ANR programme ARPEGE 2008

Systèmes Embarqués et Grandes Infrastructures

Projet SELKIS : Une méthode de développement de systèmes d'information médicaux sécurisés : de l'analyse des besoins à l'implémentation.

ANR-08-SEGI-018

Février 2009 - Décembre 2011

Implementation of the Res@mu model

Livrable 6.1.3

Akram Idani
Yves Ledru
Jean-Luc Richier
Mohamed-Amine Labiadh

Novembre 2012

Table des matières

1 Introduction **2**

2 Modèle Res@mu **3**

2.1 Retour sur le modèle fonctionnel 3

2.1.1 Vue d'ensemble 3

2.1.2 Intérêt des classes PreHospitalActor et Person pour le modèle fonctionnel 3

2.2 Le modèle de sécurité 6

2.2.1 Les rôles 6

2.2.2 Principe du filtre de sécurité 7

2.2.3 Les prises en charge 8

2.2.4 Les actes de soin 9

2.3 Spécifications B 11

3 Validation de la politique de contrôle d'accès **12**

3.1 Test à base de cas d'utilisation 12

3.2 Test systématique des règles de sécurité 12

3.3 Recherche de scénarios d'attaque 14

4 Conclusion **16**

A Specifications B issues du modèle de sécurité **17**

A.1 Partie statique 17

A.2 Initialisation 20

A.3 Calcul des permissions : ensemble isPermitted 26

Chapitre 1

Introduction

Ce livrable donne un aperçu de l'usage des modèles Graphiques et leurs contreparties en B de l'étude de cas Res@mu en vue de simuler le fonctionnement concret du système. Il ne présente pas une réelle implémentation du système d'information dans un langage de programmation, mais s'attache en revanche à valider la politique de sécurité par une activité de test. Ces tests ont été réalisés sur le modèle et peuvent être adaptés à sa future implémentation.

Nous présenterons dans un premier lieu (Chapitre 2) quelques choix techniques nécessitant le perfectionnement de nos spécifications B ainsi que les modèles de sécurité graphiques. Ensuite, dans le chapitre 3 nous donnerons une synthèse des activités de test réalisées sur la base des spécifications B que nous avons obtenues. Ces activités de test se sont effectuées au moyen de l'animateur ProB et reflètent une démarche de développement dirigée par les tests (TDD : test driven development). Cette démarche se traduit par les étapes suivantes :

1. Générer les spec B à partir des modèles graphiques en SecureUML au moyen de la plate-forme B4MSecure
2. `Pour chaque` contrainte de sécurité `faire`
   Ecrire tous les tests associés
   Vérifier que ces tests échouent
   Spécifier la contrainte en B
   `Tant que` il existe un test qui échoue `faire`
   Corriger les specifications B
   `Fin tant que`
   `Fin faire`
3. Rejouer tous les tests pour vérifier la non-regression

Les deux activités (écriture de la politique de sécurité et validation de celle-ci) ont été menées de concert par deux personnes différentes (pour garantir l'indépendance entre développeur et testeur).

Chapitre 2

Modèle Res@mu

2.1 Retour sur le modèle fonctionnel

2.1.1 Vue d'ensemble

Le sous-ensemble du modèle fonctionnel Res@mu présenté à la figure 2.1 se focalise sur les classes impliquées dans la gestion de la prise en charge de patients. Le livrable 6.1.2 a présenté les différents choix conceptuels concernant ce modèle, ainsi que le processus de sa validation en B. Dans ce livrable nous présentons le modèle de sécurité correspondant et nous discutons sa validation par l'animation. La cible de sécurité étant la classe *ManagementAct*, nous allons nous concentrer sur divers scénarios permettant la création, l'accès et la modification de cette cible. La réalisation d'un acte de soin pour un patient nécessite l'appel à diverses opérations :

– la création du patient : opération `Patient_NEW`
– la création d'une intervention : opération `Intervention_NEW`
– la création d'une équipe liée à l'intervention créée dans l'étape précédente : opération `Team_NEW`
– l'ajout de membres à l'équipe : opération `Team__addMembers`
– l'association de l'intervention au patient : opération `Intervention__AddA_Intervention_Patient`
– la création d'une prise en charge associant le patient, l'équipe et l'intervention : opération `Management_NEW`
– la création d'un acte de soin lié à la prise en charge : opération `ManagementAct__NEW`

Outre, les opérations de base (de lecture et de modification) permettant la manipulation des différentes données créées au travers des opérations précédentes, le modèle fonctionnel intègre des opérations spécifiques telles que les opérations `Intervention_Close`, `Team__removeMembers`, `ManagementAct__validate`, etc. L'animation de ces diverses opérations a permis la réalisation de l'ensemble des scénarios de base fournis par Res@mu. Les spécifications B du modèle fonctionnel couvrent donc les données et contraintes nécessaires à la mise en place des prises en charge de patients et la réalisation d'actes de soin.

2.1.2 Intérêt des classes PreHospitalActor et Person pour le modèle fonctionnel

Dans notre démarche de génération de spécifications B nous avons clairement suggéré une séparation entre le modèle fonctionnel et le modèle de sécurité en vue de pouvoir utiliser ce dernier comme étant un simple filtre vérifiant des droits d'accès. Toutefois, dans le cadre de l'étude de cas Res@mu cette séparation n'est pas aussi claire car certains concepts, principalement liés aux rôles des utilisateurs peuvent être déduits du modèle fonctionnel. En effet, l'existence des classes `PreHospitalActor` et `Person` a nécessité des adaptations de notre plateforme de génération de spécifications B.

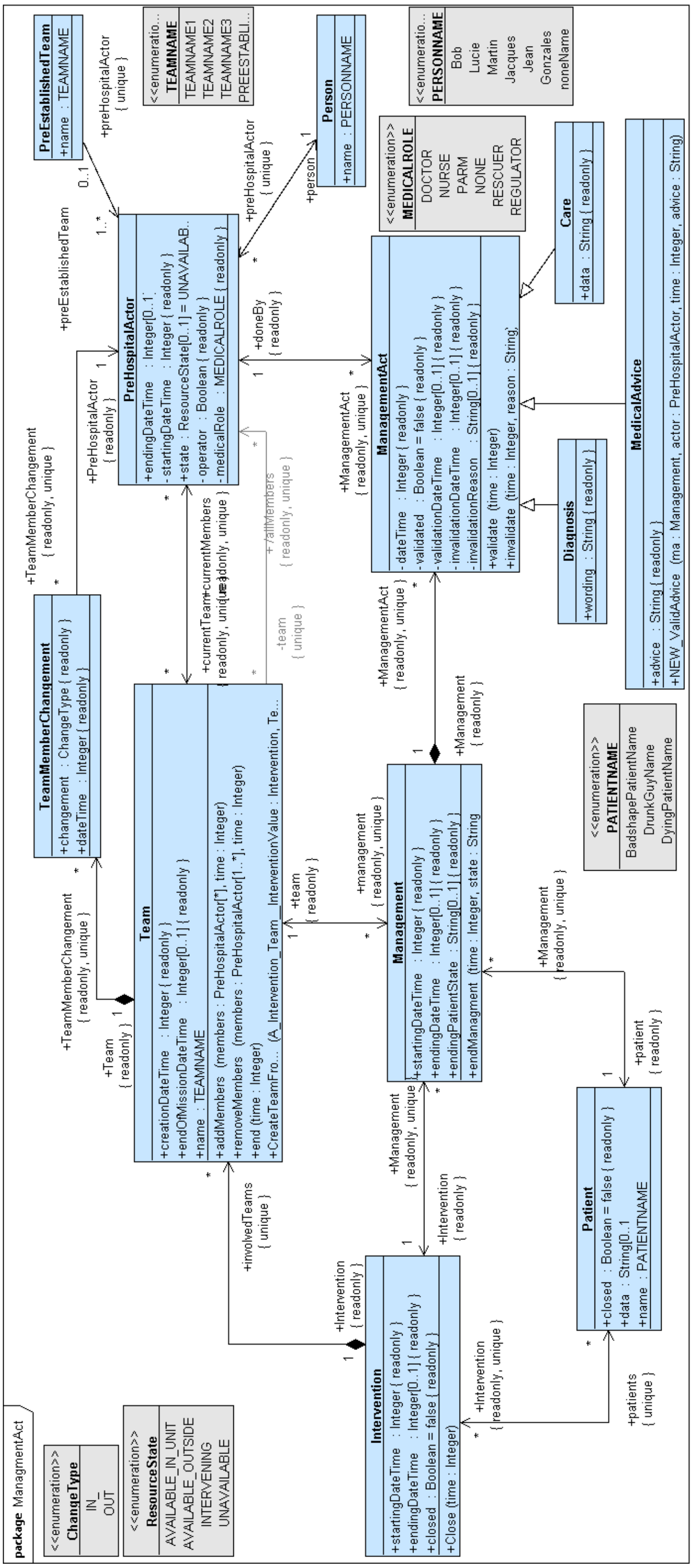

Figure 2.1 – Modèle fonctionnel Res@mu

La classe *Person* représente les utilisateurs du système d'information. Nous considérons que ceux-ci ne peuvent effectuer des actions que s'ils sont rattachés à au moins une instance de `PreHospitalActor`. Cela leur permettra de jouer un ou plusieurs rôles et d'obtenir par conséquent certains droits. La figure 2.2 illustre un exemple où la personne nommée *Bob* est perçue par le SI comme étant un acteur pré-hospitalier pouvant jouer les rôles PARM et Nurse. Par contre, la personne nommée *Martin* est reconnue par le système mais n'aura aucun accès possible. En effet, elle n'est associée à aucun objet de la classe `PreHospitalActor` et ne peut dès lors jouer aucun rôle.

