 4.4.

Codice 4.4: Policy StopWrite

---

```

PolicySet StopWrite_Policy { deny- unless- permit
  target: equal("file", file/id) && ("stopWrite", action/id)
  policies:
    Rule stopWrite ( permit target:
      equal ( status / isWriting , true )
    )
  obl:
    [ permit M flagStatus(isWriting, false) ]
}

```

---

Il target definito da questa policy è molto simile a quello precedente, la differenza è nella seconda parte: in questo caso l'azione richiesta non è Write, ma StopWrite.

Come prima c'è una *Rule* all'interno che esegue un confronto tra uno *Status Attribute* ed un valore, in questo caso true. Questo confronto in questo caso serve per verificare la reale presenza di uno scrittore al momento della richiesta. Nel caso fosse presente, e quindi la regola restituisse true, viene eseguita la *Obligation Status* che si occupa della modifica dello stato. Le altre policy, per definire le restanti due operazioni, sono analoghe a quelle appena descritte e si possono trovare in Codice A.1.

### Valutazione

Prendiamo ora una serie di richieste ed analizziamone la loro valutazione.

Codice 4.5: Richieste del primo esempio

---

```

Request:{ Request1
  (name / id , "Alice")
  (action / id, "read")
  (file / id, "file1")
}
Request:{ Request2
  (name / id , "Bob")
  (action / id, "Write")
  (file / id, "file1")
}
Request:{ Request3
  (name / id , "Bob")
  (action / id, "read")
  (file / id, "file1")
}
Request:{ Request4
  (name / id , "Alice")
  (action / id, "stopRead")
  (file / id, "file1")
}
Request:{ Request4
  (name / id , "Bob")
  (action / id, "stopRead")
  (file / id, "file1")
}
Request:{ Request6
  (name / id , "Alice")
  (action / id, "write")
  (file / id, "file1")
}

```

---

La prima richiesta proviene da Alice, e sarà una lettura sul file<sub>1</sub>, la successiva proviene da Bob, è sempre sul file<sub>1</sub>, ma l'azione richiesta è di scrittura. Le altre sono analoghe.

L'output di queste richieste è mostrato in Tabella 5. Analizziamo ora il motivo di queste decisioni. Nella prima richiesta ovviamente nessuno sta leggendo o scrivendo, quindi viene tranquillamente restituito permit. Visto che è presente una *obligation* lo stato verrà aggiornato, sommando un'unità al contatore di letture. Alla seconda richiesta l'utente richiede la scrittura, che gli viene negata perché c'è già qualcuno che sta leggendo, però lo stesso utente effettua un'altra richiesta, questa volta in lettura,

che gli viene concessa.

La quarta e la quinta richiesta vengono fatte per avvisare il sistema che la lettura è terminata, ovviamente la risposta è *permit*, e la *obligation* corrispondente decrementerà il contatore. La sesta ed ultima richiesta è una scrittura, che questa volta viene permessa, poiché nessuno sta scrivendo o leggendo.

Tabella 5.: Risultati della valutazione

	Risultato	Stato Prima	Stato dopo
<b>Richiesta 1</b>	<i>PERMIT</i>	isWriting = false CounterReadFile1 = 0	isWriting = false CounterReadFile1 = 1
<b>Richiesta 2</b>	<i>DENY</i>	isWriting = false CounterReadFile1 = 1	isWriting = false CounterReadFile1 = 1
<b>Richiesta 3</b>	<i>PERMIT</i>	isWriting = false CounterReadFile1 = 1	isWriting = false CounterReadFile1 = 2
<b>Richiesta 4</b>	<i>PERMIT</i>	isWriting = false CounterReadFile1 = 2	isWriting = false CounterReadFile1 = 1
<b>Richiesta 5</b>	<i>PERMIT</i>	isWriting = false CounterReadFile1 = 1	isWriting = false CounterReadFile1 = 0
<b>Richiesta 6</b>	<i>PERMIT</i>	isWriting = false CounterReadFile1 = 0	isWriting = true CounterReadFile1 = 0

#### 4.4.2 Noleggio e acquisto di contenuti

In questo secondo esempio analizzeremo il caso di un'azienda di distribuzione di contenuti multimediali che vuole regolare l'accesso di quest'ultimi attraverso policy. Faremo un breve esempio con un solo file e due utenti, uno dei due utenti comprerà il file, l'altro lo noleggerà a tempo determinato. Nel codice 4.6 vengono mostrate solo una parte delle policy presenti nel Codice completo A.2 mostrato in Appendice A.

Codice 4.6: Secondo Esempio

```

PolicySet Negozio { deny- unless- permit
  target: equal ( "Bob" , name / id ) || ( "Alice, name/id" )
  policies:

  PolicySet Buy_Policy { deny- unless- permit
    target: equal("file1", file/id) && ("buy", action/id)
  }
}

```

```

policies:
  Rule alice_buy ( permit target: (
    equal ( action / id , "buy" ) &&
    equal ( name / id, "Alice"))
    obl:
    [ permit M setString("accessTypeAlice", "BUY") ]
  )
  Rule bob_buy ( permit target: (
    equal ( action / id , "buy" ) &&
    equal ( name / id, "Alice"))
    obl:
    [ permit M setString("accessTypeBob", "BUY") ]
  )
}

PolicySet NUMBER_Policy { deny- unless- permit
  target: equal("file1", file/id) && ("number", action/id)
  policies:
    Rule alice_buy ( permit target: (
      equal ( action / id , "number" ) &&
      equal ( name / id, "Alice"))
      obl:
      [ permit M setString("accessTypeAlice", "NUMBER") ]
      [ permit M addStatus("aliceFileIviewNumber", 2) ]
    )
    Rule bob_buy ( permit target: (
      equal ( action / id , "number" ) &&
      equal ( name / id, "Alice"))
      obl:
      [ permit M setString("accessTypeBob", "NUMBER") ]
      [ permit M addStatus("bobFileIviewNumber", 2) ]
    )
  }
}

```

Queste due policy, e anche le altre che non sono state mostrate, sono racchiuse tutte all'interno del *Policy Set* Negoziio il quale come prima cosa verifica se chi ha fatto la richiesta ha un determinato nome, in questo caso *Bob* o *Alice*.

Successivamente, se uno dei due effettua la richiesta di BUY, ovvero l'acquisto senza alcun tipo di limitazione, si entra nella prima policy e, tramite le *Obligation* si cambia l'attributo di stato. Invece se un utente

decidesse di effettuare il noleggio con la modalità dove si limita il numero di visioni si entrerebbe nella seconda *Policy Set* la quale, attraverso *Obligations* aumenterà il numero di visioni di due unità. Analogò è il caso del noleggio a tempo.

Per disciplinare la visione è presente un altro *Policy Set*, mostrato anch'esso parzialmente in codice 4.7.

Codice 4.7: Secondo Esempio

---

```

PolicySet VIEW { deny- unless- permit
  target: equal("file1", file/id) && ("view", action/id)
  policies:
    Rule buy ( permit target: (
      equal ( status / accessTypeBob , "BUY" ) &&
      equal ( status / accessTypeAlice, "BUY"))
    )
    Rule number_alice ( permit target: (
      equal ( status / accessTypeAlice, "NUMBER" ) &&
      equal ( name / id, "Alice") && greater-than( status /
        aliceFileIviewNumber, 0))
    obl:
      [ permit M subStatus("aliceFileIviewNumber", 1) ]
    )
  }

```

---

### Valutazione

Mostriamo ora in Codice 4.8 alcune richieste che possono essere fatte al sistema ed analizziamo le risposte che produrranno, in Tabella 6 è mostrato un quadro riassuntivo del risultato delle richieste.

Codice 4.8: Richieste del Secondo Esempio

---

```

Request:{ Request1
  (name / id , "Alice")
  (action / id, "view")
  (file / id, "file1")
}

Request:{ Request2
  (name / id , "Bob")
  (action / id, "view")
  (file / id, "file1")
}

```

```

}

Request:{ Request3
  (name / id , "Alice")
  (action / id, "Buy")
  (file / id, "file1")
}

Request:{ Request4
  (name / id , "Alice")
  (action / id, "view")
  (file / id, "file1")
}

Request:{ Request4
  (name / id , "Bob")
  (action / id, "Time")
  (file / id, "file1")
}

Request:{ Request6
  (name / id , "Bob")
  (action / id, "view")
  (file / id, "file1")
}

```

---

La prima e la seconda richiesta sono richieste di visione, che ovviamente restituiranno entrambe deny, in quanto lo stato del sistema non è stato modificato da nessuno poiché nè Alice nè Bob hanno effettuato acquisti o noleggi. Successivamente Alice effettuerà un acquisto, e quindi tramite la Obligation Status verrà modificato lo stato del sistema, accreditando così l'acquisto. Dopo aver effettuato questa richiesta Alice ne effettua un'altra, questa volta di visione. A questo punto la policy che disciplina quest'ultima richiesta di Alice effettua una verifica dello *Status Attribute AccessTypeAlice*, e visto che lo trova cambiato dalla precedente richiesta di acquisto permette la visione. Bob, a cui all'inizio era stata negata la visione effettuerà una richiesta di noleggio e quindi cambierà lo stato. Dopo, sempre Bob, richiede la visione, il risultato di entrambe sarà ovviamente permit.

