Secure-UML

Secure-UML est une extension UML dont l'objectif est d'intégrer des aspects de sécurité directement dans les modèles de systèmes logiciels.
Cette extension permet de définir des politiques de sécurité et des règles d'accès au niveau de la conception, facilitant ainsi le développement de systèmes sécurisés dès les premières étapes du cycle de vie du logiciel.

Principes

Role-Based Access Control

Contrôle d'Accès Basé sur les Rôles (RBAC) :

Secure-UML utilise le modèle RBAC pour définir les permissions et les rôles des utilisateurs dans le système.

Le modèle RBAC associe les permissions aux rôles plutôt qu'aux individus, simplifiant la gestion des droits d'accès.

Rôles

Regroupent les permissions en fonction des responsabilités des utilisateurs.

Permissions

Actions autorisées que les rôles peuvent effectuer sur les objets du système (ressources).

Diagrammes de Secure-UML

Secure-UML étend les diagrammes standard d'UML pour inclure des informations de sécurité.

Diagrammes de Classes Sécurisés

Incluent des annotations pour spécifier les permissions et les rôles associés aux classes et aux opérations.

Diagrammes de Cas d'Utilisation Sécurisés

Identifient les rôles et les permissions nécessaires pour chaque cas d'utilisation.

Éléments Clés de Secure-UML

Annotations de Sécurité

Utilisées pour enrichir les éléments UML avec des informations de sécurité.

Stereotypes

Ajoutent des informations de sécurité aux éléments UML.

Par exemple, un stéréotype « role » peut être utilisé pour annoter une classe comme représentant un rôle dans le modèle RBAC.

Constraints

Règles qui spécifient les conditions de sécurité.

Par exemple, une contrainte peut définir que seulement certains rôles peuvent accéder à une méthode particulière.

Modélisation des Politiques de Sécurité

Les politiques de sécurité sont définies en termes de permissions (opérations autorisées) et de rôles (regroupements de permissions).

Définition des Rôles

Les rôles sont définis en fonction des responsabilités dans l'organisation.

Assignation des Permissions

Les permissions sont associées aux rôles plutôt qu'aux utilisateurs individuels.

Exemples

Gestion d'une bibliothèque :

Rôles :

- Bibliothécaire : Un utilisateur avec le rôle de Bibliothécaire.
- Membre : Un utilisateur avec le rôle de Membre.

Permissions :

- Lecture : Inclut les actions de lire, emprunter et retourner des livres.
- Modification : Inclut les actions d'ajouter, modifier et supprimer des livres.
- Administration : Inclut l'action de gérer les emprunts.

Cas d'utilisation :

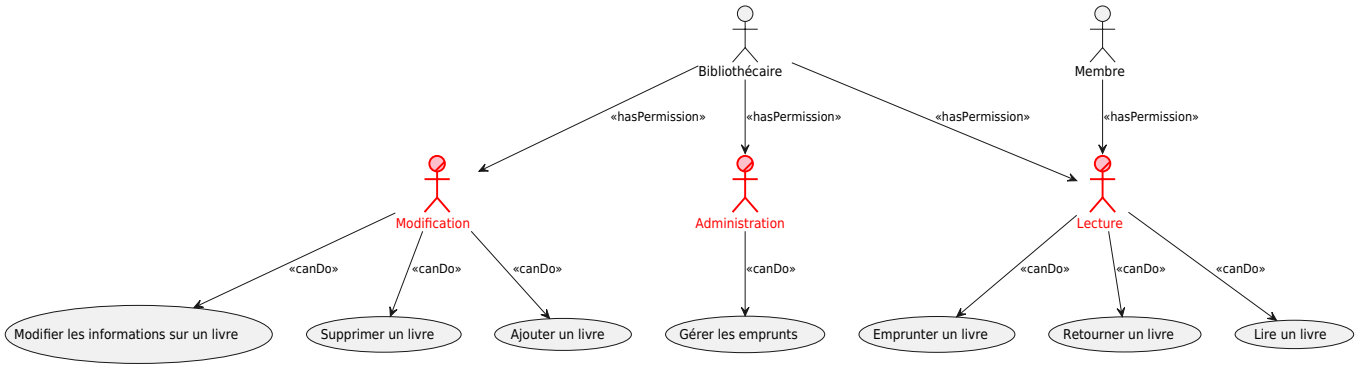
- Lecture : Lire un livre, emprunter un livre, retourner un livre.
- Modification : Ajouter un livre, modifier les informations sur un livre, supprimer un livre.

- Administration : Gérer les emprunts.

Dans ce diagramme, les rôles “Bibliothécaire” et “Membre” sont associés aux permissions appropriées.

Les permissions sont ensuite reliées aux cas d'utilisation correspondants, montrant comment les actions du système sont contrôlées par les rôles et les permissions.

