UNIVERSITÀ DEGLI STUDI DI PISA

Corso di laurea in ingegneria informatica

Reingegnerizzazione di ruoli e permessi in CHKB, una base di conoscenza web per la storia dell'informatica.

Candidato:

Edoardo Sassu

I Relatore:

Giuseppe Lettieri

II Relatore:

Giovanni Cignoni

Indice

1 Introduzione

CHKB (Computer History Knowledge Base) è una base di conoscenza pensata per raccogliere informazioni sugli strumenti per il calcolo da collezioni pubbliche e private. Lo scopo dell'interfaccia web è quello di:

- Fornire a musei e proprietari di collezioni private un'interfaccia semplice per catalogare i pezzi in loro possesso. Il sistema di catalogazione fornito da CHKB struttura le informazioni in:
  - **Schede reperto**: Raccolgono informazioni specifiche del reperto come documentazione, foto ecc.
  - **Schede informative**: Raccolgono informazioni correlate ai reperti. Esse possono riguardare il reperto stesso, una sua componente, un personaggio storico che ha contribuito al suo sviluppo ecc.

  Strutturare i dati in questo modo permette di catalogare i pezzi in maniera più efficiente, da la possibilità di creare correlazioni tra pezzi fisici e schede informative ed evita che vengano replicate informazioni già presenti nella base di conoscenza.

- Fornire al pubblico un modo alternativo per consultare le collezioni, mostrandole non solo come liste di pezzi, ma mostrando, per ogni pezzo, le varie correlazioni che esso ha con altri pezzi, personaggi storici, aziende costruttrici ecc. Cerca quindi di fornire all'utente un modo interattivo di navigare le informazioni presenti nella base di conoscenza.

L'applicazione ha quindi lo scopo di fornire un'interfaccia amichevole che renda usufruibili i contenuti agli utenti che desiderano consultarli, e fornisca un interfaccia di gestione dei contenuti a collezionisti o musei. È quindi necessario un meccanismo che permetta l'accesso alle varie parti dell'applicazione, in base al tipo di utente che la richiede.

Lo scopo di questa tesi è quello di reingegnerizzare il meccanismo di accesso alle varie parti dell'applicazione al fine di evitare accessi non autorizzati e, al tempo stesso, fornire un API che permetta di semplificare il controllo dei permessi utente.

Di seguito verranno analizzati solo i punti relativi al meccanismo di controllo dei permessi. Per conoscere i dettagli sulla struttura dell'applicazione si rimanda alla tesi di A. Biaggini[1], mentre per la struttura dei percorsi si rimanda alla relazione del tirocinio di D. Aimini[2].

2 Analisi del problema

In CHKB **non è presente un meccanismo che renda omogeneo il controllo dei permessi posseduti dall'utente** nelle varie parti dell'applicazione. Questo obbliga, ogni volta che deve essere effettuata un operazione, a ripetere il codice che verifica il la fattibilità di tale operazione da parte dell'utente. È quindi presente del **codice ridondante**, che a sua volta porta ad una scarsa manutenibilità e riusabilità dello stesso.

Un ulteriore problema è dovuto ad una **distinzione non netta tra ruoli e permessi**. Un ruolo dovrebbe fornire all'utente una serie di permessi su una parte dell'applicazione. I controlli di fattibilità delle operazioni andrebbero fati sui permessi che un utente possiede su una risorsa. Nell'applicazione questo non è sempre vero. I controlli vengono effettuati, a volte basandosi sui permessi, altre basandosi sui ruoli.

Non essendoci un modo univoco per effettuare uno stesso controllo, ed essendo possibile effettuare un controllo basandosi sui ruoli posseduti dall'utente, è facile che vengano consentiti accessi non autorizzati a varie parti dell'applicazione.

A causa dei problemi appena citati e delle varie modifiche che l'applicazione ha subito, sono stati riscontrati molti errori relativi alla visualizzazione ed alla manipolazione dei contenuti.

3 Soluzione

La soluzione che è stata adottata per risolvere i problemi appena discussi segue i seguenti punti:

- **Separazione netta tra ruoli e permessi** ad associazione di un insieme di permessi al ruolo corrispondente.
- Implementazione di una **meccanismo univoco e semplice per il controllo dei permessi** posseduti dall'utente.
- **Accorpamento di tutti i ruoli sull'utente**.

Accorpando i ruoli sull'utente si va a creare una meccanismo di controllo degli accessi basato su **capability list**. In questo modo è possibile identificare i permessi posseduti da un utente sulle varie parti dell'applicazione e valutare la fattibilità di un'operazione, confrontando i permessi da esso posseduti, con quelli necessari per svolgere tale operazione.
Da notare che, **per ogni utente**, **sono noti i ruoli da esso posseduti e non i permessi**. Si rende quindi **necessario un meccanismo che, data una risorsa ed un utente, sia in grado di ricavare la lista di permessi posseduti dall'utente su quella risorsa a partire dai suoi ruoli**.

4 Analisi dei ruoli necessari e delle risorse manipolabili

Come accennato nell'introduzione, l'applicazione deve consentire:

- La visione agli ospiti dei contenuti pubblicamente visibili.
- La visione e la manipolazione dei contenuti a musei e collezionisti.

I contenuti si possono classificare in:

- Contenuti relativi alle collezioni
- Contenuti relativi alle schede informative
- Contenuti relativi ai percorsi
- Contenuti relativi agli utenti

4.1 Contenuti relativi alle collezioni

Rientrano in questa categoria di contenuti:

- Collezioni
- Pezzi fisici
- Contenuti digitali specifici

Le collezioni vengono create da un utente **Amministratore**, che ha anche il computo di assegnare un **Responsabile** alla collezione. Il responsabile modifica le informazioni generiche della collezione, definisce il sistema di catalogazione e di collocazione e sceglie i **Catalogatori** della collezione tra gli **utenti registrati**. I catalogatori hanno il compito di creare e gestire i pezzi fisici ed i contenuti digitali specifici associati alla collezione della quale sono catalogatori.