Tabella 6.: Riassunto Valutazione

	Risultato	Stato Prima	Stato dopo
<b>Richiesta 1</b>	<i>DENY</i>	AccessTypeAlice = null AccessTypeBob = null	AccessTypeAlice = null AccessTypeBob = null
<b>Richiesta 2</b>	<i>DENY</i>	AccessTypeAlice = null AccessTypeBob = null	AccessTypeAlice = null AccessTypeBob = null
<b>Richiesta 3</b>	<i>PERMIT</i>	AccessTypeAlice = null AccessTypeBob = null	AccessTypeAlice = BUY AccessTypeBob = null
<b>Richiesta 4</b>	<i>PERMIT</i>	AccessTypeAlice = BUY AccessTypeBob = null	AccessTypeAlice = BUY AccessTypeBob = null
<b>Richiesta 5</b>	<i>PERMIT</i>	AccessTypeAlice = BUY AccessTypeBob = null	AccessTypeAlice = BUY AccessTypeBob = TIME BobFile1Expiration = 2016/04/22
<b>Richiesta 6</b>	<i>PERMIT</i>	AccessTypeAlice = BUY AccessTypeBob = TIME BobFile1Expiration = 2016/04/22	AccessTypeAlice = BUY AccessTypeBob = TIME BobFile1Expiration = 2016/04/22

Tabella 4.: Sintassi di FACPL<sub>PB</sub>


---

<b>Policy Authorisation Systems</b>	$PAS ::= (\text{pep} : \text{EnfAlg} \text{ pdp} : \text{PDP} \text{ status} : [\text{Attribute}])$
<b>Attribute</b>	$\text{Attribute} ::= (\text{Type Identifier} (= \text{Value})?)$
<b>Type</b>	$\text{Type} ::= \text{int} \mid \text{boolean} \mid \text{date} \mid \text{double}$
<b>Enforcement algorithms</b>	$\text{EnfAlg} ::= \text{base} \mid \text{deny-biased} \mid \text{permit-biased}$
<b>Policy Decision Points</b>	$\text{PDP} ::= \{\text{Alg} \text{ policies} : \text{Policy}^+\}$
<b>Combining algorithms</b>	$\text{Alg} ::= \text{p-over} \mid \text{d-over} \mid \text{d-unless-p} \mid \text{p-unless-d}$ $\mid \text{first-app} \mid \text{one-app} \mid \text{weak-con} \mid \text{strong-con}$
<b>Policies</b>	$\text{Policy} ::= (\text{Effect} \text{ target} : \text{Expr} \text{ obl} : \text{Obligation}^*)$ $\mid \{\text{Alg} \text{ target} : \text{Expr} \text{ policies} : \text{Policy}^+ \text{ obl} : \text{Obligation}^*\}$
<b>Effects</b>	$\text{Effect} ::= \text{permit} \mid \text{deny}$
<b>Obligations</b>	$\text{Obligation} ::= [\text{Effect} \text{ ObType} \text{ PepAction}(\text{Expr}^*)]$
<b>PepAction</b>	$\text{PepAction} ::= \text{add}(\text{Attribute}, \text{int}) \mid \text{flag}(\text{Attribute}, \text{boolean})$ $\mid \text{sumDate}(\text{Attribute}, \text{date}) \mid \text{div}(\text{Attribute}, \text{int})$ $\mid \text{add}(\text{Attribute}, \text{float}) \mid \text{mul}(\text{Attribute}, \text{float})$ $\mid \text{mul}(\text{Attribute}, \text{int}) \mid \text{div}(\text{Attribute}, \text{float})$ $\mid \text{sub}(\text{Attribute}, \text{int}) \mid \text{sub}(\text{Attribute}, \text{float})$ $\mid \text{sumString}(\text{Attribute}, \text{string})$ $\mid \text{setValue}(\text{Attribute}, \text{string})$ $\mid \text{setDate}(\text{Attribute}, \text{date})$
<b>Obligation Types</b>	$\text{ObType} ::= \text{M} \mid \text{O}$
<b>Expressions</b>	$\text{Expr} ::= \text{Name} \mid \text{Value}$ $\mid \text{and}(\text{Expr}, \text{Expr}) \mid \text{or}(\text{Expr}, \text{Expr}) \mid \text{not}(\text{Expr})$ $\mid \text{equal}(\text{Expr}, \text{Expr}) \mid \text{in}(\text{Expr}, \text{Expr})$ $\mid \text{greater-than}(\text{Expr}, \text{Expr}) \mid \text{add}(\text{Expr}, \text{Expr})$ $\mid \text{subtract}(\text{Expr}, \text{Expr}) \mid \text{divide}(\text{Expr}, \text{Expr})$ $\mid \text{multiply}(\text{Expr}, \text{Expr}) \mid \text{less-than}(\text{Expr}, \text{Expr})$
<b>Attribute Names</b>	$\text{Name} ::= \text{Identifier/Identifier} \mid \text{status/Identifier}$
<b>Literal Values</b>	$\text{Value} ::= \text{true} \mid \text{false} \mid \text{Double} \mid \text{String} \mid \text{Date}$
<b>Requests</b>	$\text{Request} ::= (\text{Name}, \text{Value})^+$

---



---

## ESTENSIONE DELLA LIBRERIA FACPL

---

Il linguaggio FACPL è supportato da una libreria Java. Per supportare l'estensione proposta abbiamo esteso questa libreria. Per ovvi motivi verranno mostrate solo alcune parti delle modifiche effettuate, ma il codice completo si può comunque trovare su GitHub all'url <https://github.com/andreamargheri/FACPL>.

In Sezione 5.1 è presentata l'implementazione in Java dello stato e degli attributi di stato. Lo stato ha richiesto anche l'implementazione di altre componenti come le funzioni per modificare gli attributi e l'estensione del PEP. Nella Sezione 5.2 è trattata l'estensione delle politiche. In particolare si parla di come è stato possibile effettuare comparazioni su attributi di stato e di come sono state estese le obligation. In Sezione 5.3 viene mostrato come è stato esteso il plugin di FACPL per supportare le nuove estensioni. Infine, in Sezione 5.4, vengono presentati in Java i case study già proposti in Capitolo 4 e 2

### 5.1 IMPLEMENTAZIONE STATO

Il primo passo per estendere la libreria è stato la creazione di uno *Status*, che è modellato da una semplice classe di cui ne verrà mostrato un pezzo in Codice 5.1. Il fulcro di questa classe è una Hashmap con key parametrizzata a *Status Attribute* e valore corrispondente parametrizzato ad Object. Questa Hashmap associa quindi ad uno *Status Attribute* il suo valore corrispondente.

---

Codice 5.1: Stralcio della classe Status

---

```
public class FacplStatus {  
2   private String statusID;  
   private HashMap<StatusAttribute, Object> status;  
4   public FacplStatus(String statusID) {  
       this.status = new HashMap<StatusAttribute, Object>();  
   }
```

```

6      this.statusID = statusID;
    }

```

In Figura 10 è possibile vedere la relazione che intercorre tra lo stato ed i suoi attributi.

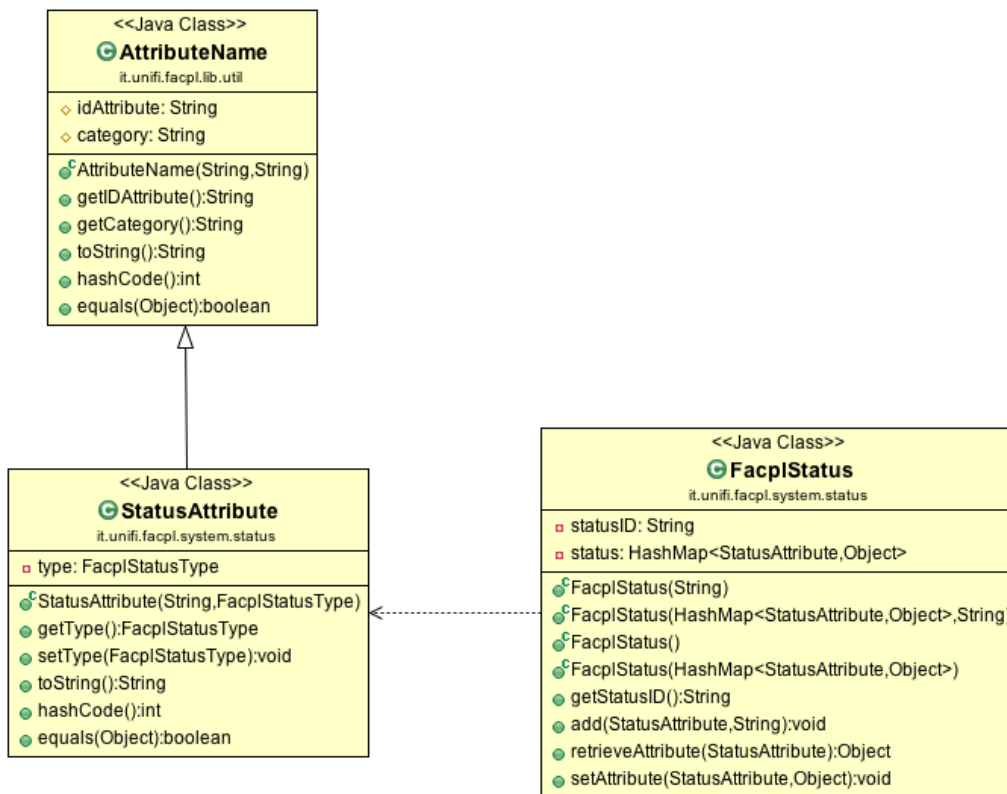


Figura 10.: Grafico UML delle classi Status e StatusAttribute

Come si vede dal grafico UML in Figura 10 la classe che modella lo stato include altri due metodi non mostrati in Codice 5.1. Per via dei rispettivi nomi i metodi risultano abbastanza autoesplicativi, per questo ne verrà mostrato solo uno in Codice 5.2.

