Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

# Progettazione e realizzazione di un algoritmo per il controllo degli accessi basato sul modello MPP-ABAC

*Mario Ciampi, Angelo Esposito, Mario Sicuranza*

**RT-ICAR-NA-2014-01** **Gennaio 2014**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Napoli, Via P. Castellino 111, I-80131 Napoli, Tel: +39-0816139508, Fax: +39-0816139531, e-mail: napoli@icar.cnr.it, URL: *www.na.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

# Progettazione e realizzazione di un algoritmo per il controllo degli accessi basato sul modello MPP-ABAC

*Mario Ciampi, Angelo Esposito, Mario Sicuranza*

***Rapporto Tecnico N:*** **RT-ICAR-NA-2014-01** ***Data:*** **Gennaio 2014**

---

*I rapporti tecnici dell'ICAR-CNR sono pubblicati dall'Istituto di Calcolo e Reti ad Alte Prestazioni del Consiglio Nazionale delle Ricerche. Tali rapporti, approntati sotto l'esclusiva responsabilità scientifica degli autori, descrivono attività di ricerca del personale e dei collaboratori dell'ICAR, in alcuni casi in un formato preliminare prima della pubblicazione definitiva in altra sede.*

# Progettazione e realizzazione di un algoritmo per il controllo degli accessi basato sul modello MPP-ABAC

Mario Ciampi, Angelo Esposito, Mario Sicuranza

Istituto di Calcolo e Reti ad Alte Prestazioni, Consiglio Nazionale delle Ricerche
Via Pietro Castellino, 111 – 80131 Napoli, Italia
E-mail: {mario.ciampi, angelo.esposito, mario.sicuranza}@na.icar.cnr.it

**Abstract**

Il "controllo degli accessi" è un sistema di sicurezza comprendente un insieme di meccanismi che sono in grado, a valle dei processi di identificazione ed autenticazione di un utente, di assicurare che quest'ultimo possa compiere tutte e solo le azioni per cui è autorizzato, consentendo quindi di definire chi può fare cosa. In questo lavoro è stata dimostrata la fattibilità implementativa di un modello di controllo degli accessi definito sulla base dei requisiti specifici per sistemi di Electronic Health Record.

Il Multilevel Patient Privacy-centric Attribute Based Access Control (MPP-ABAC) [1] è modello di controllo degli accessi (AC) avanzato che combina diverse soluzioni di AC note in letteratura, aggiungendo ad esse le caratteristiche di semplicità nella gestione delle politiche di accesso, focalizzando l'attenzione sulla privacy e sul consenso del paziente.

La proprietà di modularità degli algoritmi basati sul modello MPP-ABAC ha consentito la sua implementazione secondo un pattern architetturale di tipo Model-View-Controller (MVC). In particolare, tale pattern è stato realizzato adoperando Struts 2, un framework di sviluppo per applicazioni web molto interessante da un punto di vista tecnologico e che risulta particolarmente adatto per implementazioni di tipo prototipale.

Questo lavoro ha dimostrato la fattibilità di realizzazione di un algoritmo per il controllo degli accessi secondo il modello summenzionato ed apre ad una nuova fase di sperimentazione, testing e misurazione di performace dell'algoritmo stesso.

**Keywords**: EHR, Access Control Model, MPP-ABAC, Patient Centric, MVC.

## Introduzione

I sistemi di Electronic Health Record (EHR), in italiano Fascicolo Sanitario Elettronico, sono piattaforme tecnologiche che consentono la raccolta e la condivisione dei dati e documenti sanitari e socio-sanitari elettronici tra diverse organizzazioni sanitarie.

Un sistema di EHR fornisce una varietà di servizi per gestire i dati ed i documenti, ed un certo numero di servizi di alto livello in grado di ridurre gli errori medici e migliorare la qualità delle cure. Iakovidis [2] ha definito un Electronic Health Record come "le informazioni sanitarie memorizzate digitalmente sulla vita di un individuo con lo scopo di sostenere la continuità di cura, istruzione e ricerca, e garantendo la riservatezza in ogni momento di tali informazioni". Il core di un sistema di EHR è rappresentato dai dati e documenti dei pazienti e la possibilità di accedere ad essi e di utilizzarli. I dati e documenti presenti nei sistemi di EHR sono caratterizzati da informazioni sensibili, per cui è necessario che siano protetti da accessi non autorizzati. Pertanto, per tali tipi di sistemi è fondamentale garantire la riservatezza dei dati e della privacy del paziente. Allo scopo di soddisfare tali requisiti, un meccanismo molto utilizzato è l'Access Control (AC), che è una barriera di protezione fondamentale per la sicurezza dei dati in un sistema informativo. L'AC è un meccanismo che limita l'accesso alle informazioni, individuando altresì quali operazioni è possibile compiere su classi di dati o documenti.

I sistemi di EHR dovrebbero consentire la definizione di politiche di sicurezza (tramite il modello AC) che soddisfano una serie di esigenze derivanti: i) dalla specifica organizzazione sanitaria che gestisce i dati/documenti; ii) dalla volontà del paziente; iii), dalle norme e direttive in termini di protezione dei dati clinici e consenso del paziente.

In letteratura esistono numerosi modelli di controllo di accesso, ognuno con caratteristiche diverse ma tutti con l'obiettivo comune di proteggere i dati da accessi non autorizzati. Questi comprendono i modelli Mandatory Access

Control (MAC), Discretionary Access Control (DAC) e Role Based Access Control (RBAC) [3], a partire dai quali sono stati definiti modelli più complessi e vicini alle esigenze di privacy, come ad esempio i modelli Privacy-aware Role Based Access Control (P-RBAC) [4] ed il Purpose-Based Access Control Modello (P-BAC).