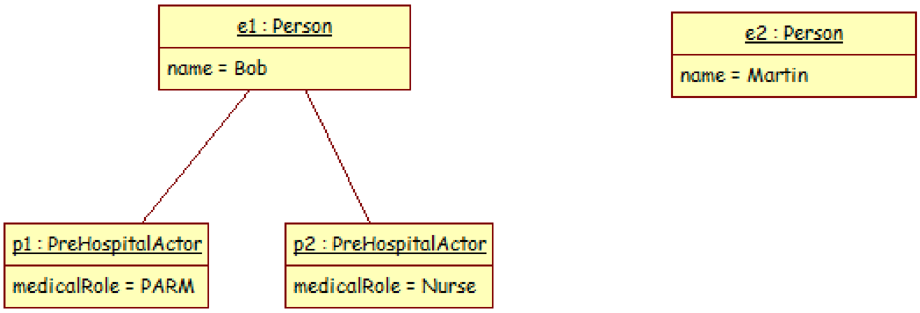

FIGURE 2.2 – Exemple d'instances des classes Person et PreHospitalActor

Remarquons aussi que la classe `Team` n'est pas directement liée à la classe `Person`. Le modèle fonctionnel considère qu'une équipe est un ensemble d'acteurs pré-hospitaliers ayant chacun un rôle spécifique dans l'équipe. Par exemple, Bob peut jouer le rôle PARM dans une équipe, mais ensuite il peut devenir Nurse dans cette même équipe ou dans une autre équipe. Une contrainte fonctionnelle renforce le modèle en vue d'éviter le cas où une même personne à un instant donné serait affectée à deux équipes distinctes.

La classe fonctionnelle `PreHospitalActor` joue donc un rôle central dans la politique de contrôle d'accès de par le fait qu'elle détermine les rôles auxquels les utilisateurs sont affectés. Elle définit également les responsabilités que ces utilisateurs peuvent avoir. La figure 2.3 est un complément au diagramme de classes de la figure 2.1 dans lequel chaque patient et chaque intervention dispose nécessairement d'un responsable.

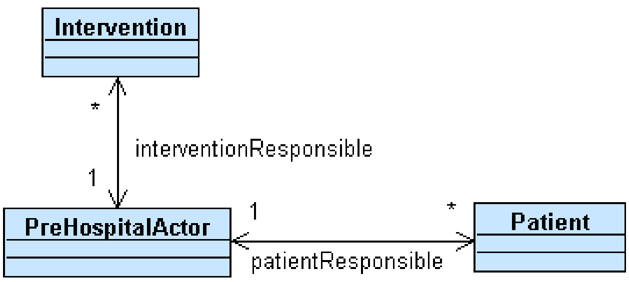

FIGURE 2.3 – Affectation des responsabilités

Notons que ces liens de responsabilité sont créés par l'opération fonctionnelle de base correspondant au constructeur d'instances effectives. En effet, puisque la multiplicité du côté de `PreHospitalActor` est égale à 1 aussi bien pour `interventionResponsible` que pour `patientResponsible`, les constructeurs des classes `Intervention` et `Patient` disposent d'un paramètre formel de type

`PreHospitalActor`. Lors de l'animation des opérations de construction des instances de ces deux classes on s'assure que le paramètre effectif correspond à l'utilisateur courant réalisant l'action de création (à condition que cet utilisateur en soit autorisé). Nous reviendrons plus tard sur les contraintes contextuelles qui en découlent.

2.2 Le modèle de sécurité

2.2.1 Les rôles

La figure 2.4 récapitule la liste des rôles explicités lors de l'analyse des besoins du système Res@mu. Pour plus de details concernant ce diagramme, le lecteur pourra se référer au livrable 6.1.1.

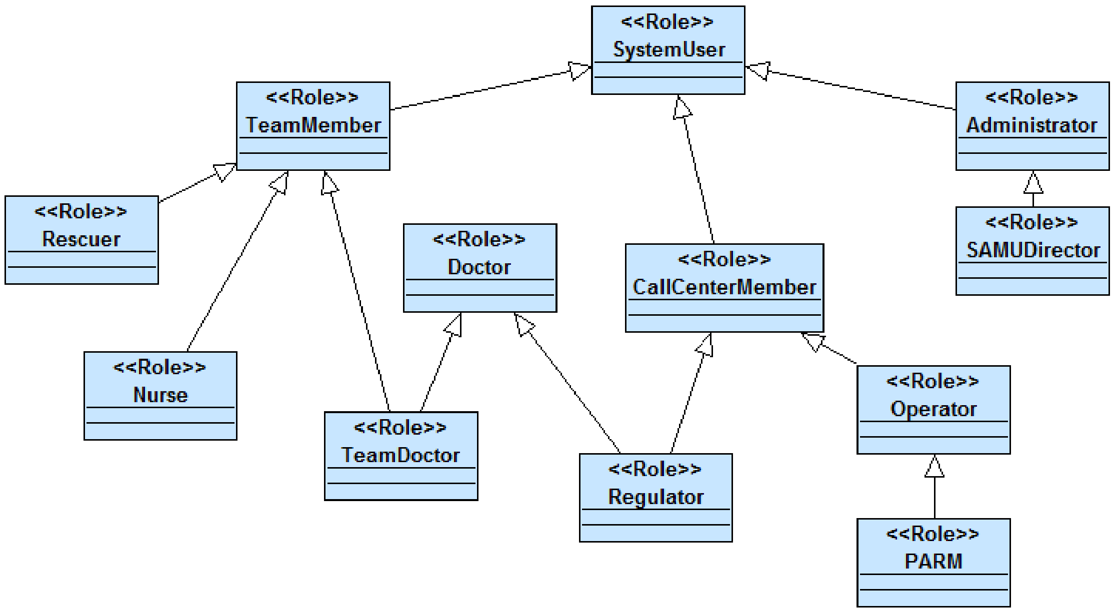

FIGURE 2.4 – Rôles issues de la politique de sécurité

Habituellement nous traduisons en B les concepts "Rôle" et "User" par les ensembles énumérés ROLES et USERS dont les éléments correspondent aux concepts de modélisation stéréotypés par ≪Role≫ et ≪User≫. Cependant, pour ce cas d'étude nous considérons que ces ensembles sont des constantes tel que :

$$\mathrm{ROLES} = \mathrm{MEDICALROLE}$$
$$\mathrm{USERS} = \mathrm{PERSON}$$

où les ensembles MEDICALROLE et PERSON correspondent aux ensembles d'instances possibles issues de classes du modèle fonctionnel. Ce faisant, nous n'explicitons pas sur le modèle de sécurité la relation d'affectation des utilisateurs aux rôles, nommée `roleOf` et définie par :

$$\mathrm{roleOf} : \mathrm{USERS} \nrightarrow \mathrm{POW(ROLES)}$$

En effet, nous avons adapté note transformation pour que cette relation soit déduite d'informations issues du modèle fonctionnel. Il s'agit de composer les relations $A\_preHospitalActor\_person$ et $PreHospitalActor\_\_medicalRole$:

$$\text{roleOf} := \text{fnc}(\text{A\_preHospitalActor\_person}^{-1}; \text{PreHospitalActor\_\_medicalRole})$$

Notons que dans le modèle fonctionnel, nous avons :
– l'association entre les classes Person et PreHospitalActor est traduite par :

$$\text{A\_preHospitalActor\_person} \in \text{PreHospitalActor} \rightarrow \text{Person}$$

– l'attribut obligatoire medicalRole de la classe PreHospitalActor est traduit par :

$$\text{PreHospitalActor\_\_medicalRole} \in \text{PreHospitalActor} \rightarrow \text{MEDICALROLE}$$

Par conséquent, la relation `roleOf` indique pour chaque instance de la classe `Person` l'ensemble des rôles auxquels elle est affectée. Elle correspond donc à un ensemble de couples issus de la relation Person $\leftrightarrow$ POW(MEDICALROLE).