I ruoli che entrano in gioco nella manipolazione delle collezioni sono:

- **Amministratore**: utente che crea la collezione ed assegna il responsabile.
- **Responsabile di collezione**: utente registrato a cui è stato dato il compito di gestire una collezione.
- **Catalogatore di collezione**: utente registrato a cui è stato dato il compito di catalogare i pezzi di una collezione.
- **Utente Registrato**

4.2 Contenuti relativi alle schede informative

Rientrano in questa categoria di contenuti:

- Schede informative.
- Versioni di scheda informativa.

- Contenuti digitali generici.

Ad un utente registrato può essere assegnato il ruolo di **Revisore**, che ha il compito di creare e validare le schede informative. Tutti gli utenti possono proporre schede informative. Tali proposte devono essere approvate dal Revisore. Il revisore può assegnare ad un utente registrato il ruolo di **Editore**, che ha il compito di compilare una nuova versione della scheda informativa e sottometterla a revisione. L'editore può creare contenuti digitali generici, impostarne la visibilità ed associarli alla scheda informativa.

I ruoli che entrano in gioco nella manipolazione delle schede informative sono:

- **Revisore delle schede**: Utente registrato che gestisce le schede informative. Ha il compito di assegnare le schede in redazione e revisionarle.
- **Editore delle schede**: Utente registrato che redige le schede informative e le sottopone a revisione.
- **Utente Registrato**: Può proporre le schede informative e diventare editore.
- **Utente ospite**: Può proporre le schede informative.

4.3 Contenuti relativi ai percorsi

Rientrano in questa categoria di contenuti:

- Percorsi.
- Elementi di percorso.

I percorsi, sono delle visite tematiche virtuali. Esse sono composte da pezzi fisici appartenenti alle collezioni e schede informative.

I percorsi sono interamente creati e gestiti da un utente che ricopre il ruolo di **Navigatore**. Il navigatore crea un nuovo percorso e i relativi elementi di percorso. Gli elementi di percorso non sono altro che dei collegamenti a pezzi fisici o schede informative presenti nella base di conoscenza.

I ruoli che entrano in gioco nella manipolazione dei percorsi sono:

- **Navigatore**: Utente registrato che gestisce i percorsi.

4.4 Contenuti relativi agli utenti

Per quanto riguarda gli utenti entrano in gioco i ruoli:

- **Amministratore**: Utente in grado di visualizzare e manipolare i ruoli posseduti da tutti gli utenti presenti nell'applicazione.
- **Utente Registrato**: Utente in grado di modificare le informazioni relative a se stesso come, nome ,cognome, password ecc.
- **Utente non Registrato**: Utente in grado di effettuare richieste di registrazione.

5 Logica di conversione Ruolo/Permessi

In questo capitolo viene analizzata la logica con cui i ruoli vengono tradotti in liste di permessi per ogni risorsa manipolabile da un utente. La logica usata per le conversioni dei ruoli in liste di permessi cambia a seconda della parte dell'applicazione che si va a considerare. Per questo motivo la logica di traduzione viene analizzata in maniera separata per ognuna delle quattro categorie di contenuti definite nel capitolo precedente.

5.1 Contenuti relativi alle collezioni

Appartengono a questa categoria collezioni, pezzi fisici e contenuti digitali specifici.

5.1.1 Collezioni

La tabella di conversione da ruoli a permessi per le collezioni è raffigurate sotto:

Permessi possibili	Utente non registrato	Utente registrato	Catalogatore della colazione	Responsabile della collezione	Amministratore
VIEW	*	*	*	*	*
CREATE					*
ASSIGN_CATALOGUER				*	*
ASSIGN_MANAGER					*
EDIT			*		
EDIT_INFO				*	
DEFINE_CONVENTION				*	
DEFINE_SHELF_CONVENTION				*	

Il significato del permessi di EDIT viene approfondito nel sottoparagrafo successivo.

5.1.2 Pezzi fisici

Le collezioni sono un aggregazione di pezzi fisici. Manipolare un pezzo fisico significa quindi modificare la collezione che lo contiene. Per questo motivo, gli utenti che devono manipolare pezzi fisici, devono possedere il permesso di modifica sulla collezione contenente il pezzo.
La stessa considerazione può essere fatta per il permesso di visibilità dei pezzi fisici. Per poter vedere un pezzo è necessario poter vedere la collezione che lo contiene.
Inoltre un pezzo fisico ha diversi gradi di riservatezza che influenzano il permesso di visibilità su quel pezzo e sulla sua collocazione.
Un pezzo fisico può essere:

- Privato: pezzo e collocazione visibile sono ai catalogatori e responsabili della collezione
- Pubblico : pezzo visibile a tutti gli utenti
    - Esposto: collocazione visibile a tutti gli utenti
    - Non Esposto: collocazione visibile sono agli utenti registrati

I possibili gradi di riservatezza sono riportati nell'immagine 1.

Illustrazione 1:Visibilità dei pezzi fisici

La tabella di conversione da ruoli a permessi per i pezzi fisici è la seguente:

Permessi possibili	Utente non registrato	Utente registrato	Catalogatore della colazione	Responsabile della collezione
VIEW	*(per pubblici)	*(per pubblici)	*	*
VIEW_COLLOCATION	*(per pubblici esposti)	*(per pubblici)	*	*
CREATE			*	
EDIT			*	
DELETE			*	

In questo caso il permesso di EDIT, oltre a consentire la modifica delle informazioni sul pezzo, permette di manipolare i contenuti digitali specifici associati a quel pezzo.

5.1.3 Contenuti digitali specifici

I contenuti digitali specifici sono associati ai pezzi fisici, per cui valgono le stesse considerazioni fatte in precedenza. Manipolare un contenuto digitale significa modificare il pezzo fisico a cui il contenuto è associato, è quindi necessario il permesso di EDIT sul pezzo fisico.
Anche per la visibilità del contenuto digitale è necessario avere la visibilità del pezzo fisico.
Se il pezzo fisico a cui il contenuto è associato è pubblico, allora il in contenuto ha un ulteriore grado di riservatezza:

- Visibilità pubblica: contenuto visibile e scaricabile da tutti gli utenti
- Visibilità ristretta: contenuto visibile e scaricabile solo dagli utenti registrati

Se il pezzo fisico a cui il contenuto è associato è privato allora il contenuto digitale ha la stessa visibilità del pezzo fisico.
La figura seguente mostra i gradi di riservatezza sui contenuti digitali in relazione ai pezzi fisici.