Codice 5.2: Set Attribute della classe Status

```

32 public void setAttribute(StatusAttribute attribute, Object o)
    throws MissingAttributeException {
    Object v = this.status.get(attribute);
34     if (v == null){
        throw new MissingAttributeException("attribute doesn't exist in the
            current status");
36     } else{ this.status.put(attribute, o); }

```

```
}
```

---

### 5.1.1 *Status Attribute*

Uno Status Attribute è un tipo particolare di attributo. Come si può vedere dalla Figura 10 per creare questo nuovo tipo è stata estesa una classe già esistente, ovvero quella che rappresenta gli attributi normali.

Codice 5.3: Classe Status Attribute

---

```
public class StatusAttribute extends AttributeName {  
2   private FacplStatusType type;  
   public StatusAttribute(String id, FacplStatusType type) {  
4       super(id,"status");  
       this.type = type;  
6   }  
}
```

---

Nel Codice 5.3 è possibile vedere il costruttore e i campi di questa nuova classe. In aggiunta agli attributi normali è stato aggiunto un Tipo, che può essere:

- Int
- String
- Date
- Boolean
- Double

Il costruttore riceve due parametri: l'ID e il tipo. In questo metodo viene invocato il costruttore della superclasse dove gli viene passata una categoria fissata, ovvero *Status* e l'ID.

### 5.1.2 *Funzioni su Status Attribute*

Le funzioni sugli status attribute sono operazioni che ne vanno a modificare il valore. Derivano tutte da un'unica interfaccia mostrata in Codice 5.4, che avrà come unico metodo quello che eseguirà la vera e propria operazione.

Codice 5.4: Interfaccia per le operazioni

---

```

1 public interface IExpressionFunctionStatus {
2     public void evaluateFunction(List<Object> args) throws Throwable;
3 }

```

---

Dal grafico UML in Figura 11 si vede chiaramente che ci sono due grandi gerarchie di classi, quelle che fanno operazioni sui numeri, e quelle che fanno operazioni sulle stringhe. Per eseguire un'operazione è sufficiente istanziare una classe che estende *StringOperationStatus* o *MathOperationStatus*, e chiamare l'unico metodo definito nell'interfaccia in Codice 5.4 passandogli i parametri corretti. La sottoclasse istanziata eredita il metodo *evaluateFunction* dalla superclasse ed implementa il metodo astratto *op*. Il metodo *evaluateFunction* implementato nella superclasse, in base ai parametri che gli vengono passati si fa restituire da una factory, in base al tipo del attributo, il valutatore corretto che passerà a sua volta al metodo astratto *op*. Il valutatore è una classe dove è scritto il codice Java che eseguirà l'operazione vera e propria. Nel Codice 5.5 viene mostrata la sottoclasse che si occupa della somma su numeri. Si nota subito che questa è una sottoclasse di *MathOperationStatus* e che implementa solo il metodo *Op*.

Codice 5.5: Add Status

---

```

1 public class AddStatus extends MathOperationStatus {
2     @Override
3     protected void op(ArithmeticEvaluatorStatus ev, StatusAttribute
4         s1, Object o2) throws Throwable {
5         ev.add(s1, o2);
6     }
7 }

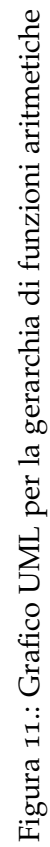
```

---

La ragione dietro la verbosità creata dall'utilizzo di un valutatore, oltre che da una moltitudine di classi e sottoclassi, potrebbe sembrare superflua; ciononostante quest'ultima è una scelta lungimirante, in quanto in futuro sarà più agevole implementare nuovi tipi di attributo e relative funzioni su di essi.

### 5.1.3 Estensione del PEP

La struttura del PEP è rimasta pressoché uguale in quanto non è stato esteso, bensì modificato. In seguito alla valutazione di una policy il PDP



darà una risposta, su questa risposta il PEP dovrà effettuare il suo processo di enforcement. Durante questo processo vengono valutate le *Obligation* in modo che l'operazione al loro interno sia eseguita; il metodo che effettua questo compito è *dischargeObligation*. La modifica ha coinvolto proprio quest'ultimo, in quanto è bastato aggiungere un *else if*, mostrato in Codice 5.12, a quelli già presenti.

Codice 5.6: Discharge delle Fulfilled Obligation di stato

---

```
102     else if (obl instanceof FulfilledObligationStatus) {  
        obl = (FulfilledObligationStatus) obl;  
104     obl.evaluateObl();  
    }
```

---

In questo modo vengono prese in considerazione anche le nuove Obligation Status, quindi ora il PEP, durante il suo processo di enforcement, riuscendo ad eseguire questo nuovo tipo di Obligation potrà modificare lo stato.

## 5.2 ESTENSIONE DELLE POLITICHE

Le politiche, in seguito all'estensione, vengono valutate basandosi anche sullo stato indi per cui è stato necessario estendere il contesto intorno a cui sono valutate. Come mostrato in Figura 12 l'estensione del contesto si ottiene mediante la creazione di una nuova classe, a estensione di quella esistente.

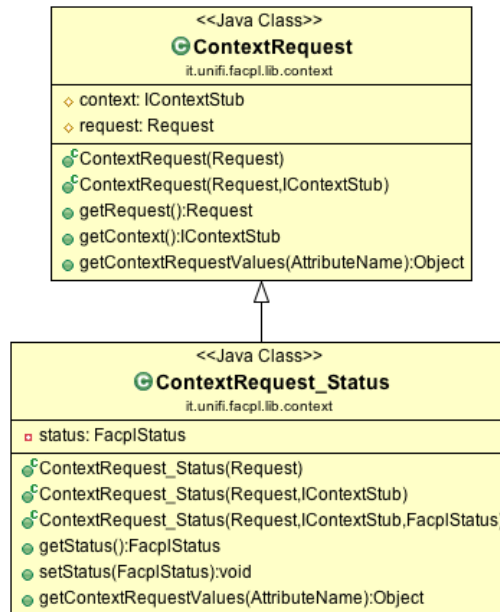


Figura 12.: Contesto

La nuova classe `ContextRequest_Status` contiene lo stato sottoforma di campo privato. L'accesso allo stato è effettuato tramite il metodo `getContextRequestValue` (Codice 5.7), che in questa sottoclasse è stato riscritto.

Codice 5.7: Discharge delle Fulfilled Obligation di stato

```

@Override
38 public Object getContextRequestValues(AttributeName name) throws
    MissingAttributeException {
    Logger l = LoggerFactory.getLogger(ContextRequest_Status.class);
40 if (name instanceof StatusAttribute) {
    if (status != null) {
42     try {
        return this.status.retrieveAttribute((StatusAttribute)
            name);
44     } catch (MissingAttributeException e) {
        l.debug("Throw MissingAttributeException for " +
            name.toString() + "Status is missing");
46     throw new MissingAttributeException();
    }
48 } else {
    l.debug("Throw MissingAttributeException for " +
        name.toString() + "Status is missing");
  
```

```

50         throw new MissingAttributeException();
    }
52     }else {
        return super.getContextRequestValues(name);
54     }
    }

```

Questo metodo riceve come parametro un attributo, ed in base al suo tipo andrà a cercarlo nello stato o nell'ambiente.

### 5.2.1 Funzioni di controllo su status

Il PDP per poter valutare correttamente le richieste deve avere la possibilità di comparare anche questo nuovo tipo di attributi. Raggiungere quest'obiettivo non è stato difficile in quanto la struttura era già esistente e funzionante, ed essendo i nuovi attributi soltanto un'estensione di quelli creati in precedenza è stato possibile usarla senza alcun tipo di problema.