Secure Use case diagram

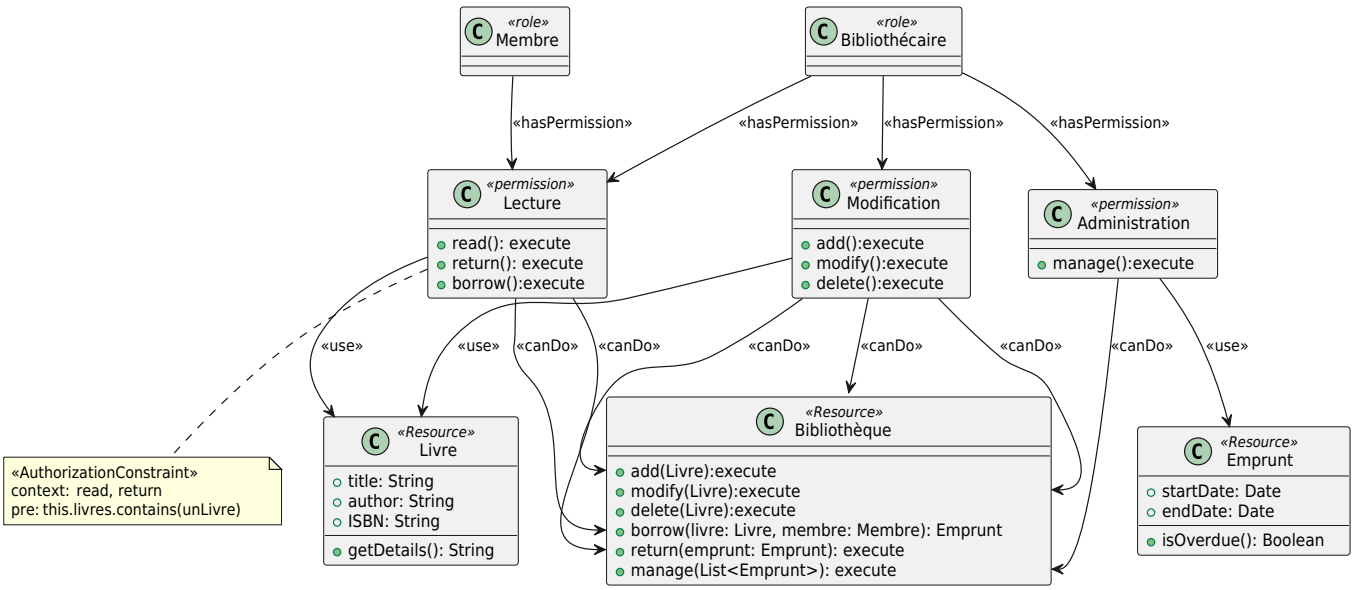


Source

```
@startuml

!define ROLE Actor
!define PERMISSION Actor/ #pink;line:red;line.bold;text:red

' Définition des rôles
ROLE Bibliothécaire
ROLE Membre

' Définition des permissions
PERMISSION Lecture
PERMISSION Modification
PERMISSION Administration

' Acteurs (rôles) et leurs permissions
Bibliothécaire -down-> Lecture : <<hasPermission>>
Bibliothécaire -down-> Modification : <<hasPermission>>
Bibliothécaire -down-> Administration : <<hasPermission>>

Membre -down-> Lecture : <<hasPermission>>

' Cas d'utilisation du système
Lecture --> (Lire un livre) : <<canDo>>
Lecture --> (Emprunter un livre) : <<canDo>>
Lecture --> (Retourner un livre) : <<canDo>>
Modification --> (Ajouter un livre) : <<canDo>>
Modification --> (Modifier les informations sur un livre) : <<canDo>>
Modification --> (Supprimer un livre) : <<canDo>>
Administration --> (Gérer les emprunts) : <<canDo>>
```

```
@enduml
```

Secure Class Diagram



Source

```
@startuml

!define ROLE Class
!define PERMISSION Class

' Définition des classes pour les rôles
ROLE Bibliothécaire <<role>>
ROLE Membre <<role>>

' Définition des permissions
PERMISSION Lecture <<permission>> {
  + read(): execute
  + return(): execute
  + borrow():execute
}
PERMISSION Modification <<permission>> {
  + add():execute
  + modify():execute
  + delete():execute
}
PERMISSION Administration <<permission>> {
  + manage():execute
}

' Association des rôles avec les permissions
Bibliothécaire -down-> Lecture : <<hasPermission>>
Bibliothécaire -down-> Modification : <<hasPermission>>
Bibliothécaire -down-> Administration : <<hasPermission>>
```

```
Membre -down-> Lecture : <<hasPermission>>

' Classes du système
class Livre <<Resource>> {
  +title: String
  +author: String
  +ISBN: String
  +getDetails(): String
}

class Emprunt <<Resource>> {
  +startDate: Date
  +endDate: Date
  +isOverdue(): Boolean
}

class Bibliothèque <<Resource>> {
  +add(Livre):execute
  +modify(Livre):execute
  +delete(Livre):execute
  +borrow(livre: Livre, membre: Membre): Emprunt
  +return(emprunt: Emprunt): execute
  +manage(List<Emprunt>): execute
}

' Permissions pour les méthodes de la classe Bibliothèque
Lecture --> Livre : <<use>>
Lecture --> Bibliothèque::borrow : <<canDo>>
Lecture --> Bibliothèque::return : <<canDo>>

Modification --> Bibliothèque::add : <<canDo>>
Modification --> Bibliothèque::remove : <<canDo>>
Modification --> Bibliothèque::modify : <<canDo>>
Modification --> Livre : <<use>>

Administration --> Bibliothèque::manage : <<canDo>>
Administration --> Emprunt : <<use>>

@enduml
```