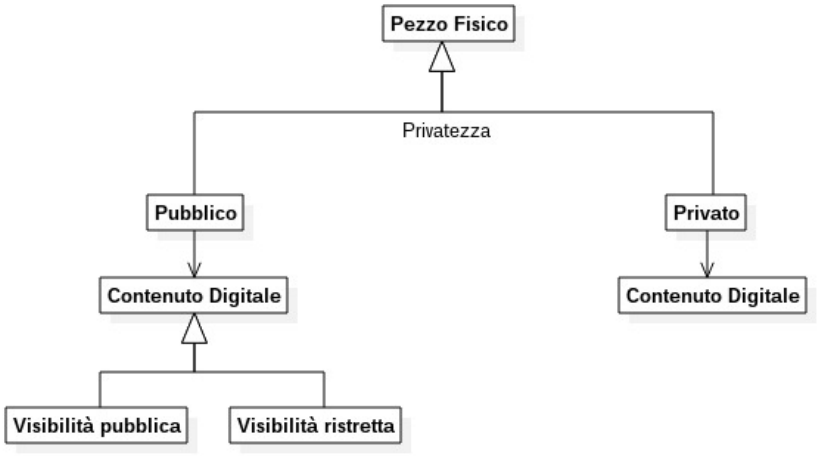

Illustrazione 2:Visibilità dei contenuti digitali specifici

La tabella di conversione da ruolo a permessi per i contenuti digitali è la seguente:

Permessi possibili	Utente non registrato	Utente registrato	Catalogatore della colazione	Responsabile della collezione
VIEW	*(pezzo pubblico, contenuto visibile)	*(pezzo pubblico)	*	*
DOWNLOAD	*(pezzo pubblico, contenuto visibile)	*(pezzo pubblico)	*	*
CREATE			*	
EDIT			*	
DELETE			*	

5.2 Contenuti relativi alle schede informative

La logica per la conversione dei permessi delle schede informative è differente da quella delle collezioni.
Una scheda informativa può essere vista come un'aggregazione di sue versioni. Alle schede informative possono essere associati contenuti digitali generici.

5.2.1 Scheda informativa

Quando una scheda informativa viene creata, se ne crea anche una nuova versione. Un utente in possesso del ruolo di revisore assegnerà il compito di redigere la scheda ad un utente. Tale utente, che acquisisce il ruolo di editore, compila la scheda informativa e la manda in revisione. Il revisore può decidere se approvare una versione o respingerla e rimandarla in redazione.
I permessi sulla scheda informativa dipendono dallo stato in cui si trovano le sue versioni.
Modificare una scheda informativa corrisponde a creare una nuova versione della scheda che andrà redatta e mandata in revisione.
Vedere una scheda informativa significa vedere l'ultima versione approvata, per cui una scheda informativa è visibile sene esiste almeno una versione approvata.
Di seguito viene riportata la tabella di conversione ruoli/permessi per le schede informative.

Permessi possibili	Utente non registrato	Utente registrato	Editore(dell'ultima versione)	Revisore

VIEW	*(se esiste una versione approvata)	*(se esiste una versione approvata)	*	*
CREATE				*
EDIT			*	
DELETE				*
ACCEPT				*
PROPOSE	*	*	*	*

5.2.2 Versione di scheda informativa

Manipolare una versione di scheda informativa significa modificare la scheda informativa a cui appartiene. Per manipolare un versione è quindi necessario il permesso di EDIT sulla scheda informativa. Tale permesso viene acquisito da un utente quando il revisore gli assegna una scheda da redigere e quindi acquisisce il ruolo di editore della scheda.

La visibilità della versione dipende dallo stato della versione:

- Approvata: visibile a tutti gli utenti
- Non approvata: visibile a revisore e editore

Di seguito viene riportata la tabella di conversione ruolo/permessi per versioni di scheda informativa.

Permessi possibili	**Utente non registrato**	**Utente registrato**	**Editore**	**Revisore**
VIEW	*(se approvata)	*(se approvata)	*	*
CREATE				*
EDIT			*	
DELETE				*
ASSIGN				*
SUBMIT			*	
REVIEW				*

5.2.3 Contenuto digitale generico

Nell'attuale versione dell'applicazione, un contenuto digitale generico può essere associato a una o più schede informative. Manipolare un contenuto significa quindi modificare tutte le schede informative a cui il contenuto è associato. Una gestione dei contenuti di questo tipo introduce alcuni problemi:

- Non si hanno informazioni sull'editore che ha aggiunto il contenuto
- Non si hanno informazioni sulle versioni nelle quali un contenuto viene aggiunto o modificato
- Per essere coerenti con la logica di traduzione dei permessi adottata sinora, un contenuto che appartiene a più schede informative dovrebbe poter essere manipolato da un editore solo se esso è editore di tutte le schede informative a cui il contenuto è associato. Questo è vero solo raramente e non consentirebbe a un editore di manipolare contenuti associati a diverse schede informative.