Ognuno di questi risponde solo parzialmente alle esigenze di privacy del paziente per sistemi di EHR, e la maggior parte di essi presentano limiti nella possibilità di gestire in maniera accurata e flessibile le politiche di sicurezza. Al contrario, lo scopo del modello Multilevel Patient Privacy-centric Attribute Based Access Control (MPP-ABAC) è di ottenere la massima uguaglianza tra ciò che i criteri di accesso consentono di determinare e ciò che i pazienti intendono definire.

Il MPP-ABAC è un modello di controllo di accesso avanzato che combina diverse soluzioni di AC note in letteratura, aggiungendo ad esse le caratteristiche di semplicità nella gestione delle politiche di accesso, e focalizzando l'attenzione sulla privacy e sul consenso del paziente (Patient Privacy-centric). Il modello, in particolare, è caratterizzato dalle seguenti peculiarità: (i) basato su attributi; (ii) multilivello; (iii) modulare; (iv) con una gestione dinamica e temporale delle liste degli utenti ai quali viene permesso o negato l'accesso ai dati e documenti di un sistema di EHR.

Il presente rapporto tecnico è organizzato come descritto di seguito. La sezione 1 presenta sinteticamente il modello di controllo degli accessi MPP-ABAC. La sezione 2 descrive l'algoritmo implementato sulla base del modello MPP-ABAC oggetto del lavoro. Nella sezione 3 sono illustrate le caratteristiche principali del modello MVC. La sezione 4 descrive brevemente la piattaforma Apache Struts 2. La sezione 5 evidenzia le scelte intraprese, sia relative alla fase di progettazione che di sviluppo software. Infine, la sezione 6 conclude il rapporto tecnico.

## 1. Il modello MPP-ABAC

Il modello MPP-ABAC è un modello per il controllo degli accessi che, rispetto ad altri modelli, è contraddistinto dalla capacità di essere "patient privacy-centric", ossia è in grado di fornire una gestione delle politiche di sicurezza che si basa fortemente sulla volontà del paziente. Tale volontà può essere espressa dal consenso del paziente indicata all'organizzazione sanitaria quando un dato o documento viene inserito nel sistema di EHR, o direttamente dal paziente che opera sul sistema (ad esempio attraverso una Graphical User Interface o GUI). In questo modo, il paziente è in grado di esprimere le proprie politiche di accesso a seconda delle sue esigenze di privacy. Questo modello estende il classico modello RBAC con ulteriori componenti, al fine di ottenere una soluzione multilivello basata su un set di attributi degli utenti di sistema con una gestione dinamica delle politiche di sicurezza stesse. In Fig. 1 è riportata una rappresentazione grafica del modello, dove le componenti in bianco sono proprie del modello RBAC.

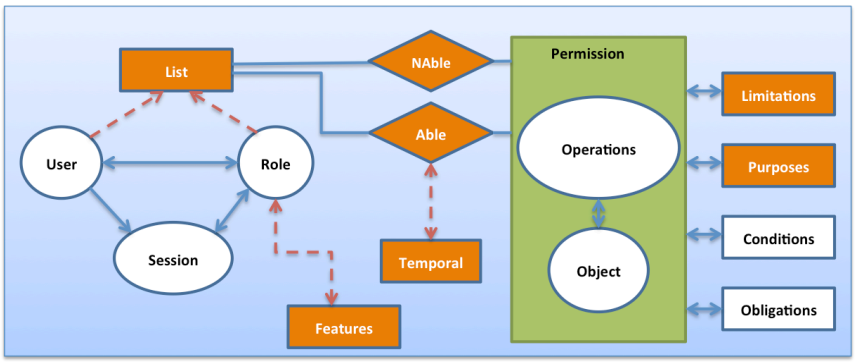


**Figura 1 – Modello MPP-RBAC**

Le componenti fondamentali del modello sono descritte di seguito.

- La componente **List** permette di specificare l'associazione tra gli utenti e i permessi sugli oggetti **(**mediante le relazioni **Able** e **Nable**). In questo modo, il modello permette di indicare facilmente quali utenti possono e non possono operare sugli oggetti.
- La componente **Temporal** permette la gestione dei diritti di accesso a seconda delle "condizioni temporali", similmente a quanto avviene nel modello Temporal Model-Based Access Control [5].
- La componente **Permission** (derivante dal modello RBAC) permette di specificare i diritti di accesso, cioè, di definire quali operazioni sono permesse su vari oggetti.

- La componente **Purposes** consente l'associazione tra un oggetto e gli scopi di memorizzazione per i quali tale oggetto è stato memorizzato. Tale componente è in grado di limitare l'accesso agli oggetti unicamente per gli scopi di memorizzazione indicati
- La componente **Features** permette di associare particolari attributi di sicurezza ai ruoli sul sistema.
- La componente **Limitation** è capace di indicare restrizioni in relazione alle associazioni Purpose-Permission e List-Permission. Essa consente quindi alle organizzazioni sanitarie di limitare il paziente in alcune scelte.

Utilizzando le diverse componenti del modello, le organizzazioni sanitarie possono, facilmente e in maniera flessibile, definire e gestire le policy di accesso nel rispetto del consenso espresso dal paziente.

## 2. Algoritmo di controllo degli accessi basato sul modello MPP-ABAC

In questo lavoro è stato implementato un possibile algoritmo di controllo degli accessi basato sul modello MPP-ABAC descritto nella sezione precedente. L'algoritmo permette di concedere o negare l'accesso ad un oggetto sulla base degli input che riceve. Gli input di tale algoritmo sono i seguenti:

- *ObjectIdentifier*: l'identificativo univoco di un documento sanitario richiesto da un utente;
- *UserIdentifier*: l'identificativo univoco del soggetto che richiede l'accesso;
- *Role*: il ruolo associato all'utente che intende accedere al documento;
- *Operation*: l'operazione richiesta sull'oggetto;
- *AccessPurpose*: lo scopo per il quale il documento è stato richiesto.