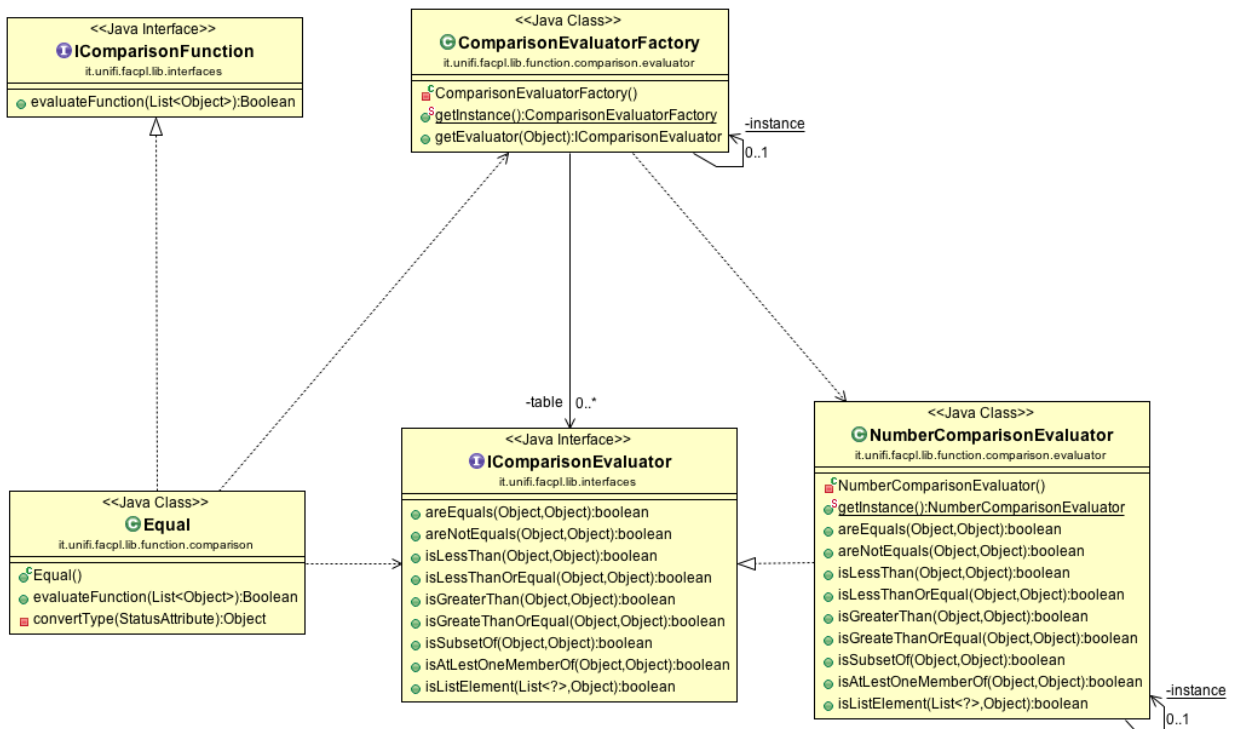


Figura 13.: Comparatori

Come nel caso delle funzioni di modifica dello stato, la struttura è basata su un factory che ritorna il valutatore corretto in base al tipo su



cui deve essere effettuato il confronto.

### 5.2.2 *Obligation*

Le *Obligation* sono state estese introducendo un nuovo tipo chiamato *Obligation Status*, questo tipo particolare di *Obligation* servono per andare ad eseguire azioni sullo stato. Nella libreria sono presenti due tipi fondamentali di *Obligation*, il primo sono quelle a livello sintattico, le seconde, chiamate *FulfilledObligations* sono quelle pronte ad essere valutate. Vediamo adesso come sono state estese quelle a livello sintattico.

Per eseguire questa estensione è stato reso necessario un refactoring, per prima cosa è stato astratto tutto il comportamento comune in una superclasse astratta, successivamente è stata creata la nuova classe che modella questo nuovo tipo. Il refactoring ha coinvolto anche il metodo che si occupa del *Fulfilling* delle *Obligation* in quanto ora deve creare anche questo nuovo tipo, la scelta più ovvia è stata creare un metodo astratto implementato nelle due sottoclassi che viene chiamato dalla superclasse per creare il tipo corretto.

---

Codice 5.8: Parte rifattorizzata del metodo che si occupa del fulfilling

---

```

16    l.debug("Fulfilling Obligation " + this.pepAction.toString() + "...");
    AbstractFulfilledObligation obl = this.createObligation();
18    if (obl instanceof FulfilledObligationCheck) {

```

---



---

Codice 5.9: CreateObligation nelle status

---

```

protected AbstractFulfilledObligation createObligation() {
18    AbstractFulfilledObligation obl = new
        FulfilledObligationStatus(this.evaluatedOn, this.typeObl,
            (IExpressionFunctionStatus) this.pepAction);
20    if (!argsStatus.isEmpty()) {
        obl.addArgStatus(argsStatus);
22    }
    return obl;
24 }

```

---



---

Codice 5.10: CreateObligation nelle normali

---

```

@Override
12    protected AbstractFulfilledObligation createObligation() {

```

```

    return new FullfilledObligation(this.evaluatedOn,
    this.typeObl, (String) this.pepAction);
14 }

```

---

La *Obligation* di stato necessiterà anche di argomenti su cui eseguire l'azione, che le verranno passati in fase di costruzione.

Alla fine della valutazione il PDP crea un oggetto di tipo *AuthorisationPDP* che conterrà la decisione e una lista di *FulfilledObligation*, quest'ultime poi andranno al PEP per la loro valutazione. Vediamo ora come sono state implementate.

Anche in questo caso è stato necessario un refactoring analogo a quello fatto per le prime.

Codice 5.11: Peculiarità della classe *FulfilledObligationStatus*

---

```

public class FulfilledObligationStatus extends
    AbstractFulfilledObligation {
2   private IExpressionFunctionStatus pepFunction;
    public FulfilledObligationStatus(Effect effect, ObligationType
        typeObl, IExpressionFunctionStatus pepFunction) {
4       super(effect, typeObl);
        this.pepFunction = pepFunction;
6   }
    public FulfilledObligationStatus(Effect effect, ObligationType
        typeObl,
8        Class<? extends IExpressionFunctionStatus> pepFunction) {
        super(effect, typeObl);
10   }
    @Override
12   public AbstractFulfilledObligation evaluateObl() throws Throwable
        {
            this.pepFunction.evaluateFunction(this.getArgsStatus());
14   return this;
        }
}

```

---

Come si può notare, in fase di costruzione, gli verrà passato un oggetto di tipo *IExpressionFunctionStatus* che sarà l'azione che andrà a eseguire sullo stato. Quest'azione andrà realmente ad essere eseguita quando verrà chiamato dal PEP il metodo *evaluateObl*.

Il PEP nella fase di enforcement effettua la valutazione delle *Obligation*,

in questo caso le modifiche per permettere al sistema di eseguirle sono state minime, è bastato modificare il metodo *DischargeObligation* in modo che quando gli viene passata una *AbstractFulfilledObligation* chiamasse il metodo *evaluateObl*.

Codice 5.12: Discharge delle Fulfilled Obligation di stato

---

```

102     else if (obl instanceof FulfilledObligationStatus) {
        obl = (FulfilledObligationStatus) obl;
104     obl.evaluateObl();
    }

```

---

### 5.3 PLUGIN ECLIPSE

[IN SOSPESO]

### 5.4 ESEMPI

In questa sezione vengono ripresi i case study già discussi in Sezione 4.4 mostrando com'è il loro corrispettivo in Java. Per facilitare la lettura verrà proposto una parte del codice in Java degli esempi, il codice completo si può trovare in Capitolo A.

#### 5.4.1 Primo case study

Il punto di partenza del sistema è la classe dov'è contenuto il metodo Main; quest'ultima rappresenta il punto dove il PDP e PEP vengono inizializzati e preparati all'esecuzione. In questa classe vengono inizializzati gli oggetti che modellano le policy, le richieste.

Le richieste sono modellate da una serie di classi che accolgono al loro interno un contesto, lo stato e diverse Hashmap. Ognuna di questa contiene gli attributi di una determinata categoria che il richiedente manda al sistema e che saranno valutati dalle policy. In Codice 5.13 si può vedere come è fatta una richiesta, in particolare la seconda di Codice 4.5.

Codice 5.13: Esempio di richiesta

---

```

public class ContextRequest_WriteRequestBob {
    private static ContextRequest_Status CxtReq;

```

```

public static ContextRequest_Status getContextReq() {
    if (CxtReq != null) {
        return CxtReq;
    }
    HashMap<String, Object> req_category_attribute_name = new
        HashMap<String, Object>();
    HashMap<String, Object> req_category_attribute_action = new
        HashMap<String, Object>();
    HashMap<String, Object> req_category_attribute_file = new
        HashMap<String, Object>();
    req_category_attribute_name.put("id", "Bob");
    req_category_attribute_action.put("id", "write");
    req_category_attribute_file.put("id", "file1");
    Request req = new Request("write_request");
    req.addAttribute("name", req_category_attribute_name);
    req.addAttribute("action", req_category_attribute_action);
    req.addAttribute("file", req_category_attribute_file);
    CxtReq = new ContextRequest_Status(req,
        ContextStub_Default.getInstance());
    StatusRW st = new StatusRW();
    CxtReq.setStatus(st.getStatus());
    return CxtReq;
}

}

```

---

Le Hashmap in questo caso sono tre, una per la categoria Name, una per Action e l'ultima per la categoria File.

Lo stato, contenuto all'interno della richiesta è rappresentato da una classe mostrata in Codice 5.14. Questa classe utilizza il pattern singleton per far sì che ne esista una sola istanza in tutta l'esecuzione.

Codice 5.14: Esempio di stato

```

public class StatusRW {

    private FacplStatus status;

    public StatusRW() {
        HashMap<StatusAttribute, Object> attributes = new
            HashMap<StatusAttribute, Object>();
        attributes.put(new StatusAttribute("isWriting",
            FacplStatusType.BOOLEAN), false);
    }
}

```

```

        attributes.put(new StatusAttribute("counterReadFile1",
            FacplStatusType.INT), 0);
        attributes.put(new StatusAttribute("counterReadFile2",
            FacplStatusType.INT), 0);
        status = new FacplStatus(attributes, this.getClass().getName());
    }
    public FacplStatus getStatus() {
        return status;
    }
}

```

Come la classe per modellare le richieste, lo stato utilizza una Hashmap per contenere gli attributi.