L'ideale sarebbe associare i contenuti digitale non alle schede informative, ma alle versioni in modo da avere informazioni su quando il contenuto è stato aggiunto e su chi l'ha aggiunto o modificato di versione in versione. Un editore dovrebbe essere in grado solo di creare e associare un contenuto alla versione di cui è editore e dovrebbe poter modificare i contenuti solo se è editore di tutte le schede a cui il contenuto è associato. Solo il revisore delle schede ha i permessi per eliminare i contenuti generici. Notare che eliminando un contenuto lo si elimina da tutte le schede a cui esso era associato.
Per problemi di tempo non è stato possibile spostare i contenuti digitali da schede informative a versioni di scheda informativa. Per questo motivo la soluzione adottata sui contenuti digitali generici è da considerarsi temporanea e da modificare in sviluppi futuri.
Per comodità viene lasciato all'editore il permesso di modifica del contenuto anche se il contenuto è associato a schede di cui l'utente non è editore. Il revisore ha invece tutti i permessi ed è l'unico a poter eliminare i contenuti digitali in accordo con quando appena detto.
Per quanto riguarda la visibilità dei contenuti, essi hanno un attributo che ne definisce la visibilità e può assumere i seguenti valori:

- Visibilità pubblica: Contenuto visibile a tutti gli utenti
- Visibilità ristretta: Contenuto visibile solo agli utenti regiatrati

La figura seguente mostra i gradi di riservatezza sui contenuti digitali generici.

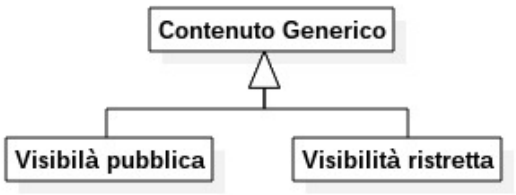

Illustrazione 3:Visibilità contenuto generico

Di seguito viene riportata la tabella di conversione ruolo/permessi per i contenuti digitali generici.

Permessi possibili	Utente non registrato	Utente registrato	Editore	Revisore
VIEW	*(se pubblico)	*	*	*
DOWNLOAD	*(se pubblico)	*	*	*
CREATE			*	*
EDIT			*	*
DELETE				*

5.3 Contenuti relativi ai percorsi

5.3.1 Percorsi

I percorsi hanno una attributo che ne determina la visibilità. Questo attributo in precedenza apparteneva agli elementi di percorso ma si è ritenuto più sensato dare la possibilità si ridurre la visibilità all'intero percorso e non ai singoli elementi. Il permesso di visibilità sul percorso è utile

per nascondere un percorso agli utenti quando questo è in fase di creazione/modifica. La visibilità si un percorso può essere:

- Pubblica: visibile a tutti gli utenti
- Privata: visibile solo agli utenti navigatori

Di seguito viene riportata la tabella di conversione ruoli/permessi per i percorsi.

Permessi possibili	Utente non registrato	Utente registrato	Navigatore
VIEW	*(se pubblico)	*(se pubblico)	*
CREATE			*
EDIT			*(se creatore del percorso)
EDIT_INFO			*(se creatore del percorso)
DELETE			*(se creatore del percorso)

5.3.2 Elementi di percorso

Un percorso può essere visto come un'insieme di elementi. Quindi manipolare un elemento significa modificare il percorso a cui appartiene. Per questo, per manipolare un elemento è necessario avere il permesso di modifica sul percorso che lo contiene.
Per quanto riguarda la visibilità vale lo stesso discorso, per poter vedere un elemento è necessario poter vedere il percorso a cui appartiene.
Come detto in precedenza gli elementi di percorso avevano un attributo di visibilità che è stato rimosso.
Gli elementi di percorso non sono altro che collegamenti a pezzi fisici o a schede informative che vengono mesi insieme in un percorso per dare all'utente un modo alternativo di navigare la base di conoscenza. La visibilità di un elemento di percorso è quindi determinata dalla visibilità dell'oggetto puntato dall'elemento di percorso, per cui se l'oggetto puntato è visibile all'utente, allora lo è anche l'elemento di percorso. Da notare che negli elementi di percorso vengono mostrati anche i contenuti digitali associati all'elemento puntato. Anche questi saranno visibili solo se l'utente è in possesso dei permessi necessari sui contenuti digitali.
Di seguito viene riportata la tabella di conversione ruolo/permessi per gli elementi di percorso.

Permessi possibili	**Utente non registrato**	**Utente registrato**	**Navigatore**
VIEW	*(se percorso ed elemento puntato sono visibili)	*(se percorso ed elemento puntato sono visibili)	*
CREATE			*(se ha i permessi di modifica sul percorso)
EDIT			*(se ha i permessi di modifica sul percorso)
DELETE			*(se ha i permessi di modifica sul percorso)
MOVE			*(se ha i permessi di modifica sul percorso)

5.4 Gestione utenti

La gestione utenti riguarda la possibilità di assegnare i ruoli agli utenti e poter vedere l'elenco degli utenti registrati.
Per acquisire tali permessi su tutti gli utenti è necessario ricoprire il ruolo di amministratore.
Di seguito viene riportata la tabella di conversione ruoli/permessi per l'oggetto utente.

Permessi possibili	Utente non registrato	Utente registrato	Amministratore
VIEW			*
ACCEPT			*
MANAGE_ROLES			*
SIGN_UP	*		
EDIT_INFO		*(su se stesso)	

6 Progettazione della soluzione

Per poter realizzare il meccanismo di traduzione è stato necessario implementare due gerarchie di classi:

- Gerarchia dei ruoli
- Gerarchia dei permessi

La gerarchia dei ruoli effettua la traduzione da ruolo a lista di permessi per ogni risorsa manipolabile dall'utente.

La gerarchia dei permessi deve valutare se un utente è in grado di effettuare un'operazione su una risorsa. Per fare ciò ricava dai ruoli posseduti dall'utente, la corrispondente lista di permessi e la confronta con quelli necessari per svolgere l'operazione richiesta.

Al fine di poter utilizzare il meccanismo di traduzione, si sono rese necessarie delle modifiche alle seguenti classi:

- Classe utente: Nella quale vengono raccolti tutti i ruoli posseduti da quell'utente
- Classi delle risorse manipolabili: A queste classi sono stati aggiunti tanti metodi quante sono le operazioni possibili su quella risorsa. Lo scopo di questi metodi è quello di fornire al meccanismo di controllo, la lista di permessi necessari per svolgere le varie operazioni possibili su quella risorsa.

L' Immagine 4 mostra l'architettura della soluzione progettata.