L'output dell'algoritmo è *Permit* solo se tutti i controlli sull'accesso al documento sono soddisfatti. Infatti, l'algoritmo effettua una serie di controlli in cascata eseguiti da diverse funzioni. Tuttavia, in modalità *emergency mode*, l'algoritmo elude una serie di controlli per garantire l'accesso all'oggetto richiesto (per esempio, in tale modalità viene evitato di controllare se il ruolo dell'utente è abilitato all'accesso al documento richiesto), coerentemente con quanto specificato nelle normative vigenti.

L'algoritmo di controllo è descritto di seguito.

```
Input:  Object id, user id, role, operation, access purpose.
Output: decision {Permit, Deny}

switch (document.levelsecurity)
Case topsecret:
       If(checkAccessTopSecret(user, object))
       then return PERMIT;
       else return DENY;
       end if
break;
Case secret:
       If(checkAccessSecret(user, object))
       then return PERMIT;
       else return DENY;
       end if
break;
Case normal :
       if AccessPurposes= ”Emergency”
       then if checkinEmergency(object,role)
       then return PERMIT;
       else return DENY;
       end if
       end if
 if checkNAble(user,object,operation)
 then return DENY;
       end if
```

```
        if not(checkPurpose(AccessPurposes, object, operation))
        then return DENY;
        end if
        if checkList(user, role, object, operation)
        then if checkCondition(operation, object, <condition>)
                    then return PERMIT;
              else return DENY;
              end if
        else return DENY;
        end if
break;
```

Le funzioni utilizzate nell'algoritmo sono:

- *The checkAccess (user, object) function*
  Riceve come input l'identificativo dell'utente e l'identificativo dell'oggetto richiesto. La funzione verifica se l'utente che ha richiesto l'oggetto è conforme alle politiche di sicurezza relative al *security level* del documento richiesto (secret o top-secret).
- *The checkinEmergency (object, role) function*
  Riceve come input l'identificativo dell'oggetto richiesto ed il ruolo dell'utente. La funzione verifica che gli scopi di memorizzazione del documento comprendano lo stato di *Emergenza* (ciò avviene attraverso la componente **Purpose** del modello). Successivamente viene verificato dalla stessa funzione se il ruolo dell'utente può accedere in emergenza (questa verifica avviene attraverso la componente **Features**). Questa funzione permette un controllo flessibile nel caso di accesso in emergenza al documento, supportando il modello denominato *Break the Glass* [6]. Infatti, in tal caso non viene effettuata una verifica di accesso mediante le componenti **Object-List-Operation**, ma la verifica viene fatta direttamente attraverso le componenti **Features** and **Purposes**. Inoltre, le condizioni di accesso (espresse attraverso la componente **Condition**) vengono rilassate. Ovviamente, le operazioni eseguite in *emergency mode* sono associate ad obblighi, come ad esempio la memorizzazione di log, i quali possono essere espressi attraverso la componente **Obbligations**.
- *The checkNAble (user, object, operation) function*
  Riceve come input l'identificativo dell'utente, l'identificativo dell'oggetto richiesto e l'operazione. La funzione verifica se l'identificativo dell'utente è presente in una delle liste di utenti non abilitati all'accesso al documento per l'operazione richiesta. Se l'utente è in una delle liste di utenti non abilitati, la funzione restituisce *Deny*.
- *The checkPurpose (Access Purpose, object, operation) function*
  Riceve come input lo scopo di accesso, l'identificativo dell'oggetto richiesto e l'operazione che l'utente vuole effettuare sull'oggetto. La funzione verifica che lo scopo di accesso è conforme ad uno degli scopi di memorizzazione dell'oggetto rispetto all'operazione richiesta. Se la verifica ha successo, la funzione restituisce *True*.
- *The checklist (user, role, object, operation) function*
  Riceve come input l'identificativo dell'utente, il ruolo, l'identificativo dell'oggetto richiesto e l'operazione richiesta sull'oggetto. La funzione verifica se l'utente o il ruolo sono inseriti in una delle liste di utenti abilitati all'accesso associate all'oggetto richiesto per la specifica operazione. In tal caso, la funzione restituisce *True*.
- *The checkCondition (operation, object) function*
  Riceve come input l'identificativo dell'oggetto e l'operazione che l'utente vuole compiere sull'oggetto. La funzione recupera la lista delle condizioni di accesso associate allo specifico oggetto e all'opportuna operazione. Successivamente, la funzione verifica che le condizioni definite attraverso la componente **Conditions** siano soddisfatte, ed in tal caso la funzione restituisce *Permit*. In caso contrario, restituisce altrimenti *Deny*.

Il modello è estremamente dinamico e semplice per quanto riguarda la gestione personalizzata delle politiche di sicurezza. Il paziente può facilmente indicare chi ha accesso ad un certo documento contenuto nel EHR, in quali periodi temporali e per quali scopi.
L'algoritmo è modulare, nel senso che è possibile usare solo un sottoinsieme di funzioni di controllo o anche invertire l'ordine delle funzioni. Per esempio, è possibile interscambiare le funzioni *checkinEmergency* e *checkNAble* per permettere al paziente di indicare soggetti che non possono accedere ad un certo documento anche nel caso di accesso in emergenza (*emergency mode*).

## 3. Pattern architetturale Model View Controller

La caratteristica di modularità dell'algoritmo che usa il modello MPP-ABAC, presentato nella sezione precedente, consente la sua implementazione secondo un pattern architetturale di tipo Model-View-Controller (MVC).

Prima di trattare l'analisi e la progettazione di tale modello in ambito tecnologico è importante descrivere brevemente il pattern MVC [7], rappresentato in Fig. 2.