2.2.2 Principe du filtre de sécurité

Le filtre de sécurité a été amplement discuté dans le livrable 3.1. Nous ne reviendrons pas sur ses principes ici ; nous rappelons en revanche qu'il correspond à une traduction en B de règles de sécurité exprimées graphiquement auxquelles nous ajoutons les diverses contraintes d'autorisation. Concrètement, nous produisons pour chaque opération fonctionnelle, sa contrepartie dans une machine B de sécurité en vue de vérifier si l'utilisateur courant a le droit d'invoquer l'opération. A titre d'exemple nous prenons le cas de l'opération `Patient_NEW` à partir de laquelle nous produisons une opération nommée `Secure_Patient_NEW` et dont la spécification est comme suit :

Secure_Patient_NEW(Instance, patientResponsible__preHospitalActorValue, Patient__nameValue) =
PRE
    Instance : PATIENT
    $\wedge$ Instance $\notin$ Patient
    $\wedge$ patientResponsible__preHospitalActorValue $\in$ PreHospitalActor
    $\wedge$ Patient__nameValue $\in$ PATIENTNAME
THEN
  SELECT
    Patient_NEW_Label : isPermitted[currentRole]
    `/* Ajout de contraintes d'autorisation */`
  THEN
    Patient_NEW(Instance, patientResponsible__preHospitalActorValue, Patient__nameValue)
  END
END

L'opération sécurisée dispose de la même signature que l'opération fonctionnelle c'est pourquoi sa pré-condition correspond simplement au typage de différents paramètres. Rappelons que la création d'une instance de la classe `Patient` prend un élément de l'ensemble des instances possibles (nommé PATIENT) qui n'appartient pas à l'ensemble des instances effectives (ensemble Patient). En outre, deux informations sont nécessaires à la création d'un patient : le nom (attribut obligatoire) et un acteur pré-hospitalier responsable (multiplicité 1 sur l'association patientResponsible, voir figure 2.3).

La clause SELECT de `Secure_Patient_NEW` correspond à une garde conditionnant le déclenchement de l'opération fonctionnelle. Il s'agit de réaliser deux vérifications :

1. Vérifier que l'utilisateur courant, au travers de ses rôles courants, a bien la permission de réaliser l'opération.
2. Vérifier que les contraintes d'autorisations sont respectées pour l'état courant des spécifications.

Si ces deux conditions sont évaluées à `true` alors l'opération fonctionnelle `Patient_NEW` peut être appelée. Un animateur, comme ProB, sera en mesure de calculer l'état obtenu après l'animation de l'opération, et montrer si l'invariant est respecté (ou pas).

2.2.3 Les prises en charge

La figure 2.5 présente les règles de contrôle d'accès graphiques permettant la mise en place d'une prise en charge par un `CallCenterMember`.

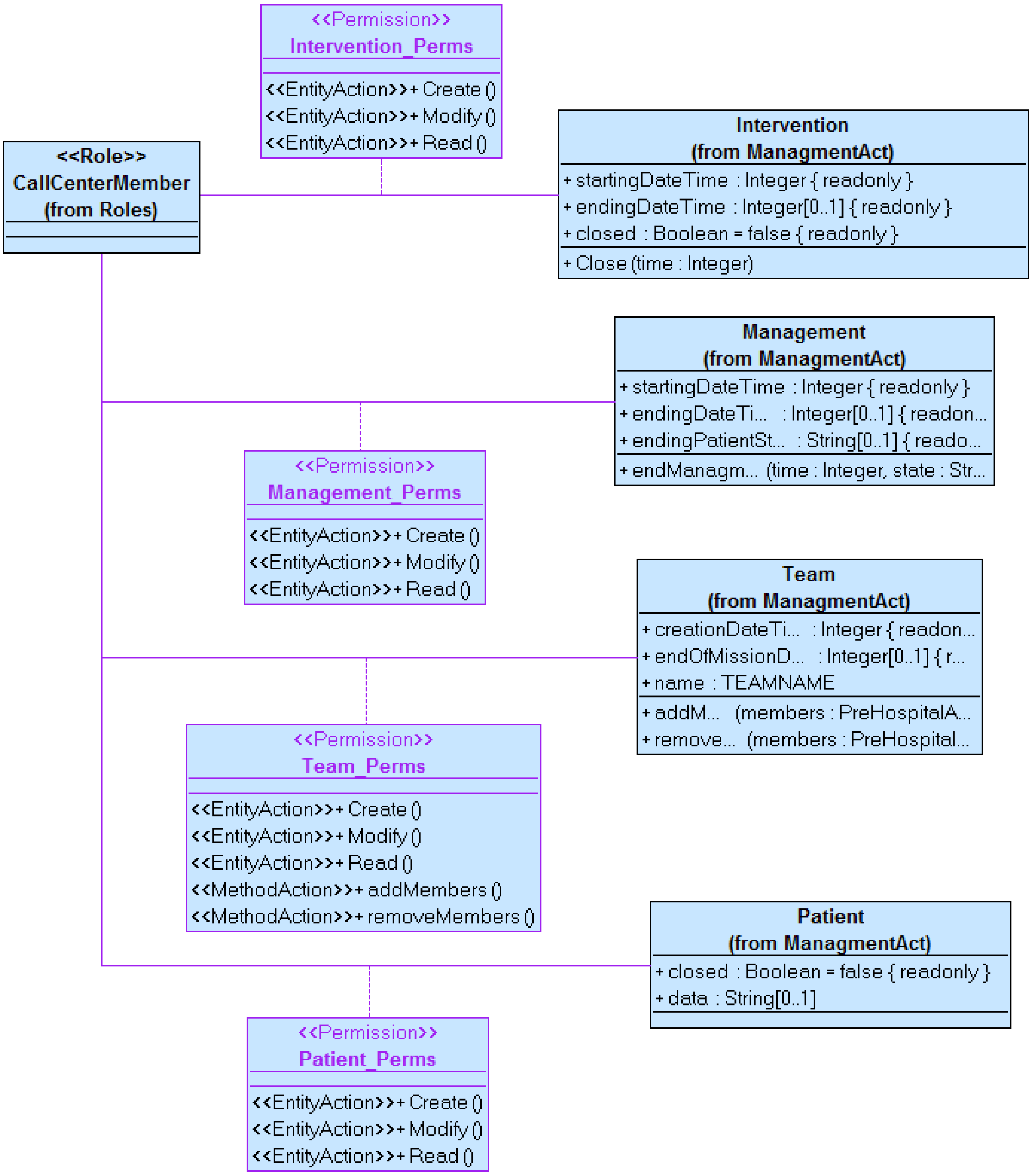

FIGURE 2.5 – Règles de contrôle d'accès pour la mise en place d'une prise en charge

Contraintes d'autorisation

Le figure 2.5 montre qu'un utilisateur connecté au système via le rôle `CallCenterMember` (ou un de sous-rôles de `CallCenterMember`) a le droit de créer un patient ainsi que de modifier, ou lire les informations d'un patient. Toutefois, la lecture et la modification ne sont possibles que pour le responsable du patient. Nous renforçons donc la garde de toutes les opérations de lecture et de modification par le prédicat suivant :

$$\mathrm{A\_preHospitalActor\_person(patientResponsible(Instance))} = \mathrm{currentUser}$$

- Instance : élément de l'ensemble Patient et correspond à l'instance de la classe Patient objet de l'opération de lecture ou de modification ;
- patientResponsible(Instance) : donne l'instance de la classe PreHospitalActor associée à Instance ;
- A_preHospitalActor_person(patientResponsible(Instance)) : donne l'instance de Person associée au PreHospitalActor en question.

Cette contrainte d'autorisation fait référence à l'état du modèle fonctionnel et doit être satisfaite pour l'utilisateur courant (currentUser).

Concernant l'opération de création de patients, elle a été améliorée pour que le responsable du patient soit celui qui l'a créé. Nous introduisons donc dans l'opération `Secure_Patient_NEW` le prédicat suivant :

$$\mathrm{patientResponsible\_\_preHospitalActorValue} \in \mathrm{A\_preHospitalActor\_person}\ [\{\mathrm{currentUser}\}]$$

Ce prédicat garantit que le paramètre effectif patientResponsible__preHospitalActorValue lors de l'appel de l'opération `Secure_Patient_NEW` est l'une des instances de PreHospitalActor associées à l'utilisateur courant.

Dans la suite, nous n'allons pas présenter les expressions B pour les différentes contraintes. Etant donné que le principe est similaire, nous allons nous contenter de les exprimer de manière informelle.

Intervention

- Pour lire ou modifier une intervention il faut être son responsable
- Le responsable d'une intervention est la personne qui l'a créé

Team

- Pour créer, lire ou modifier une équipe il faut être le responsable de l'intervention à laquelle l'équipe est rattachée

Management

- Pour créer, lire ou modifier un Management il faut être le responsable de l'Intervention à laquelle est associé le Management.

2.2.4 Les actes de soin

Les actes de soin sont modélisés par la classe `ManagementAct` et ses sous-classes `Care`, `MedicalAdvice` et `Diagnosis`.