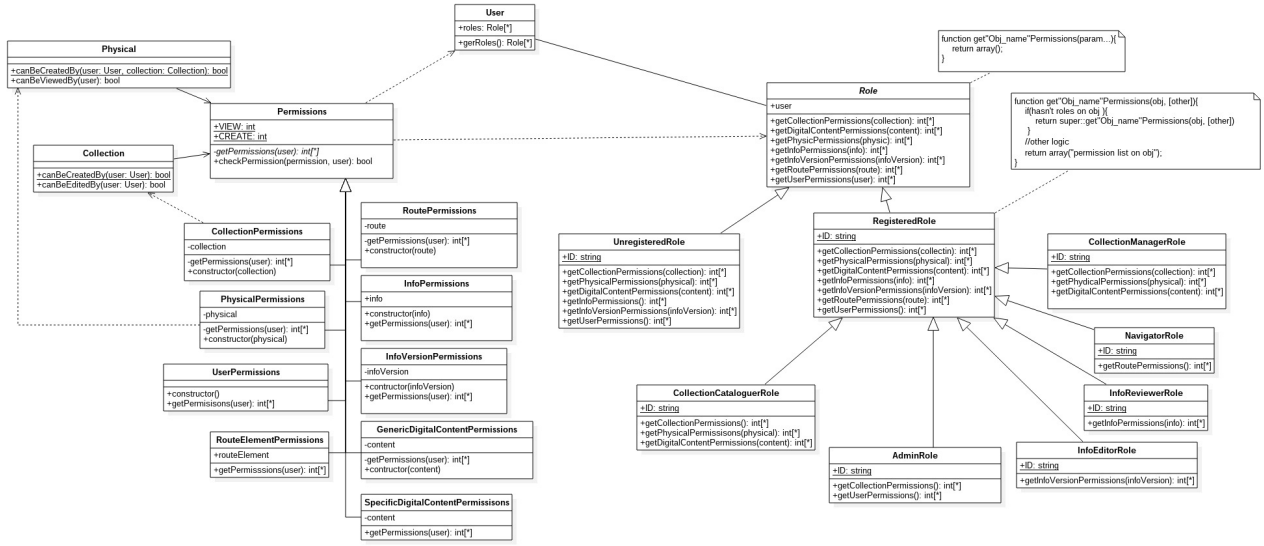

Illustrazione 4

L'immagine 5 mostra una visione generale del funzionamento del meccanismo di controllo.

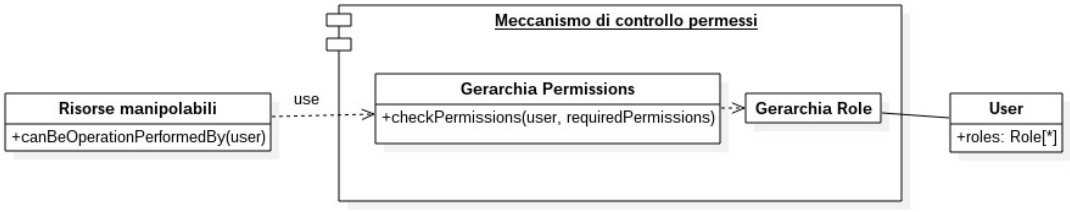

Illustrazione 5

Di seguito vengono analizzate le gerarchie che realizzano il meccanismo di controllo dei permessi.

6.1 Gerarchia ruoli

Questa gerarchia è formata da tante classi quanti sono i ruoli assegnabili ad un utente. **Ogni classe della gerarchia effettua, per ogni risorsa manipolabile dall'utente, la conversione dal ruolo alla corrispondente lista di permessi**.

Un requisito fondamentale dell'applicazione è quello di poter aggiungere ruoli in modo semplice. Per questo motivo la gerarchia è stata progettata seguendo il design pattern Visitor. La struttura adottata consente di aggiungere ruoli semplicemente aggiungendo una classe alla gerarchia senza dover modificare nessuna delle altre classi esistenti.