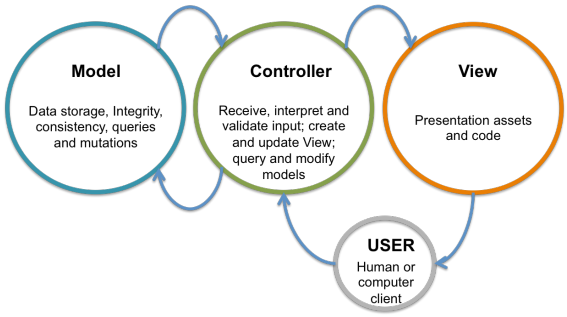


**Figura 2 – Rappresentazione grafica del pattern architetturale MVC**

MVC è un pattern architetturale molto diffuso nello sviluppo di sistemi software, in particolare nell'ambito della programmazione orientata agli oggetti, in grado di separare la logica di presentazione dei dati dalla logica di business.

Il pattern è basato sulla separazione dei compiti del software che interpretano tre ruoli principali:

- Il **Model** che fornisce i metodi per accedere ai dati utili all'applicazione;
- Il **View** che visualizza i dati contenuti nel model e si occupa dell'interazione con utenti ed agenti;
- Il **Controller** che riceve i comandi dell'utente (in genere attraverso il View) e li attua modificando lo stato delle altre due componenti.

Questo schema, fra l'altro, implica anche la tradizionale separazione fra la logica applicativa (chiamata spesso "logica di business"), a carico del Controller e del Model, e l'interfaccia utente a carico del View.

L'adozione del pattern MVC comporta tempi più lunghi in fase di progettazione, ma anche molti vantaggi, alcuni dei quali sono evidenziati di seguito:

- Un miglioramento nella comprensione del codice da parte di terzi sviluppatori che intendono aggiornare l'applicazione.
- Una semplificazione nella fase di aggiornamento del codice.
- Modifiche più rapide perché le parti del modello sono separate tra loro e questo permette un intervento puntuale sulla parte interessata dalle modifiche.
- Possibilità di sviluppare un'applicazione modulare.
- Software più flessibile, manutenibile e aggiornabile nel tempo.
- Costi di sviluppo ridotti, grazie alla separazione dei compiti e alla possibilità di operare su una singola componente indipendente.

Esistono molti framework tecnologici che utilizzano questo pattern architetturale, come ad esempio Struts. Nel seguito di questo lavoro sarà trattato tale framework, che risulta abilitante all'adozione del pattern MVC per l'implementazione software del modello di controllo degli accessi MPP-ABAC.

## 4. Apache Struts 2

Nella precedente sezione è stato brevemente descritto il pattern MVC. In questa sezione sarà illustrato il framework Struts 2 [8] che risulta abilitante all'adozione di tale pattern per l'implementazione del modello di controllo degli accessi MPP-ABAC. Apache Struts 2 è un framework estensibile per creare applicazioni web Java. Il framework è stato creato per garantire l'intero ciclo di sviluppo: dalla costruzione all'installazione su server, fino alla manutenzione dell'applicazione nel tempo. Inoltre, Struts 2 estende le Java Servlet, incoraggiando gli sviluppatori all'utilizzo del pattern Model-View-Controller. Struts 2 implementa le componenti del pattern nel seguente modo (come riportato in Fig. 3):

- *Model*: implementa la logica applicativa ed è costituita da un insieme di classi Java, tipicamente Java Bean;
- *View*: insieme di pagine JSP costruite mediante l'ausilio di particolari tag offerti da Struts 2;
- *Controller*: il controllo viene affidato ad un filtro (*FilterDispatcher*) che gestisce le classi Action ed eventuali classi Helper. Le classi Action rispecchiano una struttura flessibile e possono estendere o implementare elementi messi a servizio dal framework. In base alle richieste dell'utente, il controller decide quale Action eseguire per interagire con il modello. Queste informazioni sono elencate e gestibili dal file di configurazione *struts.xml* che si trova nel classpath. Sulla base del risultato restituito dalla Action eseguita, il controller decide a quale pagina JSP affidare la gestione della risposta.

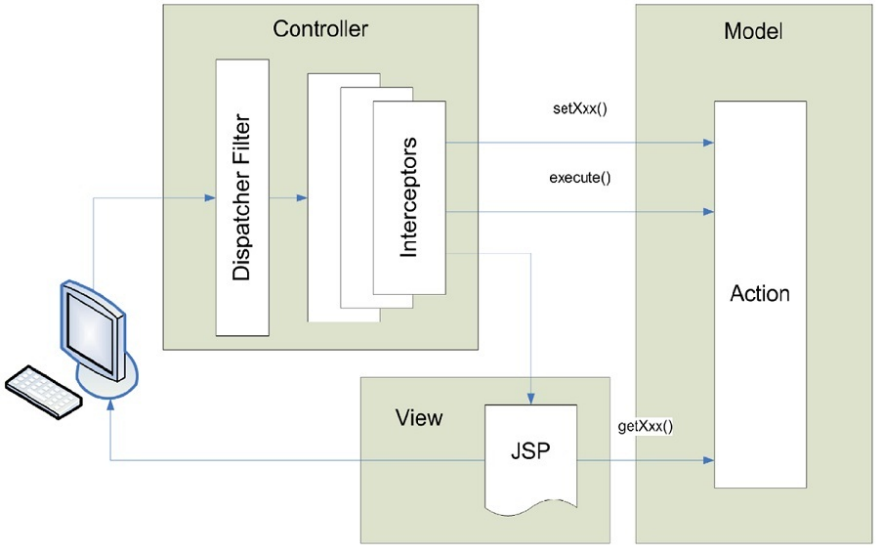


**Figura 3 – Pattern MVC in Struts 2**

Uno dei concetti innovativi introdotti da Struts 2 è rappresentato dagli *Interceptor*, classi stateless (che non mantengono uno stato tra invocazioni successive) che possono essere invocate automaticamente prima e dopo una Action. Per default, Struts 2 prevede un gruppo di interceptor che vengono richiamati prima di invocare qualsiasi Action. Il cosiddetto stack di default prevede ben 17 interceptor che lavorano dietro le quinte per offrire vari servizi. Oltre agli interceptor inclusi nello stack di default, Struts 2 permette inoltre di configurare ulteriori interceptor disponibili, utili soltanto in situazioni particolari.