Règles de création et de modification d'actes de soin

Les actions de création et de modification d'actes de soin sont autorisées pour les utilisateurs jouant le rôle `TeamDoctor`. Ce faisant, les sous-classes de `ManagementAct` ont des règles d'autorisation spécifiques illustrées par la figure 2.6.

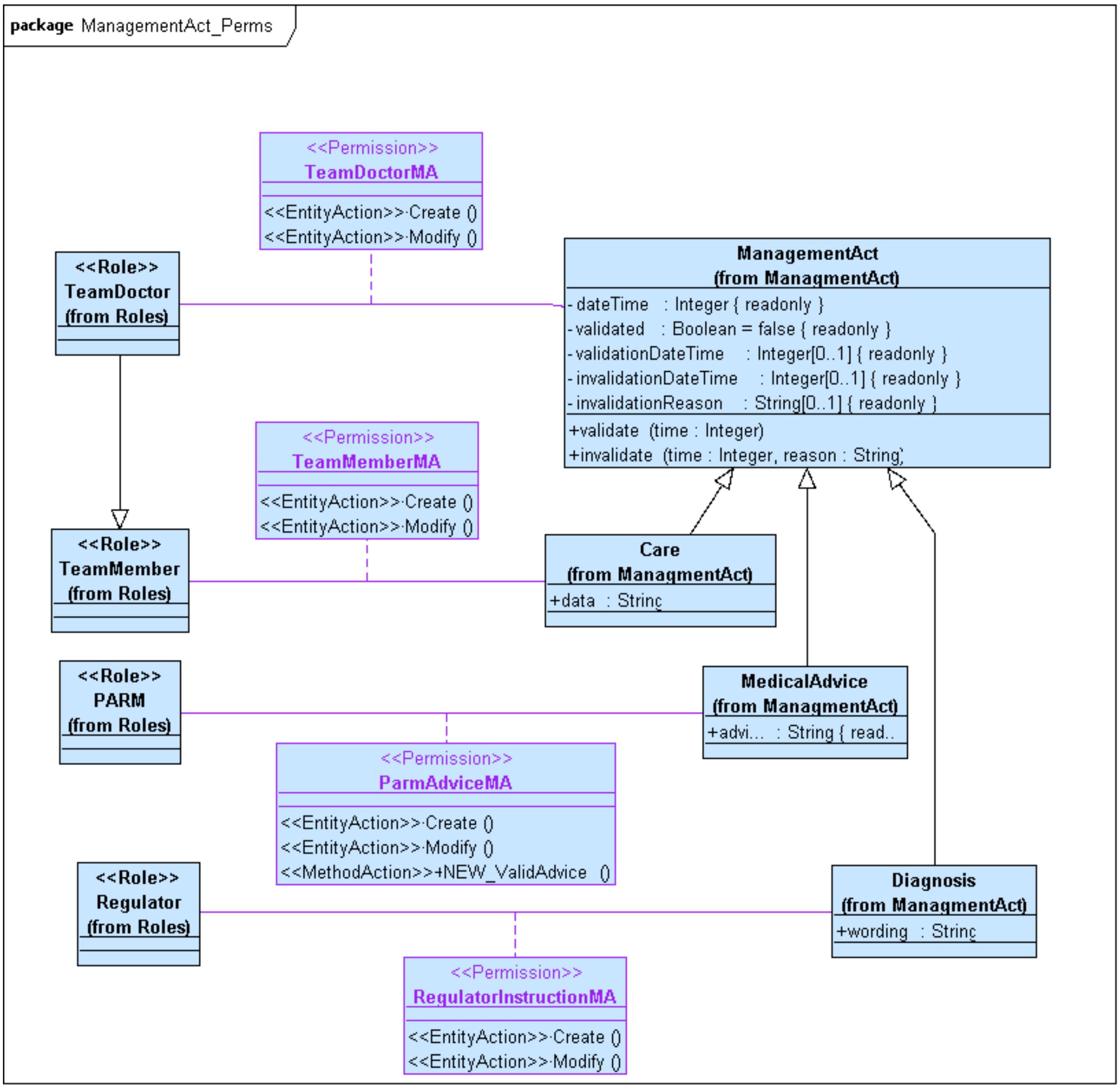

FIGURE 2.6 – Règles de création d'actes de soin

Les contraintes associées à ces règles sont les suivantes :
- Pour créer ou modifier un acte de soin il faut faire partie de l'équipe associée à sa prise en charge.
- Celui qui crée l'acte de soin est celui qui l'a réalisé (relation `doneBy` entre `ManagementAct` et `PreHospitalActor`).
- Les opérations de modification `validate` et `invalidate` ne sont autorisées que pour celui qui a réalisé l'acte de soin.

La lecture d'actes de soin est autorisée pour les rôles : `TeamMember`, `PARM` et `Regulator`. La figure 2.7 présente les règles de lecture d'actes de soin de type `MedicalAdvice`. Ces règles sont similaires à celles exprimées sur les classes `Care` et `Diagnosis`. Par ailleurs, la contrainte d'autorisation pour les opérations de lecture est la suivante :
- un `TeamMember` est autorisé à lire un acte de soin si il a été membre de l'équipe associée à la prise en charge de l'acte de soin.

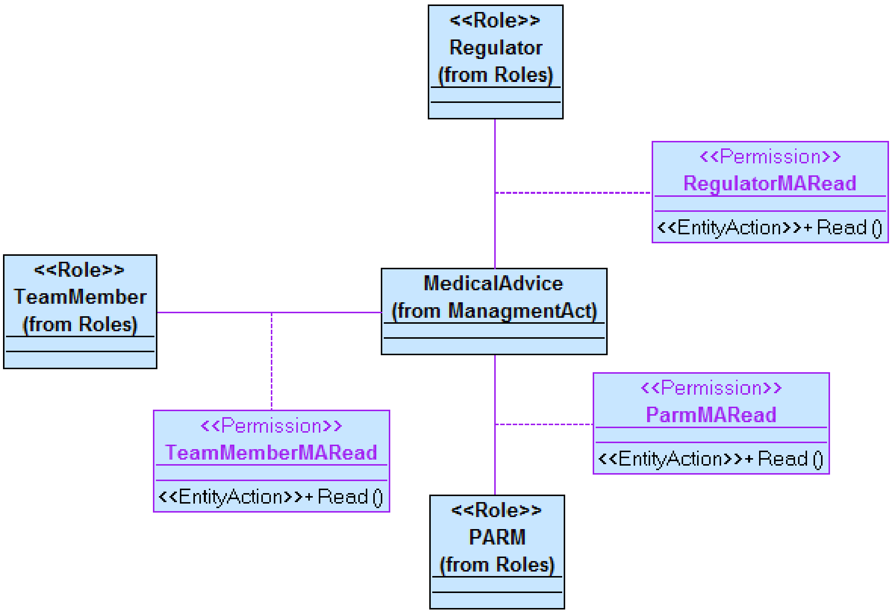

FIGURE 2.7 – Règles de lecture de MedicalAdvice

2.3 Spécifications B

La plateforme B4MSecure que nous avons développée en vue de produire les spécifications B à partir des modèles graphiques a nécessité plusieurs ajustements permettant de bien répondre aux spécificités de l'étude de cas Res@mu. Ces ajustements n'ont pas été très complexes à mettre en place, car la plateforme a été conçue de telle sorte qu'on puisse localiser aisément les extensions qu'on souhaite apporter aux différentes règles de transformation.

Les spécifications B générées se résument par :
- Modèle fonctionnel : 1730 lignes de code
- Modèle de sécurité : 2652 lignes de code

Par ailleurs, les activités de validation ont été réalisées principalement par animation. Nous avons pour ce faire élaboré des machines B additionnelles, reflétant chacune un scénario bien déterminé. Cette activité de validation est détaillée dans le chapitre suivant.

Chapitre 3

Validation de la politique de contrôle d'accès

Le modèle PIM de l'application Resamu consiste en des diagrammes SecureUML qui ont été traduits en spécifications formelles B. Ces spécifications formelles peuvent être animées en utilisant l'outil ProB. C'est cet outil qui a été utilisé dans une activité de test du modèle. Cette activité de test s'est déroulée en trois étapes :
– test à base de cas d'utilisation,
– test systématique des règles de sécurité du modèle SecureUML,
– recherche de scénarios d'attaque.

3.1 Test à base de cas d'utilisation

Nous avons sollicité dans le cadre du livrable 6.1.2 des cas d'utilisation en vue d'aboutir à un diagramme de classes traduisible en B et dont l'animation soit proche de l'exécution réelle de l'application. Nous avons ainsi étudié cinq scénarios majeurs relatifs au processus de prise en charge d'un patient. Notre objectif est de montrer que ces cas d'utilisation peuvent être joués sur le modèle. Les cinq cas d'utilisation étaient les suivants :

1. démarrer une prise en charge,
2. valider une prise en charge,
3. invalider une prise en charge,
4. enregistrer une prise en charge,
5. terminer une prise en charge.