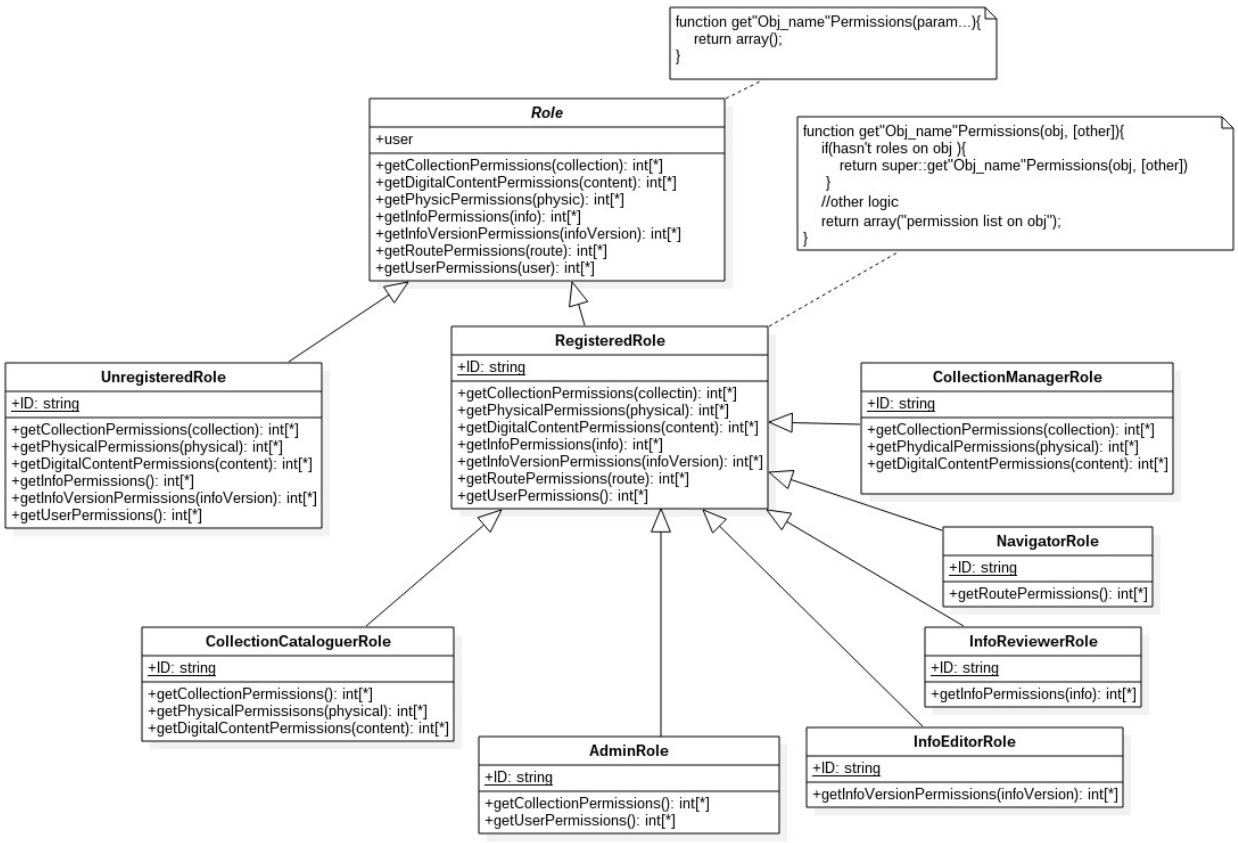

Illustrazione 6: Gerarchia ruoli

La classe Role descrive l'interfaccia che tutte le classi figlie implementano. Essa definisce tutti i possibili metodi di traduzione, restituendo di default una lista di permessi vuota.
Le classi figlie ridefiniranno il comportamento dei metodi per le quali forniscono permessi, lasciando inalterati i metodi non di competenza di quel particolare ruolo.
Tale struttura permette alle classi che necessitano di tradurre ruoli in permessi, di aver come unica dipendenza la classe Role, rendendo il meccanismo di traduzione completamente indipendente dalle classi che utilizzano tale servizio. Questo rende la gerarchia di classi mantenibile e facilmente estendibile.

6.2 Gerarchia permessi

Questa gerarchia effettua il confronto tra permessi posseduti dall'utente e permessi necessari per svolgere un'operazione.
La classe padre fornisce due metodi:

- un metodo pubblico **checkPermissions** a cui vengono passati una lista di permessi necessari a svolgere un operazione e l'utente che vuole svolgerla.
- un metodo protetto astratto **gerPermissions** a cui va passato l'utente. Questo metodo va definito nelle classi figlie e contiene la logica per la ricostruzione della lista di permessi per ogni risorsa.

Nella gerarchia di classi ci sono tante classi figlie quanti sono le risorse manipolabili. Ogni classe figlia definisce il metodo per la ricostruzione della lista di permessi posseduti dall'utente. Per

poter ricostruire la lista di permessi viene richiamato, per ogni ruolo posseduto dall'utente, il metodo di traduzione associato alla risorsa su cui s'intende effettuare un'operazione.
Una volta ricostruita la lista di permessi, è sufficiente controllare che tutti i permessi necessari per effettuare tale operazione, siano presenti tra quelli posseduti dall'utente.
Dato che, per ogni risorsa va richiamato una metodo di traduzione differente, è necessario ridefinire, per ogni risorsa possibile, il metodo che ricostruisce la lista di permessi posseduti dall'utente. Per questo motivo il metodo della classe padre "getPermissions" è astratto ed andrà ridefinito da tutte le classi della gerarchia.

7 Implementazione della soluzione

L'implementazione della soluzione progettata ha richiesto modifiche al codice ed alla base di dati.
Le modifica apportate al codice riguardano:

- Modifica del costruttore della classe utente in cui vengono istanziati gli oggetti relativi ai ruoli da esso posseduti.
- Realizzazione delle gerarchie di ruoli e permessi
- Sostituzione e aggiunta dei controlli sui permessi in tutto il codice dell'applicazione
- Implementazione dei metodi che definiscono i permessi necessari per svolgere le operazioni nelle classi delle risorse manipolabili dall'utente.

Nella base di dati sono stati modificati solo alcuni campi:

- Entità UtenteRegistrato: gli attributi relativi ai ruoli che un utente può assumere sono stati ridotti a:
  - Amministratore
  - Revisore
  - Navigatore

  tutti gli altri ruoli sono deducibili dalle associazioni presenti con le varie entità
- Entita Route e RouteElement: L'attributo "pubilc" è stato spostato dall'entità RouteElement a Route in accordo le specifiche discusse nei capitoli precedenti.

In seguito verrà approfondita la struttura del codice riguardante la gerarchia dei ruoli, quella dei permessi e le modifiche apportate alla classe utente e alle classi delle risorse manipolabili dall'utente.

7.1 Gerarchia Ruoli

Nella gerarchia di ruoli sono presenti alcune differenze tra la classe padre e le varie classi figlie.

7.1.1 Classe padre

La classe padre deve implementare i metodi di traduzione per tutte le risorse manipolabili.

```
abstract class Role{

    protected $user;

    public function __construct($user){
        $this->user = $user;
    }

    function getCollectionPermissions($collection = null){
```

```
        return array();
    }

    function getSpecificDigitalContentPermissions($content = null){
        return array();
    }

    function getGenericDigitalContentPermissions($content = null){
        return array();
    }

    function getPhysicalPermissions($collection = null,$physical = null){
        return array();
    }

    function getInfoPermissions($info = null){
        return array();
    }

    function getInfoVersionPermissions($infoVersion = null){
        return array();
    }

    function getRoutePermissions($route = null){
        return array();
    }

    function getRouteElementPermissions($route = null,$routeElement = null){
        return array();
    }

    function getUserPermissions(){
        return array();
    }
}
```

In accordo con quanto detto in precedenza, la classe padre restituisce solo liste di permessi vuote. Questo indica che, di default, un ruolo non fornisce alcun permesso sulle risorse.