### 4.1 Funzionamento del framework Struts 2

In Struts 2 ogni azione è associata ad un URL della forma http://localhost:8080/app/index.action. Il pattern *.action permette di identificare una richiesta che il *FilterDispatcher* prenderà in carico. Il *FilterDispatcher* ha il compito di intercettare tutte le richieste e indirizzarle verso l'azione ad esse associate. Esso, oltre ad avere il compito di intercettare e dirigere le azioni, ricopre anche il ruolo di configuratore in fase di startup dell'applicazione, cioè è suo compito:

- Inizializzare i contenuti statici del server, come i DOJO, gli script in JavaScript e tutti i file configurati dall'utente.
- Inizializzare il *ConfigurationManager* e gli *ActionMapper*, che servono per instradare le richieste verso le

giuste azioni.

- Creare i contesti di azione.
- Creare gli *ActionProxy*, che sono classi che contengono le configurazioni e le informazioni di contesto per eseguire le richieste e, dopo l'esecuzione, il risultato che è stato processato.
- Eseguire il cleanup degli oggetti *ActionContext* per assicurare il buon funzionamento dell'applicazione.

Come mostra il diagramma di sequenza in Fig. 4, ogni azione richiesta dall'utente viene intercettata dal *FilterDispatcher* che invoca la classe *ActionInvocation*. Questa classe gestisce l'ambiente di esecuzione e contiene lo stato della conversazione dell'azione in fase di elaborazione. Questa classe è il nucleo della classe *ActionProxy*. L'ambiente di esecuzione è composto da tre componenti: le azioni, gli interceptors ed i risultati. Uno dei compiti dell'*ActionInvocation* è di invocare i metodi delle azioni al momento opportuno. Un altro è creare le istanze delle azioni; al contrario della versione precedente, infatti, in Struts 2 ogni richiesta prevede la creazione di un nuovo oggetto azione. Questo può creare un lieve calo prestazionale dell'applicazione, ma concede il vantaggio di trattare questi oggetti come dei Plain Old Java Object (POJO). Dopo l'esecuzione della *ActionInvocation*, si entra nel core dell'applicazione, ossia si invoca l'azione e si esegue la logica di business prevista per la richiesta effettuata.

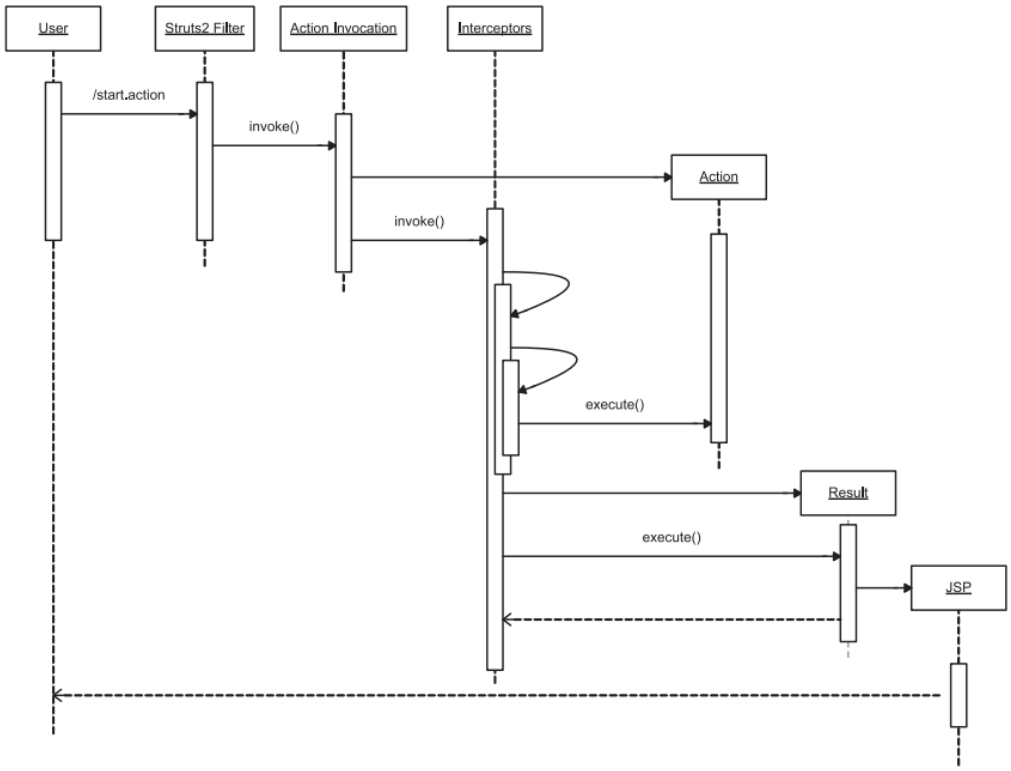


**Figura 4 – Ciclo di vita in Struts 2**

Come detto, in Struts 2 il core dell'applicazione è rappresentato dalle azioni, espresse da classi Java che soddisfano il requisito di possedere un metodo a firma vuota che restituisca una stringa o un oggetto di tipo *Result*. Per default questo metodo viene chiamato *execute()*. Quando il risultato è una stringa, il corrispondente oggetto *Result* viene ricercato nella mappatura e istanziato. In Struts 2 non è più necessario che un'azione estenda un'azione preconfigurata fornita dal framework. Ciononostante, Struts 2 fornisce allo sviluppatore alcune interfacce con cui implementare le azioni sviluppate: *Action* e *ActionSupport*. All'interno del framework, oltre al filtro e alle azioni, sono presenti anche gli interceptor, che concedono la possibilità di rendere più snello e leggibile il codice di core delle azioni lasciando agli interceptor le funzioni comuni a tutte le azioni, come per esempio la validazione dell'input.