En fait, ces cas d'utilisation s'enchaînent et il est possible de les regrouper dans deux tests. Ils ont été joués avec succès sur le modèle sécurisé.
Ces tests donnent un élément positif de validation, mais qui reste relativement limité. En effet, ces tests montrent que ces cas d'utilisation sont possibles et autorisés. Ils ne disent pas si la politique de contrôle d'accès interdit certaines actions et empêche les attaques. La phase suivante permet d'explorer plus systématiquement les aspects sécuritaires de l'application.

3.2 Test systématique des règles de sécurité

Dans cette deuxième phase, chacune des règles de sécurité du diagramme SecureUML a été testée. Les dix règles concernées apparaissent dans la première colonne de la table 3.1. Conformément à la méthode proposée au chapitre 2 du livrable 3.3. Pour chaque règle de sécurité, on définit des tests positifs et des tests négatifs. Les tests positifs sont destinés à exercer le comportement normal de l'application. Ils sont destinés à se terminer par un succès. Les tests négatifs testent la réaction du système à des comportements interdits. Ils se terminent généralement par un échec résultant de l'interdiction d'accéder à une des opérations.

Ces tests sont des tests de sécurité. L'objectif est de tester les permissions et les interdictions, et pas de vérifier que le résultat fonctionnel est correct. Par exemple, quand un test consiste en une lecture d'un objet, on vérifiera simplement que la lecture a eu lieu, mais pas qu'elle a produit la bonne valeur.

Les interdictions résultent de trois cas de figure :

– l'appel d'une opération sécurisée dans un rôle qui n'a pas accès à cette opération,
– l'appel d'une opération sécurisée dans un rôle autorisé, mais en dehors d'une contrainte d'autorisation,
– l'appel d'une opération sécurisée dans un rôle autorisé, en remplissant les contraintes d'autorisation, mais en dehors de la précondition de l'opération.

Par exemple, pour appeler l'opération de création d'un valid medical advice, il faut être dans un rôle qui le permet (PARM), il faut passer en paramètre l'auteur du medical advice qui doit être membre de l'équipe d'intervention (précondition fonctionnelle), et l'utilisateur qui crée cet advice doit être celui passé en paramètre comme auteur.

Les deux derniers cas sont relativement semblables car ils dépendent des paramètres d'entrée et de l'état fonctionnel de l'application (l'état de l'équipe d'intervention).

Tests positifs Le but des tests positifs est de vérifier qu'un utilisateur peut utiliser les opérations concernées par une permission si il est un rôle autorisé et qu'il satisfait la précondition et la contrainte d'utilisation. Formellement, il suffit d'un test positif pour cette vérification, cependant quand des contraintes d'autorisation s'appliquent à certaines entity Actions mais pas à d'autres, nous avons multiplié les tests de ces entityActions concernées par la même permission, et dans certains cas, nous avons essayé quelques variations autour du cas positif (par exemple, déclarer un management act réalisé par un autre acteur).

34 tests positifs ont été écrits (Fig. 3.1). Ils établissent la disponibilité de ces opérations pour les rôles concernés.

Tests négatifs Les tests négatifs sont les plus proches possibles des tests positifs, mais ils invalident l'un des éléments de la permission (rôle, précondition ou contrainte d'autorisation). Ceci permet de vérifier que chacune de ces interdictions fonctionne. Contrairement au test positif, le test négatif ne peut pas s'exécuter entièrement.

87 tests négatifs ont été écrits (Fig. 3.1). Ils établissent les propriétés de confidentialité et d'intégrité attendues en empêchant de violer ces propriétés pour des personnes non autorisées.

Conclusion sur les tests de sécurité Au total 121 tests ont été définis. Ils permettent de vérifier de façon détaillée que les règles de sécurité sont effectives.

Ces tests garantissent un minimum de sécurité. D'une part, les utilisateurs dont le rôle ne permet pas d'accéder à une opération ne pourront pas le faire. D'autre part, cela garantit la disponibilité des opérations dans les cas normaux. Cependant, nous avons vu lors du projet Selkis que les contraintes d'autorisation pouvaient donner lieu à des scénarios d'attaque, en faisant évoluer l'état du système d'information vers un état où la contrainte d'autorisation est vraie. C'est ce qui sera examiné dans la section suivante.

Les tests de sécurité ne sont cependant pas exhaustifs. Ils se contentent de couvrir l'ensemble

Nom de la règle	Nb tests positifs	Nb tests négatifs
Intervention_Perms	3	7
Patient_Perms	3	7
Team_Perms	5	10
Management_Perms	3	6
TeamDoctorMA	4	12
TeamMemberMA	5	14
ParmAdviceMA	4	15
RegulatorInstructionMA	3	12
RegParmMAPerm	2	2
TeamMemberMAPerm	2	2
Total	34	87

Table 3.1 – Règles testées et tests associés.

des entity actions (par exemple create() ou modify()), mais ne couvrent pas dans le détail toutes les instanciations de ces actions. Une campagne de tests plus complète devrait couvrir toutes les méthodes des classes sécurisées et tous les rôles possibles. Ceci mène à une explosion combinatoire du nombre de tests à réaliser, qui prendrait beaucoup de temps pour exécuter ces tests sur le modèle B (l'exécution des tests est actuellement en partie manuelle). Nous pensons cependant que l'échantillonnage des rôles et des méthodes est suffisant pour offrir une garantie satisfaisante. Par contre, il serait intéressant d'étudier la possibilité d'effectuer ces tests plus complets sur l'implémentation du système Resamu, en utilisant le modèle B comme oracle du test.

Ces tests de sécurité ont permis de trouver plusieurs défauts dans le modèle. Ces défauts ont été corrigés et tous les tests ont été rejoués sur la version corrigée.

3.3 Recherche de scénarios d'attaque

La troisième phase de la campagne de test s'intéresse à l'identification de scénarios d'attaque qui exploitent les failles des contraintes d'autorisation.

Le principe d'un scénario d'attaque consiste à faire évoluer l'état fonctionnel du système pour qu'une personne non autorisée soit finalement autorisée à effectuer une action qui compromet l'un des objectifs de sécurité.

Dans notre cas, nous nous concentrons sur l'intégrité et la confidentialité des Management Acts. Un scénario d'attaque est un scénario qui conduit une personne non autorisée à lire ou à modifier un management act auquel elle ne devrait pas avoir accès, voire à créer un management act fictif. Dans sa version la plus simple, c'est l'attaquant qui fait évoluer l'état du système pour se donner un accès au management act. Une version plus complexe fait intervenir une collusion entre plusieurs utilisateurs pour accéder à cette information.

Il faut cependant distinguer les utilisateurs autorisés des autres utilisateurs. Pour ceux qui sont autorisés, on fera l'hypothèse qu'ils n'abusent pas de leurs droits.

Il faut noter que plusieurs éléments temporels rythment une intervention :
- l'ouverture et la fermeture d'un management
- l'ouverture et la fermeture d'une intervention
- la validation d'un acte de soin (qui est définitive).

Quand un management est fermé, il n'est plus possible de modifier les management acts qui y sont rattachés, par contre, il est toujours possible de les lire pour les membres de l'équipe associée.

Nous proposons donc les scénarios d'attaque suivants :

1. Pendant une intervention, un personnel non membre de l'équipe associée à l'intervention tente (a) d'accéder aux données de l'intervention, (b) de participer à l'équipe d'intervention puis d'accéder aux données de l'intervention. On peut exécuter ces actions dans le rôle TeamDoctor et dans le rôle DrRegulator, qui sont les rôles qui devraient ouvrir le plus de permissions.
2. A la fin d'une intervention, un personnel qui n'a pas participé à l'intervention tente d'accéder aux informations de l'intervention, notamment en se rajoutant dans l'équipe d'intervention.
3. A la fin d'une intervention, est-il possible de remonter dans le temps en utilisant une heure différente de l'heure réelle ?
4. Comme la composition d'une équipe est modifiée par le responsable de l'intervention, est-il possible pour un attaquant de devenir responsable d'une intervention en cours, ou terminée ?

Comme pour les tests de sécurité, nous avons écrit quelques tests positifs pour montrer que ces scénarios fonctionneraient si ils étaient exécutés par des personnes habilitées.

Au total, 9 tests positifs et 13 tests négatifs ont été écrits. L'ensemble de ces scénarios d'attaque ont été exécutés avec succès sur la dernière version du modèle de sécurité.

Conclusion sur les scénarios d'attaque Les scénarios d'attaque présentés ci-dessus se sont focalisés sur l'atteinte à la confidentialité des informations. On peut imaginer d'autres scénarios visant l'intégrité des informations liées à une intervention. Cependant, les permissions associées aux contraintes d'intégrité sont souvent les mêmes que celles associées à la confidentialité, voire sont plus restrictives. Il est donc probable que ces nouveaux scénarios d'attaque ne révéléront pas ou peu de défauts dans le modèle de sécurité.