7.1.2 Classi figlie

Le classi figlie hanno una struttura analoga al codice riportato sotto.
Ogni classe figlia deve:

- Definire una costante ID=”NOME_RUOLO”. L'id è importante perché viene usato dal costruttore della classe utente e serve a determinare se uno specifico ruolo è posseduto da un utente.
- Chiamare il costruttore della classe padre passandogli l'utente a cui il ruolo appartiene.
- Ridefinire i metodi associati alle risorse per le quali il ruolo fornisce permessi.

```
class ReviewerRole extends RegisteredRole{
    const ID = "REVIEWER";

    function __construct($user){
        parent::__construct($user);
    }

    //ridefinizione dei metodi relativi agli
    //oggetti su cui il ruolo corrente fornisce dei permessi

    function getInfoPermissions($info = null){

        return array(
            Permissions::VIEW,
            Permissions::CREATE,
            Permissions::DELETE,
            Permissions::ACCEPT,
            Permissions::EDIT,
        );
    }
```

```
    function getInfoVersionPermissions($info, $infoVersion = null){
        return array(
            Permissions::VIEW,
            Permissions::CREATE,
            Permissions::DELETE,
            Permissions::ASSIGN,
            Permissions::REVIEW,
        );
    }

    function getGenericDigitalContentPermissions($info, $content){
        return array(
            Permissions::CREATE,
            Permissions::VIEW,
            Permissions::DOWNLOAD,
            Permissions::EDIT,
            Permissions::DELETE,
        );
    }
}
```

7.2 Struttura generale delle classi permesso

Anche per questa gerarchia verranno analizzate la classe padre e la struttura generale delle classi figlie.

7.2.1 Classe padre

```
abstract class Permissions{
    const CREATE = 0;
    const EDIT = 1;
    const VIEW = 3;
    const DELETE = 4;
    const EDIT_INFO = 5;
    const DEFINE_CONVENTION = 6;
    const DEFINE_SHELF_CONVENTION = 7;
    const ASSIGN_CATALOGUER = 8;
    const ASSIGN_MANAGER = 9;
    const ASSIGN_EDITOR = 10;
    const VIEW_COLLOCATION = 11;
    const REVIEW = 12;
    const ACCEPT = 13;
    const PROPOSE = 14;
    const MANAGE_ROLES = 15;
    const DOWNLOAD = 16;
    const ASSIGN = 17;
    const SUBMIT = 18;
    const MOVE = 19;
    const SIGN_UP = 20;

    protected abstract function getPermissions($user);

    public function checkPermissions($requiredPermissions, $user){
        $userPermissions = $this->getPermissions($user);

        foreach ($permissions as $permission) {
            if(!in_array($permission, $userPermissions)){
                return false;
            }
        }
        return true;
    }
}
```

La classe fornisce tutti i permessi usati dalle varie parti dell'applicazione e definisce il metodo che effettua il confronto tra i permessi necessari per svolgere un operazione e quelli posseduti dall'utente. Da notare che il metodo per ricavare la lista di permessi posseduti da un utente è astratto ed andrà quindi definito nelle classi figlie.

7.2.2 Classi figlie

```
class CollectionPermissions extends Permissions{
    private $collection;

    function __construct($collection = null){
        $this->collection = $collection;
    }

    protected function getPermissions($user){
        $permissions = array();
        $roles = $user->getRoles();
        foreach ($roles as $role){
            $permissions = array_merge(
                      $permissions,
                      $role->getCollectionPermissions($this->collection)
                );
        }
        return $permissions;
    }
}
```

La classe in questione ricava la lista di permessi posseduti da un utente su una collezione. Per ogni ruolo posseduto dall'utente, essa richiama il metodo di traduzione "getCollectionPermissions" in modo da ricavare i permessi posseduti da un utente su quella particolare collezione.

7.3 Classe utente

```
class User extends Model {

        //...codice omesso...

        private $roles;

        public function getRoles(){
                return $this->roles;
        }

        protected function __construct (
                /*parametri omessi*/,
                $admin_role,
                $info_reviewer_role,
                $navigator_role,
                $cataloguer_at,
                $manager_at,
                $editor_at,
                $created_routes
        ){
                parent::__construct($db, NULL);

                //...codice omesso...
                $this->roles = array();
                if($this->isGuest()){
                        $this->roles[UnregisteredRole::ID] = new UnregisteredRole($this);
                }
                else{
                        $this->roles[RegisteredRole::ID] = new RegisteredRole($this);
                        if(count($manager_at)>0){
                                $this->roles[ManagerRole::ID] = new ManagerRole(
                                     $manager_at,
                                     $this
                                 );
                        }
                        if(count($cataloguer_at)>0){
                                $this->roles[CataloguerRole::ID] = new CataloguerRole(
                                     $cataloguer_at,
                                     $this
                                 );
                        }
                        if(count($editor_at)>0){
                                $this->roles[EditorRole::ID] = new EditorRole($editor_at, $this);
                        }
```

```
                        if($admin_role){
                                $this->roles[AdminRole::ID] = new AdminRole($this);
                        }
                        if($info_reviewer_role){
                                $this->roles[ReviewerRole::ID] = new ReviewerRole($this);
                        }
                        if($navigator_role){
                                $this->roles[NavigatorRole::ID] = new NavigatorRole(
                                     $created_routes,
                                     $this
                                 );
                        }
                }
        }

       //...codice omesso...

}
```

Nella classe utente vengono creati gli oggetti che rappresentano i ruoli posseduti dall'utente e che ne permettono la traduzione.