Usando gli interceptors è possibile:

- fornire una logica in pre-configurazione, ossia prima della chiamata dell'azione;

- fornire una logica post-configurazione, cioè dopo l'esecuzione dell'azione;
- catturare le eccezioni in modo che possa essere eseguito codice alternativo o possa essere restituito un risultato diverso.

Struts 2 fornisce al suo interno una serie di interceptor già configurati e pronti per essere utilizzati, anche se è possibile scriverne di nuovi: è sufficiente creare un'interfaccia che estenda l'interfaccia *Interceptor*.

## 5. Scelte progettuali ed implementative

La presente sezione illustra le scelte intraprese per la realizzazione dell'algoritmo, sia in termini di progettazione che di implementazione.

### 5.1 Scelte progettuali

Nella progettazione del modello si è scelto di utilizzare data base (DB) separati per la memorizzazione delle diverse informazioni utilizzate dal modello di controllo degli accessi. Nello specifico si è scelto dividere le informazioni in 2 diversi DB:

- *DB Informazioni di controllo*: contiene informazioni di controllo sull'accesso ai documenti;
- *DB Utenti*: contiene le informazioni degli utenti del sistema.

Inoltre, i documenti a cui l'utente richiede l'accesso sono disponibili nel sistema di EHR.

La scelta di separare le diverse informazioni è stata adottata per limitare l'accesso alle sole informazioni di sicurezza indispensabili per valutare la legittimità dell'utente nella richiesta di accesso ad un particolare documento da parte del sistema, in ottemperanza al *Principle of least privilege*. La rappresentazione grafica del modello MPP-ABAC realizzato secondo il pattern architetturale MVC è riportata in Fig. 5.

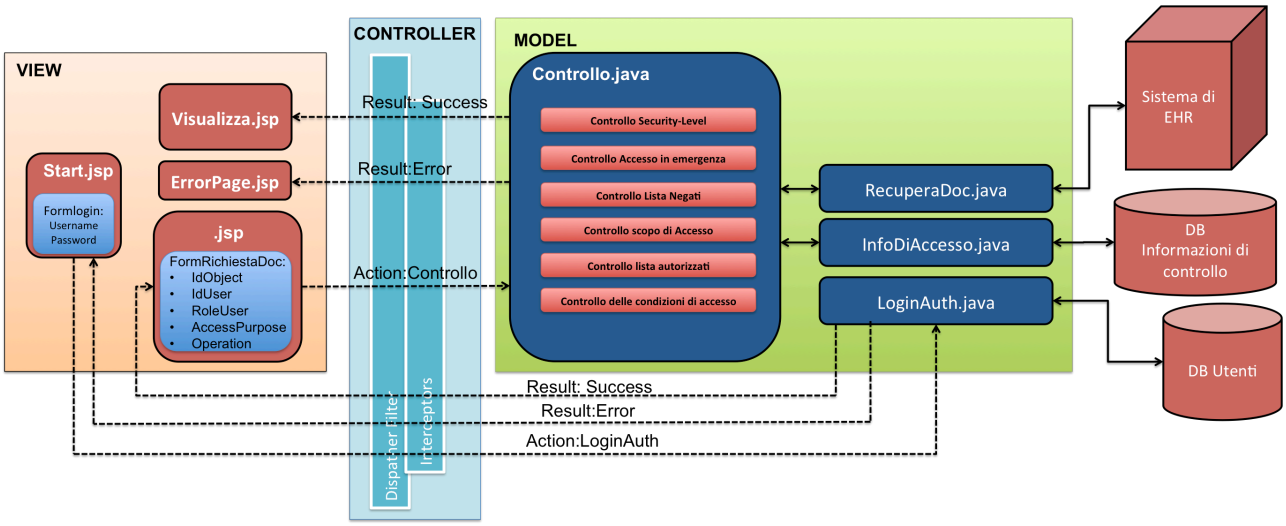


**Figura 5 – Rappresentazione grafica del modello implementato MPP-ABAC**

La progettazione del modello ha previsto la definizione di diverse funzionalità per il controllo di accesso alle risorse, che sono descritte di seguito:

- *Start.jsp* è una pagina di presentazione con form di login per l'accesso ai servizi di consultazione/gestione dei documenti presenti nel sistema di EHR. L'autenticazione dell'utente avviene mediante la Servlet *LoginAuth* (mappata come Action *LoginAuth*). Se l'autenticazione ha esito positivo, la Action restituisce *Success* ed il *FilterDispatcher* chiama la JSP *Welcome*, altrimenti il *FilterDispatcher* chiama la JSP *Login* che visualizza una messaggio di errore riferito alla credenziali inserite.

- *Welcome.jsp* è una pagina di redirect:
    - Nel caso l'utente loggato è un operatore sanitario viene presentata la pagina *ConsultaFSE.jsp*.
    - Nel caso l'utente loggato è un paziente viene presentata la pagina *GestionePrivacyFSE.jsp*.