Chapitre 4

Conclusion

Dans ce livrable nous avons présenté la représentation graphique de la politique de contrôle d'accès à un sous-ensemble de l'application Res@mu. Les règles de cette politique de sécurité sont associées à diverses contraintes d'autorisation que nous avons présenté informellement. La politique de sécurité et les contraintes d'autorisation ont été traduites en un modèle formel écrit dans le langage B.

Dans un deuxième temps, nous avons validé ce modèle de sécurité en l'animant avec l'outil ProB. Les tests menés dans cette activité d'animation ont visé tout d'abord à vérifier que le modèle permettait d'exécuter 5 cas d'utilisation, fournis par les concepteurs de Res@mu, ensuite à exercer chaque permission du modèle de sécurité en tant qu'autorisation ou interdiction, et enfin à contrôler que le modèle est résistant à quelques scénarios d'attaque, qui tentent de faire évoluer l'état du modèle fonctionnel pour changer faire basculer l'état de contraintes d'autorisation.

Cette validation du modèle n'est certes pas parfaite. On peut la compléter en exerçant systématiquement chaque rôle et chaque opération dans divers états du modèle fonctionnel. On peut également rechercher de nouveaux scénarios d'attaque. Cependant, les tests réalisés couvrent toutes les permissions et diverses variantes des scénarios d'attaque envisagés lors de l'analyse des besoins. Ces tests ont permis de trouver un nombre significatif de défauts dans le modèle de sécurité et de vérifier qu'ils étaient ensuite corrigés. Il nous semble que la politique de sécurité est suffisamment validée pour envisager son implémentation.

Annexe A

Specifications B issues du modèle de sécurité

A.1 Partie statique

```
MACHINE
    RBAC_Model
INCLUDES
    ManagmentAct, /* this is the functional model */
    UserAssignments
SEES
ContextMachine
SETS
    ENTITIES = {
        MedicalAdvice_Label,Patient_Label,
        ManagementAct_Label,Care_Label,
        Intervention_Label,Diagnosis_Label,
        Management_Label,PreEstablishedTeam_Label,
        Person_Label,PreHospitalActor_Label,Team_Label} ;
    Attributes = {  MedicalAdvice_advice_Label,Patient_closed_Label,
                    Patient_data_Label,Patient_name_Label,
                    ManagementAct_dateTime_Label,ManagementAct_validated_Label,
                    ManagementAct_validationDateTime_Label,
                    ManagementAct_invalidationDateTime_Label,
                    ManagementAct_invalidationReason_Label,Care_data_Label,
                    Intervention_startingDateTime_Label,
                    Intervention_endingDateTime_Label,Intervention_closed_Label,
                    Diagnosis_wording_Label,Management_startingDateTime_Label,
                    Management_endingDateTime_Label,
                    Management_endingPatientState_Label,PreEstablishedTeam_name_Label,
                    Person_name_Label,
                    PreHospitalActor_endingDateTime_Label,
                    PreHospitalActor_startingDateTime_Label,
                    PreHospitalActor_state_Label,PreHospitalActor_operator_Label,
                    PreHospitalActor_medicalRole_Label,
                    Team_creationDateTime_Label,Team_endOfMissionDateTime_Label,Team_name_Label
    };

    Operations = {  Diagnosis__validate_Label
                    ,ManagementAct__validate_Label
                    ,ManagementAct__invalidate_Label
                    ,MedicalAdvice__NEW_ValidAdvice_Label
```

```
,MedicalAdvice__validate_Label
,Management__endManagment_Label
,Team__addMembers_Label
,Team__removeMembers_Label
,Team__end_Label
,Team__CreateTeamFromprestablishedOne_Label
,Intervention__Close_Label
,Care__validate_Label
,PreHospitalActor_NEW_Label
,Patient_NEW_Label
,ManagementAct_NEW_Label
,Management_NEW_Label
,Team_NEW_Label
,TeamMemberChangement_NEW_Label
,Intervention_NEW_Label
,PreEstablishedTeam_NEW_Label
,Person_NEW_Label
,Diagnosis_NEW_Label
,MedicalAdvice_NEW_Label
,Care_NEW_Label
,PreHospitalActor_Free_Label
,Patient_Free_Label
,ManagementAct_Free_Label
,Management_Free_Label
,Team_Free_Label
,TeamMemberChangement_Free_Label
,Intervention_Free_Label
,PreEstablishedTeam_Free_Label
,Person_Free_Label
,Diagnosis_Free_Label
,MedicalAdvice_Free_Label
,Care_Free_Label
,Intervention__GetA_Intervention_Team_Label
,Team__GetA_Intervention_Team_Label
,Intervention__GetA_Intervention_Patient_Label
,Patient__GetA_Intervention_Patient_Label
,Intervention__GetA_Management_Intervention_Label
,Management__GetA_Management_Intervention_Label
,Management__GetA_Management_ManagementAct_Label
,ManagementAct__GetA_Management_ManagementAct_Label
,Management__GetA_Patient_Management_Label
,Patient__GetA_Patient_Management_Label
,ManagementAct__GetA_PreHospitalActor_ManagementAct_Label
,PreHospitalActor__GetA_PreHospitalActor_ManagementAct_Label
,TeamMemberChangement__GetA_PreHospitalActor_TeamMemberChangement_Label
,Team__GetA_Team_TeamMemberChangement_Label
,TeamMemberChangement__GetA_Team_TeamMemberChangement_Label
,Management__GetA_Team_Management_Label
,Team__GetA_Team_Management_Label
,Team__GetA_team_AllPreHospitalActor_Label
,PreEstablishedTeam__GetA_preEstablishedTeam_preHospitalActor_Label
,Person__GetA_preHospitalActor_person_Label
,PreHospitalActor__GetA_preHospitalActor_person_Label
,Team__GetA_Team_PreHospitalActor_Label
,PreHospitalActor__GetA_Team_PreHospitalActor_Label
,PreHospitalActor__GetInterventionResponsible_Label
,Intervention__GetInterventionResponsible_Label
,Patient__GetPatientResponsible_Label
```

```
                    ,PreHospitalActor__GetPatientResponsible_Label
                    ,PreHospitalActor__SetA_preHospitalActor_person_Label
                    ,Intervention__SetInterventionResponsible_Label
                    ,Patient__SetPatientResponsible_Label
                    ,Intervention__AddA_Intervention_Team_Label
                    ,Intervention__AddA_Intervention_Patient_Label
                    ,PreEstablishedTeam__AddA_preEstablishedTeam_preHospitalActor_Label
                    ,Person__AddA_preHospitalActor_person_Label
                    ,PreHospitalActor__AddInterventionResponsible_Label
                    ,PreHospitalActor__AddPatientResponsible_Label
                    ,Intervention__RemoveA_Intervention_Team_Label
                    ,Intervention__RemoveA_Intervention_Patient_Label
                    ,PreEstablishedTeam__RemoveA_preEstablishedTeam_preHospitalActor_Label
                    ,Person__RemoveA_preHospitalActor_person_Label
                    ,PreHospitalActor__RemoveInterventionResponsible_Label
                    ,PreHospitalActor__RemovePatientResponsible_Label
                    ,PreHospitalActor__GetEndingDateTime_Label
                    ,PreHospitalActor__GetStartingDateTime_Label
                    ,PreHospitalActor__GetState_Label
                    ,PreHospitalActor__GetOperator_Label
                    ,PreHospitalActor__GetMedicalRole_Label
                    ,Patient__GetClosed_Label
                    ,Patient__GetData_Label
                    ,Patient__GetName_Label
                    ,Diagnosis__GetWording_Label
                    ,ManagementAct__GetDateTime_Label
                    ,ManagementAct__GetValidated_Label
                    ,ManagementAct__GetValidationDateTime_Label
                    ,ManagementAct__GetInvalidationDateTime_Label
                    ,ManagementAct__GetInvalidationReason_Label
                    ,MedicalAdvice__GetAdvice_Label
                    ,Management__GetStartingDateTime_Label
                    ,Management__GetEndingDateTime_Label
                    ,Management__GetEndingPatientState_Label
                    ,Team__GetCreationDateTime_Label
                    ,Team__GetEndOfMissionDateTime_Label
                    ,Team__GetName_Label
                    ,TeamMemberChangement__GetChangement_Label
                    ,TeamMemberChangement__GetDateTime_Label
                    ,Intervention__GetStartingDateTime_Label
                    ,Intervention__GetEndingDateTime_Label
                    ,Intervention__GetClosed_Label
                    ,Care__GetData_Label
                    ,PreEstablishedTeam__GetName_Label
                    ,Person__GetName_Label
                    ,PreHospitalActor__SetEndingDateTime_Label
                    ,PreHospitalActor__SetState_Label
                    ,Diagnosis__SetWording_Label
                    ,Patient__SetData_Label
                    ,Patient__SetName_Label
                    ,Team__SetName_Label
                    ,Care__SetData_Label
                    ,PreEstablishedTeam__SetName_Label
                    ,Person__SetName_Label};
KindsOfAtt = {public, private};
PERMISSIONS = { RegulatorMARead,Patient_Perms,RegParmMAPerm,
                RegulatorCareRead,Intervention_Perms,
                ParmDiagnosisRead,ParmAdviceMA,
```

```
                TeamMemberMARead,Management_Perms,
                AdminPreTeamPerm,AdminPersonPerm,
                AdminPreActorPerm,TeamMemberMAPerm,
                TeamMemberDiagnosisRead,TeamMemberMA,
                RegulatorInstructionMA,Team_Perms,
                TeamDoctorMA,ParmCareRead,
                RegulatorDiagnosisRead,
                ParmMARead,TeamMemberCareRead};
ActionsType = {read, create, modify, delete, privateRead, privateModify,fullAccess};
Stereotypes = {readOp, modifyOp}
;
AssociationRole = {not_yet_implemented}
VARIABLES
    AttributeKind, AttributeOf, OperationOf,
    constructorOf, destructorOf, setterOf, getterOf,
    AssoRoleOf,AssoRoleAccessorOf,AssoRoleMutatorOf,
    PermissionAssignement,
    EntityActions,
MethodActions,
    StereotypeOps,
    isPermitted
INVARIANT
    /*Function model Elements*/
AttributeKind : Attributes --> KindsOfAtt &
AttributeOf : Attributes --> ENTITIES &
OperationOf : Operations --> ENTITIES &
constructorOf : Operations +-> ENTITIES &
destructorOf : Operations +-> ENTITIES &
setterOf : Operations +-> Attributes &
getterOf : Operations +-> Attributes &
StereotypeOps :  Stereotypes <-> Operations &

setterOf /\ getterOf = {} &

/* ClassRoles */
AssoRoleOf : AssociationRole +-> ENTITIES &
AssoRoleAccessorOf : Operations <-> AssociationRole &
AssoRoleMutatorOf : Operations <-> AssociationRole &

/* Access Control Elements */
PermissionAssignement : PERMISSIONS --> (ROLES * ENTITIES) &
EntityActions : PERMISSIONS +-> POW(ActionsType) &
MethodActions : PERMISSIONS +-> POW(Operations) &

isPermitted : ROLES <-> Operations
```