7.3.1 Classi delle risorse manipolabili

```
class Collection extends Model {

        //...codice omesso...

        private $collection_permissions_;

        protected function __construct (/*parametri omessi*/)
        {
                //...codice omesso...

                $this->collection_permissions_ = new CollectionPermissions($this);
        }


        //...codice omesso...
        public static function canBeCreatedBy($user){
                 $permission = array(Permissions::CREATE, Permissions::ASSIGN_MANAGER);
                 $collectionPermissions = new CollectionPermissions();
                 return $collectionPermissions->checkPermissions($permission, $user);
        }

        public static function canBeCataloguersAssignedBy($user,$collection){
                 $permission = array(Permissions::ASSIGN_CATALOGUER);
                 $collectionPermissions = new CollectionPermissions($collection);
                 return $collectionPermissions->checkPermissions($permission, $user);
        }

        public static function canBeManagerAssignedBy($user,$collection){
                 $permission = array(Permissions::ASSIGN_MANAGER);
                 $collectionPermissions = new CollectionPermissions($collection);
                 return $collectionPermissions->checkPermissions($permission, $user);
        }

        public function canBeEditedBy($user){
                 $permission = array(Permissions::EDIT);
                 return $this->collection_permissions_->checkPermissions($permission, $user);
        }

        public function canBeInfoEditedBy($user){
                 $permissions =array(Permissions::EDIT_INFO);
                 return $this->collection_permissions_->checkPermissions($permissions, $user);
        }

        public function canBeDeletedBy($user){
                 $permission = array(Permissions::DELETE);
                 return $this->collection_permissions_->checkPermissions($permission, $user);
        }

        public function canBeConventionDefinedBy($user){
                 $permissions =array(
```

```
                    Permissions::DEFINE_CONVENTION,
                    Permissions::DEFINE_SHELF_CONVENTION
                );
                return $this->collection_permissions_->checkPermissions($permissions, $user);
        }

        public function canBeViewedBy($user){
                $permissions =array(Permissions::VIEW);
                return $this->collection_permissions_->checkPermissions($permissions, $user);
        }
}
```

Nel codice è riportata la struttura generale delle classi relative alle risorse manipolabili dall'utente.

Nel costruttore viene istanziato l'oggetto della gerarchia permissions, necessario per interfacciarsi con il meccanismo di controllo.

Sono stati aggiunti i metodi che, per ogni operazione possibile, definiscono la lista dei permessi necessari al fine di svolgere l'operazione e ne controllano la fattibilità richiamando la checkPermissions fornita dal meccanismo di controllo.

Da notare che esistono operazioni, come la create, che vengono effettuate quando la collezione, e di conseguenza l'oggetto ad essa associato, non esiste. In questo caso il metodo per il controllo di fattibilità dell'operazione sarà statico.

8 Conclusioni

L'implementazione del meccanismo per il controllo dei permessi ha prodotto i risultati sperati.
Gli accessi non autorizzati vengono bloccati facendo funzionare l'applicazione nel modo corretto.
I vari errori relativi alla visualizzazione di contenuti non consentiti non sono più presenti.
La struttura adottata per realizzare il meccanismo di controllo consente di aggiungere o rimuovere i ruoli e modificare agevolmente i permessi da essi forniti.
Dal punto di vista dello sviluppo e del codice, risulta ora molto più semplice aggiungere controlli relativi agli accessi nelle varie parti dell'applicazione. I punti in cui i sono presenti dei controlli sono ben distinguibili e leggibili. Essendo inoltre la logica di controllo dei permessi ben incapsulata, non è necessario conoscere dettagli sul funzionamento del meccanismo di controllo per poter verificare la fattibilità delle operazioni.

9 Sviluppi futuri

L'implementazione e l'adattamento dell'applicazione al nuovo meccanismo di controllo hanno reso possibile individuare parti dell'applicazione funzionanti in modo scorretto o migliorabili.
In aggiunta alla riprogettazione della parte relativa ai percorsi proposta da E. Meloni[3] sono migliorabili le seguenti parti.

- Architettura di classi relativa ai contenuti digitali.
- Modifica delle schede informative.

9.1 Architettura di classi relativa ai contenuti digitali

Le classi relative ai contenuti digitali, cercano di rispecchiare l'architettura adottata nella base di dati. Il problema di tale architettura è che, a livello di codice, non rispecchia il modo in cui i contenuti dovrebbero essere gestiti.
Nello specifico, la corrente architettura adottata dai contenuti digitali è la seguente:

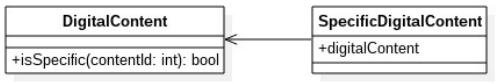

Illustrazione 7

Nella classe DigitalContent sono presenti sia metodi comuni a contenuti generici e specifici, sia metodi appartenenti ai soli contenuti digitali generici. Tale classe viene infatti usata nell'applicazione come se fosse la classe GenericDigitalContent.
Un ulteriore problema si presenta nell'utilizzo di un contenuto specifico. Esso infatti richiama metodi presenti nella classe DigitalContent che, come già detto, contiene anche metodi specifici per i contenuti generici che non dovrebbe essere possibile chiamare in un contenuto specifico.
L'architettura consigliabile in questo caso sarebbe la seguente:

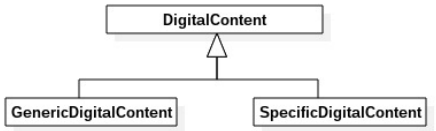

Illustrazione 8

Tale architettura porterebbe ad una notevole semplificazione del codice relativo ai contenuti digitali e diminuirebbe notevolmente la complessità del codice relativo alla manipolazione di tali contenuti, specialmente nella View, dove i problemi introdotti dall'architettura corrente sono parecchio evidenti.

9.2 Modifica schede informative

Una scheda informativa è formata da varie sue versioni. Lo scopo delle versioni è quelli di mantenere informazioni sulle varie modifiche che una scheda subisce nel corso del tempo.

Un problema presente nella corrente implementazione dell'applicazione è dovuta al fatto che i contenuti digitali generici vengono associati alle schede informative e non alle versioni. Questo non fornisce nessuna informazione sulle manipolazioni che quel contenuto ha subito nelle varie versioni.

L'ideale sarebbe che un contenuto digitale venisse associato alle versioni di scheda informativa e che le varie modifiche che esso subisce siano mappate nel tempo.

Questo problema è già stato approfondito nel capitolo relativo alla traduzione dei ruoli in liste di permessi.

Bibliografia

1: A. Biaggini, Progetto e realizzazione di CHKB,
2: D. Aimini, Progettazione e Realizzazione di un Modulo per i Percorsi Virtuali in una Base di Conoscenza Informatica,
3: E. Meloni, Estensione e validazione di CHKB, una base di conoscenza web per il Museo degli Strumenti per il Calcolo,