- *ConsultaFSE.jsp* è una pagina di testing dell'algoritmo di AC. Infatti, molte delle informazioni richieste via form sono disponibili nel *DB Utenti*. In tale pagina è presente un form di compilazione per la richiesta di uno specifico documento. I campi richiesti nel form sono:
    - l'identificativo del documento;
    - l'identificativo dell'operatore che richiede il documento (deve corrispondere all'identificativo dell'utente loggato);
    - il ruolo che riveste l'utente nel contesto della richiesta del documento;
    - lo scopo di accesso al documento;
    - l'operazione che richiede di effettuare.

  Una volta compilato il form, il *FilterDispatcher* si occupa di chiamare l'Action *ControlloAccesso*. Se l'operazione richiesta dall'utente è consentita, la Action restituisce *Success* e il *FilterDispatcher* chiama *Visualizza.jsp*, in caso contrario viene chiamata *Errore.jsp*.

- *GestionePrivacyFSE.jsp* implementa una GUI per la creazione di nuove liste utenti e condizioni di accesso aggiuntive da associare ai documenti presenti nel sistema di EHR (nel contesto di questo rapporto tecnico non si è ritenuto interessante trattare le interfacce utente per la modifica della privacy sui documenti, con le operazioni di creazione liste, associazioni liste al documento, ecc).

- La Servlet *ControlloAccesso* rappresenta l'algoritmo descritto nella sezione 2, i cui controlli, effettuati in cascata, sono i seguenti:
    - *ControlloSecurityLevel* controlla a quale livello di sicurezza è associato al documento. Nel caso esso sia *secret* o *top-secret* vengono effettuati i controlli sul ruolo previsti dal sistema. In particolare:
        - *Top-secret*: prevede l'accesso al documento unicamente al medico che ha prodotto tale documento e al paziente;
        - *Secret*: prevede l'accesso anche al medico medicina generale (MMG), oltre ai ruoli previsti dal livello Top-secret.
    - *ControlloAccessoInEmergenza* è un metodo che verifica se l'utente che ha richiesto l'accesso in emergenza possa accedere con tale modalità e se il documento richiesto ha tra gli scopi di memorizzazione lo scopo di emergenza.
    - *ControlloListaNegati* è un metodo che controlla se l'utente che ha richiesto il documento è inserito nella Lista di Utenti non abilitati all'accesso a tale documento.
    - *ControlloScopoDiAccesso* è un metodo che verifica se lo scopo di accesso al documento è compatibile con uno degli scopi di memorizzazione del documento richiesto.
    - *ControlloListaAutorizzati* è un metodo che verifica se l'utente è inserito nella lista di utenti abilitati all'accesso al documento richiesto.
    - *ControlloDelleCondizioniDiAccesso* è un metodo che controlla se le condizioni di accesso al documento sono rispettate (come condizioni di localizzazione dell'utente o condizioni temporali di accesso).

- La Servlet *RecuperaDoc,* istanziata dalla classe *ControlloAccesso,* si occupa di effettuare la query ed il recupero del documento dal sistema di EHR.

La Servlet che permette il recupero delle informazioni relative ai permessi di accesso ai documenti è *RecuperoInfoDiAccesso*. Questa Servlet viene richiamata da tutte le Servlet di controllo che hanno la necessità di ottenere tali informazioni.

## 5.2 Dettagli implementativi

Il progetto sviluppato ha previsto la predisposizione di alcuni file di configurazione indispensabili per il funzionamento del framework Struts 2. In questa sezione sono riportati alcuni dei listati più interessanti da un punto di vista tecnologico.

- *web.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <filter>
    <filter-name>struts</filter-name>
    <filter-class>
      org.apache.struts2.dispatcher.FilterDispatcher
    </filter-class>
  </filter>
  <filter-mapping>
    <filter-name>struts</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <welcome-file-list>
    <welcome-file>Start.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

**Listato 5.1 – Frammento web.xml**

Il codice riportato nel listato 5.1 mostra la creazione del filtro standard di Struts 2 *FilterDispatcher* che si fa carico di instradare le richieste provenienti dall'utente.

Ogni richiesta intercettata dal Controller viene inoltrata ad una Action, una classe Java che implementa l'interfaccia com.opensymphony.xwork2.Action. Il file di configurazione che consente al controller di inoltrare in modo corretto le richieste e le risposte delle diverse Action è struts.xml.

- *struts.xml*

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
      <constant name="struts.devMode" value="true" />
      <package name="MPP-ABAC" namespace="/" extends="struts-default">
            <action name="LoginAuth”>
                  <result name="success">/Welcome.jsp<result>
                  <result name="Error">/Start.jsp<result>
            </action>
            <action name="ControlloAccesso”>
                  <result name="success">/Visualizza.jsp<result>
                  <result name="Error">/ErrorePage.jsp<result>
            </action>
      </package>
<struts>
```

**Listato 5.2 – Frammento struts.xml**

La pagina JSP *AccessoAlFSE*, che consente all'operatore sanitario di inserire i campi relativi alla richiesta di accesso al documento, è molto semplice ma significativa in quanto è la pagina JSP che innesca il controllo dei parametri di accesso secondo l'algoritmo implementato dalla Servlet *ControlloAccesso*. A seguire viene presentato il listato di tale pagina.

- *AccessoAlFSE.jsp*

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head>
<title>Richiesta Documento</title>
</head>

<body>
<h2>Informazioni relative al documento richiesto </h2>
<s:form action="ControlloAccesso.action" method="post">
<s:textfield name="idDoc" key="label.idDoc" size="20" />
<s:textfield name="idOper" key="label.idOper" size="20" />
<s:textfield name="OpRuol" key="label.OpRuol" size="20" />
<s:textfield name="Scop" key="label.Scop" size="20" />
<s:textfield name="Oper" key="label.Oper" size="20" />
<s:submit method="execute" key="label.ControlloAccesso" align="center" />
</s:form>
</body>
</html>
```

**Listato 5.3 – Frammento AccessoAlFSE.jsp**

L'ultimo listato riguarda la Servlet *ControlloAccesso*, che è sicuramente la Servlet più importante in quanto implementa in sé l'algoritmo di controllo dell'accesso alla risorsa.