La policy è definita estendendo una classe astratta presente nella libreria. Per aggiungere tutti gli elementi è sufficiente istanziarli e usare i metodi ereditati per inserirli al suo interno. La Policy *Write* in Codice 4.3 ha al suo interno un target, una obligation ed una Rule; vediamo, in Codice 5.15, com'è il suo corrispettivo in Java.

Codice 5.15: Policy Write

```

private class PolicySet_Write extends PolicySet {
    public PolicySet_Write() {
        addId("Write_Policy");
        addCombiningAlg(it.unifi.facpl.lib.algorithm.DenyUnlessPermitGreedy.class);
        ExpressionFunction e1 = new
            ExpressionFunction(it.unifi.facpl.lib.function.comparison.Equal.class,
                "file1",
                new AttributeName("file", "id")
            );
        ExpressionFunction e2 = new
            ExpressionFunction(it.unifi.facpl.lib.function.comparison.Equal.class,
                "write",
                new AttributeName("action", "id")
            );
        ExpressionBooleanTree ebt = new
            ExpressionBooleanTree(ExprBooleanConnector.AND, e1, e2);
        addTarget(ebt);
        addPolicyElement(new Rule_write());
        addObligation(new ObligationStatus(new FlagStatus(),
            Effect.PERMIT, ObligationType.M,
            new StatusAttribute("isWriting",
                FacplStatusType.BOOLEAN), true));
    }
}

```

```

}
private class Rule_write extends Rule {
  Rule_write() {
    addId("write");
    addEffect(Effect.PERMIT);
    ExpressionFunction e1=new
      ExpressionFunction(it.unifi.facpl.lib.function.comparison.Equal.class,
        new StatusAttribute("isWriting",
          FacplStatusType.BOOLEAN),
        false);
    ExpressionFunction e2=new
      ExpressionFunction(it.unifi.facpl.lib.function.comparison.Equal.class,
        new StatusAttribute("counterReadFile1",
          FacplStatusType.INT),
        0);
    ExpressionBooleanTree ebt = new
      ExpressionBooleanTree(ExprBooleanConnector.AND, e1, e2);
    addTarget(ebt);
  }
}

```

---

Questa classe, come detto in precedenza estende una classe già esistente nella libreria, ovvero la classe `PolicySet`. In questo caso gli elementi che caratterizzano la Policy sono tre, il target, l'obligation e la rule al suo interno. I primi due vengono istanziati e aggiunti tramite metodi ereditati senza alcun problema. Per la rule invece il discorso è un po' più complicato, in quanto viene creata una classe annidata del tutto analoga a quella creata per la policy. Dopo aver creato questa classe viene semplicemente aggiunta un'istanza di essa attraverso un metodo ereditato.

#### 5.4.2 *Secondo case study*

Verrebbe abbastanza uguale, lo devo fare lo stesso? In caso non lo dovessi scrivere levo le sottosezioni e ne faccio una singola chiamata *Esempio*.

---

## CONCLUSIONI

---

Durante questa tesi è stata affrontato il lavoro di implementazione di *Usage Control* in FACPL. Come primo compito ci siamo occupati di analizzare i principali modelli dedicati all'*Access Control* e successivamente è seguita una fase di approfondimento sul modello *Usage Control* proposto da Sandhu e Park.

Il lavoro è seguito con una disamina sul linguaggio FACPL in modo da comprendere al meglio la sintassi, la semantica e soprattutto il processo di valutazione così da avere un background e sapere come, e dove, intervenire per implementare del concetto di *Status* e di tutte le cose che conseguentemente ne derivano da esso, come nuovi attributi, nuove obligation e funzioni per operare su attributi.

Nella sintassi estesa sono state aggiunte nuove regole di produzione e ne sono state modificate alcune. Quelle modificate includono la definizione del sistema, mentre quelle aggiunte riguardano nuove funzioni, ed un nuovo tipo di attributo, chiamato *Status Attribute*. Successivamente è stato svolto del lavoro sulla libreria Java per implementare queste nuove caratteristiche.

Nel capitolo 5 viene descritta l'implementazione delle nuove caratteristiche del linguaggio in Java. All'inizio del Capitolo viene descritta l'implementazione dello *Status* e di conseguenza degli *Status Attribute*. Successivamente sono stata necessaria l'implementazione delle funzioni per la modifica di questi attributi in modo da poter cambiare lo stato con l'avanzare delle valutazioni delle richieste. L'estensione ha coinvolto anche il PEPin quanto deve valutare un nuovo tipo di Obligation, ovvero le Obligation Status. La differenza con le Obligation normali risiede nel fatto che le Obligation Status possono eseguire le funzioni per la modifica dello stato mostrate all'inizio del capitolo. Durante la valutazione di una richiesta devono essere valutati anche gli attributi di stato. Permettere la

valutazione di quest'ultimi da parte delle funzioni già esistenti è stato abbastanza semplice, poiché è bastato estendere la classe che implementa il contesto in modo tale che facesse la ricerca degli attributi anche all'interno dello stato.

### 6.1 SVILUPPI FUTURI

Gli esempi mostrati durante questa tesi sono basilari, e fondamentalmente sono stati scritti con il solo scopo di provare il funzionamento delle nuove funzionalità di FACPL. Potrebbe risultare interessante applicare FACPL a casi di studio reali, in modo da poterne verificare le potenzialità sul campo.

Durante lo sviluppo sono stati implementati solo alcuni tipi di dato e relative funzioni su di essi, in futuro sarebbe facilmente possibile implementarne di nuovi in quanto la libreria è stata progettata per favorirne la rapida e semplice estendibilità. Per esempio potrebbe essere stimolante l'implementazione di un tipo nuovo come le Liste e funzioni come ricerca all'interno di esse, aggiunta ed eliminazione di elementi o il conteggio del numero di elementi.



---

## BIBLIOGRAFIA

---

- [1] *FACPL guide*. [http://facpl.sourceforge.net/guide/facpl\\_guide.html](http://facpl.sourceforge.net/guide/facpl_guide.html).
- [2] Information technology - syntactic metalanguage - extended bnf. ISO/IEC 14977.
- [3] *Java Platform, Standard Edition 8 API Specification*. <https://docs.oracle.com/javase/8/docs/api/>.
- [4] *Xtext Documentation*. <https://eclipse.org/Xtext/documentation/>.
- [5] E. Chabrow. Nist guide aims to ease access control. <http://www.bankinfosecurity.com/nist-publication-aims-to-ease-access-control-a-6612/op-1>.
- [6] A. Lazouski, G. Mancini, F. Martinelli, and P. Mori. Usage control in cloud systems. In Savage et al. [14], pages 202–207. (Cited on page 18.)
- [7] A. Lazouski, F. Martinelli, and P. Mori. Usage control in computer security: A survey. *Computer Science Review*, 4(2):81–99, 2010. (Cited on page 17.)
- [8] A. Margheri. Progetto e realizzazione di un linguaggio formale per il controllo degli accessi basato su politiche, 2011/12. Tesi di laurea magistrale in Informatica.
- [9] A. Margheri, M. Masi, R. Pugliese, and F. Tiezzi. A formal framework for specification, analysis and enforcement of access control policies. (Cited on page 28.)
- [10] NIST. A survey of access control models. [http://csrc.nist.gov/news\\_events/privilege-management-workshop/PvM-Model-Survey-Aug26-2009.pdf](http://csrc.nist.gov/news_events/privilege-management-workshop/PvM-Model-Survey-Aug26-2009.pdf). (Cited on page 11.)
- [11] J. Park. *Usage control: A unified framework for next generation access control*. PhD thesis, George Mason University, 2003.

- [12] J. Park and R. S. Sandhu. Towards usage control models: beyond traditional access control. In *SACMAT*, pages 57–64, 2002.
- [13] J. Park and R. S. Sandhu. The ucon<sub>abc</sub> usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, 2004. (Cited on page 9.)
- [14] N. Savage, S. E. Assad, and C. A. Shoniregun, editors. *7th International Conference for Internet Technology and Secured Transactions, ICITST 2012, London, United Kingdom, December 10-12, 2012*. IEEE, 2012. (Cited on page 63.)
- [15] Wikipedia. Access control list — Wikipedia, the free encyclopedia. [https://en.wikipedia.org/w/index.php?title=Access\\_control\\_list&oldid=710571612](https://en.wikipedia.org/w/index.php?title=Access_control_list&oldid=710571612), 2016. [Online; accessed 20-March-2016].
- [16] Wikipedia. Attribute-based access control — Wikipedia, the free encyclopedia. [https://en.wikipedia.org/w/index.php?title=Attribute-based\\_access\\_control&oldid=694270795](https://en.wikipedia.org/w/index.php?title=Attribute-based_access_control&oldid=694270795), 2016. [Online; accessed 20-March-2016].
- [17] Wikipedia. Role-based access control — Wikipedia, the free encyclopedia. [https://en.wikipedia.org/w/index.php?title=Role-based\\_access\\_control&oldid=709886924](https://en.wikipedia.org/w/index.php?title=Role-based_access_control&oldid=709886924), 2016. [Online; accessed 20-March-2016].