A.2 Initialisation

```
INITIALISATION
    AttributeKind := {  (MedicalAdvice_advice_Label|->public),
                        (Patient_closed_Label|->public),
                        (Patient_data_Label|->public),
                        (Patient_name_Label|->public),
                        (ManagementAct_dateTime_Label|->private),
                        (ManagementAct_validated_Label|->private),
                        (ManagementAct_validationDateTime_Label|->private),
```

```
                    (ManagementAct_invalidationDateTime_Label|->private),
                    (ManagementAct_invalidationReason_Label|->private),
                    (Care_data_Label|->public),
                    (Intervention_startingDateTime_Label|->public),
                    (Intervention_endingDateTime_Label|->public),
                    (Intervention_closed_Label|->public),
                    (Diagnosis_wording_Label|->public),
                    (Management_startingDateTime_Label|->public),
                    (Management_endingDateTime_Label|->public),
                    (Management_endingPatientState_Label|->public),
                    (PreEstablishedTeam_name_Label|->public),
                    (Person_name_Label|->public),
                    (PreHospitalActor_endingDateTime_Label|->public),
                    (PreHospitalActor_startingDateTime_Label|->private),
                    (PreHospitalActor_state_Label|->public),
                    (PreHospitalActor_operator_Label|->private),
                    (PreHospitalActor_medicalRole_Label|->private),
                    (Team_creationDateTime_Label|->public),
                    (Team_endOfMissionDateTime_Label|->public),
                    (Team_name_Label|->public)}
||
AttributeOf := {(MedicalAdvice_advice_Label|->MedicalAdvice_Label),
                (Patient_closed_Label|->Patient_Label),
                (Patient_data_Label|->Patient_Label),
                (Patient_name_Label|->Patient_Label),
                (ManagementAct_dateTime_Label|->ManagementAct_Label),
                (ManagementAct_validated_Label|->ManagementAct_Label),
                (ManagementAct_validationDateTime_Label|->ManagementAct_Label),
                (ManagementAct_invalidationDateTime_Label|->ManagementAct_Label),
                (ManagementAct_invalidationReason_Label|->ManagementAct_Label),
                (Care_data_Label|->Care_Label),
                (Intervention_startingDateTime_Label|->Intervention_Label),
                (Intervention_endingDateTime_Label|->Intervention_Label),
                (Intervention_closed_Label|->Intervention_Label),
                (Diagnosis_wording_Label|->Diagnosis_Label),
                (Management_startingDateTime_Label|->Management_Label),
                (Management_endingDateTime_Label|->Management_Label),
                (Management_endingPatientState_Label|->Management_Label),
                (PreEstablishedTeam_name_Label|->PreEstablishedTeam_Label),
                (Person_name_Label|->Person_Label),
                (PreHospitalActor_endingDateTime_Label|->PreHospitalActor_Label),
                (PreHospitalActor_startingDateTime_Label|->PreHospitalActor_Label),
                (PreHospitalActor_state_Label|->PreHospitalActor_Label),
                (PreHospitalActor_operator_Label|->PreHospitalActor_Label),
                (PreHospitalActor_medicalRole_Label|->PreHospitalActor_Label),
                (Team_creationDateTime_Label|->Team_Label),
                (Team_endOfMissionDateTime_Label|->Team_Label),
                (Team_name_Label|->Team_Label)}
||

OperationOf := {(Diagnosis__validate_Label|->Diagnosis_Label)
,(ManagementAct__validate_Label|->ManagementAct_Label)
,(ManagementAct__invalidate_Label|->ManagementAct_Label)
,(MedicalAdvice__NEW_ValidAdvice_Label|->MedicalAdvice_Label)
,(MedicalAdvice__validate_Label|->MedicalAdvice_Label)
,(Management__endManagment_Label|->Management_Label)
,(Team__addMembers_Label|->Team_Label)
,(Team__removeMembers_Label|->Team_Label)
```

```
,(Team__end_Label|->Team_Label)
,(Team__CreateTeamFromprestablishedOne_Label|->Team_Label)
,(Intervention__Close_Label|->Intervention_Label)
,(Care__validate_Label|->Care_Label)
,(PreHospitalActor_NEW_Label|->PreHospitalActor_Label)
,(Patient_NEW_Label|->Patient_Label)
,(ManagementAct_NEW_Label|->ManagementAct_Label)
,(Management_NEW_Label|->Management_Label)
,(Team_NEW_Label|->Team_Label)
,(TeamMemberChangement_NEW_Label|->Team_Label)
,(Intervention_NEW_Label|->Intervention_Label)
,(PreEstablishedTeam_NEW_Label|->PreEstablishedTeam_Label)
,(Person_NEW_Label|->Person_Label)
,(Diagnosis_NEW_Label|->Diagnosis_Label)
,(MedicalAdvice_NEW_Label|->MedicalAdvice_Label)
,(Care_NEW_Label|->Care_Label)
,(PreHospitalActor_Free_Label|->PreHospitalActor_Label)
,(Patient_Free_Label|->Patient_Label)
,(ManagementAct_Free_Label|->ManagementAct_Label)
,(Management_Free_Label|->Management_Label)
,(Team_Free_Label|->Team_Label)
,(TeamMemberChangement_Free_Label|->Team_Label)
,(Intervention_Free_Label|->Intervention_Label)
,(PreEstablishedTeam_Free_Label|->PreEstablishedTeam_Label)
,(Person_Free_Label|->Person_Label)
,(Diagnosis_Free_Label|->Diagnosis_Label)
,(MedicalAdvice_Free_Label|->MedicalAdvice_Label)
,(Care_Free_Label|->Care_Label)
,(Intervention__GetA_Intervention_Team_Label|->Intervention_Label)
,(Team__GetA_Intervention_Team_Label|->Team_Label)
,(Intervention__GetA_Intervention_Patient_Label|->Intervention_Label)
,(Patient__GetA_Intervention_Patient_Label|->Patient_Label)
,(Intervention__GetA_Management_Intervention_Label|->Intervention_Label)
,(Management__GetA_Management_Intervention_Label|->Management_Label)
,(Management__GetA_Management_ManagementAct_Label|->Management_Label)
,(ManagementAct__GetA_Management_ManagementAct_Label|->ManagementAct_Label)
,(Management__GetA_Patient_Management_Label|->Management_Label)
,(Patient__GetA_Patient_Management_Label|->Patient_Label)
,(ManagementAct__GetA_PreHospitalActor_ManagementAct_Label|->ManagementAct_Label)
,(PreHospitalActor__GetA_PreHospitalActor_ManagementAct_Label|->PreHospitalActor_Label)
,(TeamMemberChangement__GetA_PreHospitalActor_TeamMemberChangement_Label|->Team_Label)
,(Team__GetA_Team_TeamMemberChangement_Label|->Team_Label)
,(TeamMemberChangement__GetA_Team_TeamMemberChangement_Label|->Team_Label)
,(Management__GetA_Team_Management_Label|->Management_Label)
,(Team__GetA_Team_Management_Label|->Team_Label)
,(Team__GetA_team_AllPreHospitalActor_Label|->Team_Label)
,(PreEstablishedTeam__GetA_preEstablishedTeam_preHospitalActor_Label|->PreEstablishedTeam_Label
,(Person__GetA_preHospitalActor_person_Label|->Person_Label)
,(PreHospitalActor__GetA_preHospitalActor_person_Label|->PreHospitalActor_Label)
,(Team__GetA_Team_PreHospitalActor_Label|->Team_Label)
,(PreHospitalActor__GetA_Team_PreHospitalActor_Label|->PreHospitalActor_Label)
,(PreHospitalActor__GetInterventionResponsible_Label|->PreHospitalActor_Label)