- *ControlloAccesso.java*

```
public class LoginAuthAction extends ActionSupport {

    private String idDoc;
    private String idOper;
    private String OpRuol;
    private String Scop;
    private String Oper;
    private Boolean error;

    public String execute() {
      InfoDiAccesso document = new InfoDiAccesso(idDoc);


    if (document.levelsecurity == "TopSecret"){
        if(checkAccessTopSecret(idOper,document)){
          return Success;
      }
      else return Error;
    }
    if(document.levelsecurity == "Secret"){
      if(checkAccessSecret(idOper,document)){
          return Success;
      }
      else return Error;
    }
    if(document.levelsecurity == "Normal"){
      if(scop==Emergency){
            if (checkinEmergency(document,OpRuol)){
                return Success;
            }
            else return Error;
      }
      if(checkNAble(idOper,OpRuol,document,Oper)){
          return Error;
      }
      if(!(checkPurpose(Scop, document, Oper)){
```

```
          return Error;
      }

      if(!checkList(idOper, OpRuol, document, Oper){
          return Error;
      }
      if(!checkCondition(Oper, document, document.condition){
          return Error;
      }
      return Success;
    }

    }

    public Boolean checkTopSecret(String idOper, InfoDiAccesso document) {
      if(idOper == document.writer || idOper == document.patient){
      return true;
      }
      else return false;
    }

    public Boolean checkSecret(String idOper, InfoDiAccesso document) {
      if(idOper == document.writer || idOper == document.patient || idOper ==
document.patient.MMG){
      return true;
      }
      else return false;
    }

    public Boolean checkinEmergency (InfoDiAccesso document,String OpRuol){
      if(document.emergencymode && accedeinEmergencymode(OpRuol)){
          return true;
      }
      else return false;
    }

    public Boolean checkNAble(String idOper, InfoDiAccesso document, String
Oper){

          for(j=0;j<document.Oper.listaNable.lenght;j++){
            if(document.Oper.listaNable[j]==idOper){
                return false;
            }
          }
      return true;
    }

    public Boolean checkPurpose(String Scop, InfoDiAccesso document, String
Oper)){

          for(j=0;j<document.Oper.scope.lenght;j++){
            if(document.Oper.scope[j]==Scop){
                return true;
            }
          }
      return false;
    }

    public Boolean checkList(String idOper, String OpRuol, InfoDiAccesso
document, String Oper)){

          for(j=0;j<document.Oper.list.lenght;j++){
            for(i=0;i<document.Oper.list[j].lenght;i++){
                if(document.Oper.Oper.list[j][i]==idOper){
                return true;
                }
            }
          }
          return false;
    }
```

```
        public Boolean checkCondition(string Oper, InfoDiAccesso document){
              return true;

        }

    }
```

**Listato 5.4 – Frammento ControlloAccesso.java**

## 6. Conclusioni

Partendo dall'analisi del modello MPP-ABAC, in questo lavoro si è dimostrata la fattibilità implementativa del modello stesso progettando ed implementando un algoritmo basato su di esso. In particolare, si è scelto di utilizzare un pattern architetturale MVC data la natura modulare e flessibile offerta e richiesta, al tempo stesso, dal modello MPP-ABAC. Il framework abilitante adottato è stato Apache Struts 2. Tale framework risulta particolarmente interessante da un punto di vista tecnologico perché usa un controller molto flessibile e personalizzabile che ha consentito una implementazione manutenibile e facilmente aggiornabile.

La realizzazione dell'algoritmo di AC apre una nuova fase di sperimentazione, testing e misurazione di performace dell'algoritmo, che sarà svolta nell'ambito di attività future.

## Riferimenti bibliografici


[1] M. Sicuranza, A. Esposito "An Access Control Model for Easy Management of Patient Privacy in EHR Systems", in ICITST 2013: Proceedings of the 8th International Conference for Internet Technology and Secured Transactions, IEEE, 2013.

[2] I. Iakovidis, "Towards Personal Health Record: Current Situation, Obstacles and Trends in Implementation of Electronic Healthcare Record in Europe", International Journal of Medical Informatics, 52(1–3), pp. 105–115, 1998.

[3] D. F. Ferraiolo, J. Cugini, and D. R. Kuhn, "Role-Based Access Control (RBAC): Features and Motivations", in Proceedings of the 11th Annual Computer Security Application Conference, New Orleans, LA, December 11–15 1995, pp. 241–248.

[4] K. Yoonjeong, S. Eunjee, "Privacy-Aware Role Based Access Control Model: Revisited for Multi-Policy Conflict Detection," in Proceedings of the International Conference of Information Science and Applications (ICISA), vol. 1, no. 7, pp. 21-23, April 2010.

[5] E. Bertino, P. A. Bonatti, and E. Ferrari, "TRBAC: A Temporal Role-Based Access Control Model", in RBAC '00: Proceedings of the fifth ACM Workshop on Role-Based Access Control, pp. 21.30, New York, NY, USA, 2000, ACM Press.

[6] A. Ferreira, D. Chadwick, P. Farinha, R. Correia, Z. Gansen, R. Chilro, L. Antunes, "How to Securely Break into RBAC: The BTG-RBAC Model", in ACSAC '09: Proceedings of Annual Computer Security Applications Conference, pp. 23-31, 7-11 Dec. 2009.

[7] Model View Controller Pattern, http://en.wikibooks.org/wiki/Computer_Science_Design_Patterns/Model–view–controller [Online; accessed – January-2014].

[8] Apache Software Foundation, Apache Struts 2, http:// struts.apache.org/2.2.1/index.html, 2010. [Online; accessed – January-2014].