#### LISTA DI ACRONIMI

ACL	Access Control List
ABAC	Attribute Based Access Control
PBAC	Policy Based Access Control
RBAC	Role Based Access Control
PDP	Policy Decision Point
PEP	Policy Enforcement Point
XACML	eXtensible Access Control Markup Language
XML	eXtensible Markup Language

<b>FACPL</b>	Formal Access Control Policy Language
<b>ACS</b>	Access Control Policy
<b>AAS</b>	Authoritative Attribute Source
<b>PAS</b>	Policy Authorisation System
<b>IDE</b>	Integrated Development Environment
<b>PR</b>	Policy Repository
<b>EBNF</b>	Extended Backus Naur form



---

## CODICE COMPLETO

---

### A.1 CODICE DEL CAPITOLO 4

---

#### Codice A.1: Policy set completo di 4.4.1

---

```
PolicySet ReadWrite_Policy { deny- unless- permit
  target: equal ( "Bob" , name / id ) && ("Alice, name/id")
  policies:
    PolicySet Write_Policy { deny- unless- permit
      target: equal("file", file/id) && ("write", action/id)
      policies:
        Rule write ( permit target:
          equal ( status / isWriting , false ) &&
          equal ( status / counterReadFile1, 0)
        )
      obl:
        [ permit M flagStatus(isWriting, true) ]
    }
  PolicySet Read_Policy { deny- unless- permit
    target: equal("file", file/id) && ("read", action/id)
    policies:
      Rule read ( permit target:
        equal ( status / isWriting , false ) &&
        less-than (status / counterReadFile1, 2)
      )
    obl:
      [ permit M addStatus(counterReadFile1, 1) ]
    }
  PolicySet StopWrite_Policy { deny- unless- permit
    target: equal("file", file/id) && ("stopWrite", action/id)
    policies:
      Rule stopWrite ( permit target:
```

```

        equal ( status / isWriting , true )
    )
    obl:
    [ permit M flagStatus(isWriting, false) ]
}
PolicySet StopRead_Policy { deny- unless- permit
    target: equal("file", file/id) && ("stopRead", action/id)
    policies:
        Rule stopRead ( permit target:
            greater-than ( status / counterReadFile1 , 0 )
        )
    obl:
    [ permit M subStatus(counterReadFile1, 1) ]
}
}
PAS {
    pep: deny- biased
    pdp: deny- unless- permit
    status: [(boolean isWriting = false), (int counterReadFile1 = 0)]
}

```

---

## Codice A.2: Policy set completo di 4.4.2

```

PolicySet Negozio { deny- unless- permit
    target: equal ( "Bob" , name / id ) || ("Alice, name/id")
    policies:

    PolicySet Buy_Policy { deny- unless- permit
        target: equal("file1", file/id) && ("buy", action/id)
        policies:
            Rule alice_buy ( permit target: (
                equal ( action / id , "buy" ) &&
                equal ( name / id, "Alice"))
            obl:
            [ permit M setString("accessTypeAlice", "BUY") ]
        )
            Rule bob_buy ( permit target: (
                equal ( action / id , "buy" ) &&
                equal ( name / id, "Alice"))
            obl:
            [ permit M setString("accessTypeBob", "BUY") ]
        )
    }
}

```

```

PolicySet NUMBER_Policy { deny- unless- permit
  target: equal("file1", file/id) && ("number", action/id)
  policies:
    Rule alice_buy ( permit target: (
      equal ( action / id , "number" ) &&
      equal ( name / id, "Alice" ))
      obl:
        [ permit M setString("accessTypeAlice", "NUMBER") ]
        [ permit M addStatus("aliceFileIviewNumber", 2) ]
      )
    Rule bob_buy ( permit target: (
      equal ( action / id , "number" ) &&
      equal ( name / id, "Alice" ))
      obl:
        [ permit M setString("accessTypeBob", "NUMBER") ]
        [ permit M addStatus("bobFileIviewNumber", 2) ]
      )
  )
}

PolicySet TIME_Policy { deny- unless- permit
  target: equal("file1", file/id) && ("TIME", action/id)
  policies:
    Rule alice_buy ( permit target: (
      equal ( action / id , "time" ) &&
      equal ( name / id, "Alice" ))
      obl:
        [ permit M setString("accessTypeAlice", "TIME") ]
        [ permit M
          sumDate("aliceFileIexpiration", "0000/00/00-48:00:00") ]
      )
    Rule bob_buy ( permit target: (
      equal ( action / id , "time" ) &&
      equal ( name / id, "Alice" ))
      obl:
        [ permit M setString("accessTypeBob", "TIME") ]
        [ permit M
          sumDate("bobFileIexpiration", "0000/00/00-48:00:00") ]
      )
  )
}

PolicySet VIEW { deny- unless- permit
  target: equal("file1", file/id) && ("view", action/id)
  policies:

```

```

Rule buy ( permit target: (
    equal ( status / accessTypeBob , "BUY" ) &&
    equal ( status / accessTypeAlice, "BUY"))
)
Rule number_alice ( permit target: (
    equal ( status / accessTypeAlice, "NUMBER" ) &&
    equal ( name / id, "Alice") && greater-than( status /
        aliceFileIviewNumber, 0))
    obl:
    [ permit M subStatus("aliceFileIviewNumber", 1) ]
)
Rule number_bob ( permit target: (
    equal ( status / accessTypeBob, "NUMBER" ) &&
    equal ( name / id, "Bob") && greater-than( status /
        bobFileIviewNumber, 0))
    obl:
    [ permit M subStatus("bobFileIviewNumber", 1) ]
)
Rule time_alice ( permit target: (
    equal ( status / accessTypeAlice, "TIME" ) &&
    equal ( name / id, "Alice") && greater-than( status /
        aliceFileIexpiration, today))
)
Rule number_alice ( permit target: (
    equal ( status / accessTypeAlice, "TIME" ) &&
    equal ( name / id, "Bob") && greater-than( status /
        bobFileIexpiration, today))
)
}

}
PAS {
    Combined Decision : false ;
    Extended Indeterminate : false ;
    Java Package : "example" ;
    Requests To Evaluate : Request1, Request2, Request3, Request4,
        Request5, Request6 ;
    pep: deny- biased
    pdp: deny- unless- permit
    status: [ (date aliceFileIexpiration = today), (date

```



```

        bobFileExpiration = today),
        (int bobFileViewNumber = 0), (int aliceFileViewNumber = 0),
        (String accessTypeAlice = "no"), (String accessTypeBob = "no")]
    }

```

---

## A.2 CODICE DEL CAPITOLO 5

### Codice A.3: Richiesta di Sezione 5.4.1

---

```

package primoEsempioStatus;

import java.util.ArrayList;
import java.util.HashMap;

import it.unifi.facpl.lib.context.ContextRequest_Status;
import it.unifi.facpl.lib.context.ContextStub_Default;

import it.unifi.facpl.lib.context.Request;
import it.unifi.facpl.lib.enums.FacplStatusType;
import it.unifi.facpl.system.status.FacplStatus;
import it.unifi.facpl.system.status.StatusAttribute;

@SuppressWarnings("all")
public class ContextRequest_WriteRequestAlice {
    private static ContextRequest_Status CxtReq;

    public static ContextRequest_Status getContextReq() {
        if (CxtReq != null) {
            return CxtReq;
        }
        // create map for each category
        HashMap<String, Object> req_category_attribute_name = new
            HashMap<String, Object>();
        HashMap<String, Object> req_category_attribute_action = new
            HashMap<String, Object>();
        HashMap<String, Object> req_category_attribute_file = new
            HashMap<String, Object>();
        // add attribute's values
        req_category_attribute_name.put("id", "Alice");
        req_category_attribute_action.put("id", "write");
        req_category_attribute_file.put("id", "file1");
    }
}

```

```

    // add attributes to request
    Request req = new Request("write_request");
    req.addAttribute("name", req_category_attribute_name);
    req.addAttribute("action", req_category_attribute_action);
    req.addAttribute("file", req_category_attribute_file);
    // context stub: default-one
    CxtReq = new ContextRequest_Status(req,
        ContextStub_Default.getInstance());
    StatusRW st = new StatusRW();
    CxtReq.setStatus(st.getStatus());
    return CxtReq;
}
}

```

---

#### Codice A.4: Richiesta di Sezione 5.4.1

---

```

public class ContextRequest_WriteRequestBob {
    private static ContextRequest_Status CxtReq;
    public static ContextRequest_Status getContextReq() {
        if (CxtReq != null) {
            return CxtReq;
        }
        HashMap<String, Object> req_category_attribute_name = new
            HashMap<String, Object>();
        HashMap<String, Object> req_category_attribute_action = new
            HashMap<String, Object>();
        HashMap<String, Object> req_category_attribute_file = new
            HashMap<String, Object>();
        req_category_attribute_name.put("id", "Bob");
        req_category_attribute_action.put("id", "write");
        req_category_attribute_file.put("id", "file1");
        Request req = new Request("write_request");
        req.addAttribute("name", req_category_attribute_name);
        req.addAttribute("action", req_category_attribute_action);
        req.addAttribute("file", req_category_attribute_file);
        CxtReq = new ContextRequest_Status(req,
            ContextStub_Default.getInstance());
        StatusRW st = new StatusRW();
        CxtReq.setStatus(st.getStatus());
        return CxtReq;
    }
}

```

```
}

```

---

#### Codice A.5: Richiesta di Sezione 5.4.1

---

```
package primoEsempioStatus;

import java.util.ArrayList;
import java.util.HashMap;

import it.unifi.facpl.lib.context.ContextRequest_Status;
import it.unifi.facpl.lib.context.ContextStub_Default;

import it.unifi.facpl.lib.context.Request;
import it.unifi.facpl.lib.enums.FacplStatusType;
import it.unifi.facpl.system.status.FacplStatus;
import it.unifi.facpl.system.status.StatusAttribute;

@SuppressWarnings("all")
public class ContextRequest_StopWriteRequest {
    private static ContextRequest_Status CxtReq;

    public static ContextRequest_Status getContextReq() {
        if (CxtReq != null) {
            return CxtReq;
        }
        // create map for each category
        HashMap<String, Object> req_category_attribute_name = new
            HashMap<String, Object>();
        HashMap<String, Object> req_category_attribute_action = new
            HashMap<String, Object>();
        // add attribute's values
        req_category_attribute_name.put("id", "Alice");
        req_category_attribute_action.put("id", "stopWrite");
        // add attributes to request
        Request req = new Request("stop_write_request");
        req.addAttribute("name", req_category_attribute_name);
        req.addAttribute("action", req_category_attribute_action);
        // context stub: default-one
        CxtReq = new ContextRequest_Status(req,
            ContextStub_Default.getInstance());
        StatusRW st = new StatusRW();
        CxtReq.setStatus(st.getStatus());
    }
}
```

```

        return CxtReq;
    }

}

```

---

#### Codice A.6: Richiesta di Sezione 5.4.1

---

```

package primoEsempioStatus;

import java.util.ArrayList;
import java.util.HashMap;

import it.unifi.facpl.lib.context.ContextRequest_Status;
import it.unifi.facpl.lib.context.ContextStub_Default;
import it.unifi.facpl.lib.context.Request;
import it.unifi.facpl.lib.enums.FacplStatusType;
import it.unifi.facpl.system.status.FacplStatus;
import it.unifi.facpl.system.status.StatusAttribute;

@SuppressWarnings("all")
public class ContextRequest_StopReadRequestBob {
    private static ContextRequest_Status CxtReq;

    public static ContextRequest_Status getContextReq() {
        if (CxtReq != null) {
            return CxtReq;
        }
        // create map for each category
        HashMap<String, Object> req_category_attribute_name = new
            HashMap<String, Object>();
        HashMap<String, Object> req_category_attribute_action = new
            HashMap<String, Object>();
        HashMap<String, Object> req_category_attribute_file = new
            HashMap<String, Object>();
        // add attribute's values
        req_category_attribute_name.put("id", "Alice");
        req_category_attribute_action.put("id", "stopRead");
        req_category_attribute_file.put("id", "file1");
        // add attributes to request
        Request req = new Request("stop_write_request");
        req.addAttribute("name", req_category_attribute_name);
        req.addAttribute("action", req_category_attribute_action);
    }
}

```

```

    req.addAttribute("file", req_category_attribute_file);
    // context stub: default-one
    CxtReq = new ContextRequest_Status(req,
        ContextStub_Default.getInstance());
    StatusRW st = new StatusRW();
    CxtReq.setStatus(st.getStatus());
    return CxtReq;
}
}

```

---

Codice A.7: Richiesta di Sezione 5.4.1

---

```

package primoEsempioStatus;

import java.util.ArrayList;
import java.util.HashMap;

import it.unifi.facpl.lib.context.ContextRequest_Status;
import it.unifi.facpl.lib.context.ContextStub_Default;
import it.unifi.facpl.lib.context.Request;
import it.unifi.facpl.lib.enums.FacplStatusType;
import it.unifi.facpl.system.status.FacplStatus;
import it.unifi.facpl.system.status.StatusAttribute;

@SuppressWarnings("all")
public class ContextRequest_StopReadRequestAlice {
    private static ContextRequest_Status CxtReq;

    public static ContextRequest_Status getContextReq() {
        if (CxtReq != null) {
            return CxtReq;
        }
        // create map for each category
        HashMap<String, Object> req_category_attribute_name = new
            HashMap<String, Object>();
        HashMap<String, Object> req_category_attribute_action = new
            HashMap<String, Object>();
        HashMap<String, Object> req_category_attribute_file = new
            HashMap<String, Object>();
        // add attribute's values
        req_category_attribute_name.put("id", "Alice");
        req_category_attribute_action.put("id", "stopRead");
    }
}

```

```

    req_category_attribute_file.put("id", "file1");
    // add attributes to request
    Request req = new Request("stop_read_request");
    req.addAttribute("name", req_category_attribute_name);
    req.addAttribute("action", req_category_attribute_action);
    req.addAttribute("file", req_category_attribute_file);
    // context stub: default-one
    CxtReq = new ContextRequest_Status(req,
        ContextStub_Default.getInstance());
    StatusRW st = new StatusRW();
    CxtReq.setStatus(st.getStatus());
    return CxtReq;
}
}

```

---

#### Codice A.8: Richiesta di Sezione 5.4.1

---

```

package primoEsempioStatus;

import java.util.ArrayList;
import java.util.HashMap;

import counterExample.Status_1;
import it.unifi.facpl.lib.context.ContextRequest_Status;
import it.unifi.facpl.lib.context.ContextStub_Default;
import it.unifi.facpl.lib.context.Request;
import it.unifi.facpl.lib.enums.FacplStatusType;
import it.unifi.facpl.system.status.FacplStatus;
import it.unifi.facpl.system.status.StatusAttribute;

@SuppressWarnings("all")
public class ContextRequest_ReadRequestAlice {
    private static ContextRequest_Status CxtReq;

    public static ContextRequest_Status getContextReq() {
        if (CxtReq != null) {
            return CxtReq;
        }
        // create map for each category
        HashMap<String, Object> req_category_attribute_name = new
            HashMap<String, Object>();
        HashMap<String, Object> req_category_attribute_action = new

```

```

        HashMap<String, Object>();
        HashMap<String, Object> req_category_attribute_file = new
            HashMap<String, Object>();
        // add attribute's values
        req_category_attribute_name.put("id", "Alice");
        req_category_attribute_action.put("id", "read");
        req_category_attribute_file.put("id", "file1");
        // add attributes to request
        Request req = new Request("read_request");
        req.addAttribute("name", req_category_attribute_name);
        req.addAttribute("action", req_category_attribute_action);
        req.addAttribute("file", req_category_attribute_file);
        // context stub: default-one
        CxtReq = new ContextRequest_Status(req,
            ContextStub_Default.getInstance());
        StatusRW st = new StatusRW();
        CxtReq.setStatus(st.getStatus());
        return CxtReq;
    }
}

```

---

#### Codice A.9: Richiesta di Sezione 5.4.1

---

```

package primoEsempioStatus;

import java.util.ArrayList;
import java.util.HashMap;

import it.unifi.facpl.lib.context.ContextRequest_Status;
import it.unifi.facpl.lib.context.ContextStub_Default;
import it.unifi.facpl.lib.context.Request;
import it.unifi.facpl.lib.enums.FacplStatusType;
import it.unifi.facpl.system.status.FacplStatus;
import it.unifi.facpl.system.status.StatusAttribute;

@SuppressWarnings("all")
public class ContextRequest_ReadRequestBob {
    private static ContextRequest_Status CxtReq;

    public static ContextRequest_Status getContextReq() {

```

```

    if (CxtReq != null) {
        return CxtReq;
    }
    // create map for each category
    HashMap<String, Object> req_category_attribute_name = new
        HashMap<String, Object>();
    HashMap<String, Object> req_category_attribute_action = new
        HashMap<String, Object>();
    HashMap<String, Object> req_category_attribute_file = new
        HashMap<String, Object>();
    // add attribute's values
    req_category_attribute_name.put("id", "Bob");
    req_category_attribute_action.put("id", "read");
    req_category_attribute_file.put("id", "file1");
    // add attributes to request
    Request req = new Request("read_request");
    req.addAttribute("name", req_category_attribute_name);
    req.addAttribute("action", req_category_attribute_action);
    req.addAttribute("file", req_category_attribute_file);
    // context stub: default-one
    CxtReq = new ContextRequest_Status(req,
        ContextStub_Default.getInstance());
    StatusRW st = new StatusRW();
    CxtReq.setStatus(st.getStatus());
    return CxtReq;
}
}

```

---

## Codice A.10: Main di Sezione 5.4.1

```

public class MainFACPL {
    private PDP pdp;
    private PEP pep;
    public MainFACPL() throws MissingAttributeException {
        LinkedList<FacplPolicy> policies = new
            LinkedList<FacplPolicy>();
        policies.add(new PolicySet_ReadWrite());
        this.pdp = new
            PDP(it.unifi.facpl.lib.algorithm.PermitUnlessDenyGreedy.class,
                policies, false);
        this.pep = new PEP(EnforcementAlgorithm.DENY_BIASED);
        this.pep.addPEPActions(PEPAction.getPepActions());
    }
}

```



```

}
public static void main(String[] args) throws
    MissingAttributeException {
    MainFACPL system = new MainFACPL();
    StringBuffer result = new StringBuffer();
    LinkedList<ContextRequest_Status> requests = new
        LinkedList<ContextRequest_Status>();
    requests.add(ContextRequest_ReadRequestAlice.getContextReq());
    requests.add(ContextRequest_ReadRequestAlice.getContextReq());
    requests.add(ContextRequest_ReadRequestAlice.getContextReq());
    requests.add(ContextRequest_WriteRequestBob.getContextReq());
    requests.add(ContextRequest_ReadRequestBob.getContextReq());
    requests.add(ContextRequest_StopReadRequestAlice.getContextReq());
    requests.add(ContextRequest_StopReadRequestBob.getContextReq());
    requests.add(ContextRequest_WriteRequestAlice.getContextReq());
    for (ContextRequest_Status rcxt : requests) {
        result.append("-----");
        AuthorisationPDP resPDP = system.pdp.doAuthorisation(rcxt);
        result.append("\nRequest: " + resPDP.getId() + "\n\n");
        result.append("PDP Decision=\n " + resPDP.toString() +
            "\n\n");
        AuthorisationPEP resPEP =
            system.pep.doEnforcement(resPDP, rcxt);
        result.append("time per request "+(endR-startR));
        result.append("\nPEP Decision=\n " + resPEP.toString() +
            "\n");
        result.append("-----");
    }
    result.append("\ntime for all requests "+(end-start));
    System.out.println(result.toString());
}
}

```

---

## Codice A.11: Policy di Sezione 5.4.1

```

public class PolicySet_ReadWrite extends PolicySet {
    public PolicySet_ReadWrite(){
        addId("ReadWrite_Policy");
        addCombiningAlg(it.unifi.facpl.lib.algorithm.DenyUnlessPermitGreedy.class);
        ExpressionFunction e1 = new
            ExpressionFunction(it.unifi.facpl.lib.function.comparison.Equal.class,
                "Bob",
                new AttributeName("name", "id"));
    }
}

```

```

ExpressionFunction e2 = new
    ExpressionFunction(it.unifi.facpl.lib.function.comparison.Equal.class,
        "Alice",
        new AttributeName("name", "id"));
ExpressionBooleanTree ebt = new
    ExpressionBooleanTree(ExprBooleanConnector.OR, e1, e2);
addTarget(ebt);
addPolicyElement(new PolicySet_Write());
addPolicyElement(new PolicySet_Read());
addPolicyElement(new PolicySet_StopWrite());
addPolicyElement(new PolicySet_StopRead());
}

private class PolicySet_Write extends PolicySet {
    public PolicySet_Write() {
        addId("Write_Policy");
        addCombiningAlg(it.unifi.facpl.lib.algorithm.DenyUnlessPermitGreedy.class);
        ExpressionFunction e1 = new
            ExpressionFunction(it.unifi.facpl.lib.function.comparison.Equal.class,
                "file1",
                new AttributeName("file", "id")
            );
        ExpressionFunction e2 = new
            ExpressionFunction(it.unifi.facpl.lib.function.comparison.Equal.class,
                "write",
                new AttributeName("action", "id")
            );
        ExpressionBooleanTree ebt = new
            ExpressionBooleanTree(ExprBooleanConnector.AND, e1, e2);
        addTarget(ebt);
        addPolicyElement(new Rule_write());
        addObligation(new ObligationStatus(new FlagStatus(),
            Effect.PERMIT, ObligationType.M,
            new StatusAttribute("isWriting",
                FacplStatusType.BOOLEAN), true));
    }
}

private class Rule_write extends Rule {
    Rule_write() {
        addId("write");
        addEffect(Effect.PERMIT);
        ExpressionFunction e1=new
            ExpressionFunction(it.unifi.facpl.lib.function.comparison.Equal.class,

```

```

        new StatusAttribute("isWriting",
            FacplStatusType.BOOLEAN),
        false);
    ExpressionFunction e2=new
        ExpressionFunction(it.unifi.facpl.lib.function.comparison.Equal.class,
            new StatusAttribute("counterReadFile1",
                FacplStatusType.INT),
            0);
    ExpressionBooleanTree ebt = new
        ExpressionBooleanTree(ExprBooleanConnector.AND, e1, e2);
    addTarget(ebt);
    }
}
}

private class PolicySet_Read extends PolicySet {

    public PolicySet_Read() {

        addId("Read_Policy");
        // Algorithm Combining
        addCombiningAlg(it.unifi.facpl.lib.algorithm.DenyUnlessPermitGreedy.class);
        // Target
        ExpressionFunction e1 = new
            ExpressionFunction(it.unifi.facpl.lib.function.comparison.Equal.class,
                "file1",
                new AttributeName("file", "id")
            );
        ExpressionFunction e2 = new
            ExpressionFunction(it.unifi.facpl.lib.function.comparison.Equal.class,
                "read",
                new AttributeName("action", "id")
            );
        ExpressionBooleanTree ebt = new
            ExpressionBooleanTree(ExprBooleanConnector.AND, e1, e2);
        addTarget(ebt);
        // PolElements
        addPolicyElement(new Rule_read());
        // Obligation
        addObligation(new ObligationStatus(new AddStatus(),
            Effect.PERMIT, ObligationType.M,

```

```

        new StatusAttribute("counterReadFile1",
            FacplStatusType.INT), 1));
    }

    private class Rule_read extends Rule {

        Rule_read() {
            addId("read");
            // Effect
            addEffect(Effect.PERMIT);
            ExpressionFunction e1 = new
                ExpressionFunction(it.unifi.facpl.lib.function.comparison.Equal.class,
                    new StatusAttribute("isWriting",
                        FacplStatusType.BOOLEAN),
                    false); //nessuno scrive
            ExpressionFunction e2 = new
                ExpressionFunction(it.unifi.facpl.lib.function.comparison.LessThan.class,
                    new StatusAttribute("counterReadFile1",
                        FacplStatusType.INT),
                    2); //i lettori sono meno di due

            ExpressionBooleanTree ebt = new
                ExpressionBooleanTree(ExprBooleanConnector.AND, e1, e2);
            addTarget(ebt);
        }
    }

}

private class PolicySet_StopRead extends PolicySet {

    public PolicySet_StopRead() {

        addId("StopRead_Policy");
        // Algorithm Combining
        addCombiningAlg(it.unifi.facpl.lib.algorithm.DenyUnlessPermitGreedy.class);
        // Target
        ExpressionFunction e1 = new
            ExpressionFunction(it.unifi.facpl.lib.function.comparison.Equal.class,
                "file1",
                new AttributeName("file", "id")

```

```

    );
    ExpressionFunction e2 = new
        ExpressionFunction(it.unifi.facpl.lib.function.comparison.Equal.class,
            "stopRead",
            new AttributeName("action", "id")
        );
    ExpressionBooleanTree ebt = new
        ExpressionBooleanTree(ExprBooleanConnector.AND, e1, e2);
    addTarget(ebt);
    // PolElements
    addPolicyElement(new Rule_stopRead());
    // Obligation
    addObligation(new ObligationStatus(new SubStatus(),
        Effect.PERMIT, ObligationType.M,
            new StatusAttribute("counterReadFile1",
                FacplStatusType.INT), 1)); //meno uno sui lettori
}

private class Rule_stopRead extends Rule {

    Rule_stopRead() {
        addId("stopRead");
        // Effect
        addEffect(Effect.PERMIT);
        addTarget(new
            ExpressionFunction(it.unifi.facpl.lib.function.comparison.GreaterThan.class,
                new StatusAttribute("counterReadFile1",
                    FacplStatusType.INT),
                0)); //solo se ci sono lettori
    }
}

private class PolicySet_StopWrite extends PolicySet {

    public PolicySet_StopWrite(){
        addId("StopWrite_Policy");
        // Algorithm Combining
        addCombiningAlg(it.unifi.facpl.lib.algorithm.DenyUnlessPermitGreedy.class);
        // Target
        ExpressionFunction e1 = new

```

```

        ExpressionFunction(it.unifi.facpl.lib.function.comparison.Equal.class,
            "file1",
            new AttributeName("file", "id")
        );
    ExpressionFunction e2 = new
        ExpressionFunction(it.unifi.facpl.lib.function.comparison.Equal.class,
            "stopWrite",
            new AttributeName("action", "id")
        );
    ExpressionBooleanTree ebt = new
        ExpressionBooleanTree(ExprBooleanConnector.AND, e1, e2);
    addTarget(ebt);
    // PolElements
    addPolicyElement(new Rule_write());
    // Obligation
    addObligation(new ObligationStatus(new FlagStatus(),
        Effect.PERMIT, ObligationType.M,
            new StatusAttribute("isWriting",
                FacplStatusType.BOOLEAN), false));
}

private class Rule_write extends Rule {

    Rule_write() {
        addId("stopWrite");
        // Effect
        addEffect(Effect.PERMIT);
        addTarget(new
            ExpressionFunction(it.unifi.facpl.lib.function.comparison.Equal.class,
                new StatusAttribute("isWriting",
                    FacplStatusType.BOOLEAN),
                true));
    }
}
}
}
}

```

Codice A.12: Status 5.4.1

---

```

public class StatusRW {

    private FacplStatus status;

```

```
public StatusRW() {
    HashMap<StatusAttribute, Object> attributes = new
        HashMap<StatusAttribute, Object>();
    attributes.put(new StatusAttribute("isWriting",
        FacplStatusType.BOOLEAN), false);
    attributes.put(new StatusAttribute("counterReadFile1",
        FacplStatusType.INT), 0);
    attributes.put(new StatusAttribute("counterReadFile2",
        FacplStatusType.INT), 0);
    status = new FacplStatus(attributes, this.getClass().getName());
}
public FacplStatus getStatus() {
    return status;
}
}
```

---