,(Intervention__GetInterventionResponsible_Label|->Intervention_Label)
,(Patient__GetPatientResponsible_Label|->Patient_Label)
,(PreHospitalActor__GetPatientResponsible_Label|->PreHospitalActor_Label)
,(PreHospitalActor__SetA_preHospitalActor_person_Label|->PreHospitalActor_Label)
,(Intervention__SetInterventionResponsible_Label|->Intervention_Label)
,(Patient__SetPatientResponsible_Label|->Patient_Label)
```

```
,(Intervention__AddA_Intervention_Team_Label|->Intervention_Label)
,(Intervention__AddA_Intervention_Patient_Label|->Intervention_Label)
,(PreEstablishedTeam__AddA_preEstablishedTeam_preHospitalActor_Label|->PreEstablishedTeam_Label
,(Person__AddA_preHospitalActor_person_Label|->Person_Label)
,(PreHospitalActor__AddInterventionResponsible_Label|->PreHospitalActor_Label)
,(PreHospitalActor__AddPatientResponsible_Label|->PreHospitalActor_Label)
,(Intervention__RemoveA_Intervention_Team_Label|->Intervention_Label)
,(Intervention__RemoveA_Intervention_Patient_Label|->Intervention_Label)
,(PreEstablishedTeam__RemoveA_preEstablishedTeam_preHospitalActor_Label|->PreEstablishedTeam_La
,(Person__RemoveA_preHospitalActor_person_Label|->Person_Label)
,(PreHospitalActor__RemoveInterventionResponsible_Label|->PreHospitalActor_Label)
,(PreHospitalActor__RemovePatientResponsible_Label|->PreHospitalActor_Label)
,(PreHospitalActor__GetEndingDateTime_Label|->PreHospitalActor_Label)
,(PreHospitalActor__GetStartingDateTime_Label|->PreHospitalActor_Label)
,(PreHospitalActor__GetState_Label|->PreHospitalActor_Label)
,(PreHospitalActor__GetOperator_Label|->PreHospitalActor_Label)
,(PreHospitalActor__GetMedicalRole_Label|->PreHospitalActor_Label)
,(Patient__GetClosed_Label|->Patient_Label)
,(Patient__GetData_Label|->Patient_Label)
,(Patient__GetName_Label|->Patient_Label)
,(Diagnosis__GetWording_Label|->Diagnosis_Label)
,(ManagementAct__GetDateTime_Label|->ManagementAct_Label)
,(ManagementAct__GetValidated_Label|->ManagementAct_Label)
,(ManagementAct__GetValidationDateTime_Label|->ManagementAct_Label)
,(ManagementAct__GetInvalidationDateTime_Label|->ManagementAct_Label)
,(ManagementAct__GetInvalidationReason_Label|->ManagementAct_Label)
,(MedicalAdvice__GetAdvice_Label|->MedicalAdvice_Label)
,(Management__GetStartingDateTime_Label|->Management_Label)
,(Management__GetEndingDateTime_Label|->Management_Label)
,(Management__GetEndingPatientState_Label|->Management_Label)
,(Team__GetCreationDateTime_Label|->Team_Label)
,(Team__GetEndOfMissionDateTime_Label|->Team_Label)
,(Team__GetName_Label|->Team_Label)
,(TeamMemberChangement__GetChangement_Label|->Team_Label)
,(TeamMemberChangement__GetDateTime_Label|->Team_Label)
,(Intervention__GetStartingDateTime_Label|->Intervention_Label)
,(Intervention__GetEndingDateTime_Label|->Intervention_Label)
,(Intervention__GetClosed_Label|->Intervention_Label)
,(Care__GetData_Label|->Care_Label)
,(PreEstablishedTeam__GetName_Label|->PreEstablishedTeam_Label)
,(Person__GetName_Label|->Person_Label)
,(PreHospitalActor__SetEndingDateTime_Label|->PreHospitalActor_Label)
,(PreHospitalActor__SetState_Label|->PreHospitalActor_Label)
,(Diagnosis__SetWording_Label|->Diagnosis_Label)
,(Patient__SetData_Label|->Patient_Label)
,(Patient__SetName_Label|->Patient_Label)
,(Team__SetName_Label|->Team_Label)
,(Care__SetData_Label|->Care_Label)
,(PreEstablishedTeam__SetName_Label|->PreEstablishedTeam_Label)
,(Person__SetName_Label|->Person_Label)}
||
constructorOf := {  (MedicalAdvice__NEW_ValidAdvice_Label|->MedicalAdvice_Label),
                    (PreHospitalActor_NEW_Label|->PreHospitalActor_Label),
                    (Patient_NEW_Label|->Patient_Label),
                    (ManagementAct_NEW_Label|->ManagementAct_Label),
                    (Management_NEW_Label|->Management_Label),
                    (Team_NEW_Label|->Team_Label),
                    (TeamMemberChangement_NEW_Label|->Team_Label),
```

```
                      (Intervention_NEW_Label|->Intervention_Label),
                      (PreEstablishedTeam_NEW_Label|->PreEstablishedTeam_Label),
                      (Person_NEW_Label|->Person_Label),
                      (Diagnosis_NEW_Label|->Diagnosis_Label),
                      (MedicalAdvice_NEW_Label|->MedicalAdvice_Label),
                      (Care_NEW_Label|->Care_Label)}
  ||
  destructorOf := {   (PreHospitalActor_Free_Label|->PreHospitalActor_Label),
                      (Patient_Free_Label|->Patient_Label),
                      (ManagementAct_Free_Label|->ManagementAct_Label),
                      (Management_Free_Label|->Management_Label),
                      (Team_Free_Label|->Team_Label),
                      (TeamMemberChangement_Free_Label|->Team_Label),
                      (Intervention_Free_Label|->Intervention_Label),
                      (PreEstablishedTeam_Free_Label|->PreEstablishedTeam_Label),
                      (Person_Free_Label|->Person_Label),
                      (Diagnosis_Free_Label|->Diagnosis_Label),
                      (MedicalAdvice_Free_Label|->MedicalAdvice_Label),
                      (Care_Free_Label|->Care_Label)}
  ||
  StereotypeOps := {  (modifyOp|->Diagnosis__validate_Label)
                      ,(modifyOp|->ManagementAct__validate_Label)
                      ,(modifyOp|->ManagementAct__invalidate_Label)
                      ,(modifyOp|->MedicalAdvice__validate_Label)
                      ,(modifyOp|->Management__endManagment_Label)
                      ,(modifyOp|->Team__addMembers_Label)
                      ,(modifyOp|->Team__removeMembers_Label)
                      ,(modifyOp|->Team__end_Label)
                      ,(modifyOp|->Team__CreateTeamFromprestablishedOne_Label)
                      ,(modifyOp|->Intervention__Close_Label)
                      ,(modifyOp|->Care__validate_Label)}
\/ {(modifyOp|->Intervention__AddA_Intervention_Patient_Label),
   (modifyOp|->Intervention__SetInterventionResponsible_Label)
   }
  ||
  setterOf := {(PreHospitalActor__SetEndingDateTime_Label|->PreHospitalActor_endingDateTime_Label
              (PreHospitalActor__SetState_Label|->PreHospitalActor_state_Label),
              (Diagnosis__SetWording_Label|->Diagnosis_wording_Label),
              (Patient__SetData_Label|->Patient_data_Label),
              (Patient__SetName_Label|->Patient_name_Label),
              (Team__SetName_Label|->Team_name_Label),
              (Care__SetData_Label|->Care_data_Label),
              (PreEstablishedTeam__SetName_Label|->PreEstablishedTeam_name_Label),
              (Person__SetName_Label|->Person_name_Label)}
  ||
  getterOf := {(PreHospitalActor__GetEndingDateTime_Label|->PreHospitalActor_endingDateTime_Label
              (PreHospitalActor__GetStartingDateTime_Label|->PreHospitalActor_startingDateTime_La
              (PreHospitalActor__GetState_Label|->PreHospitalActor_state_Label),
              (PreHospitalActor__GetOperator_Label|->PreHospitalActor_operator_Label),
              (PreHospitalActor__GetMedicalRole_Label|->PreHospitalActor_medicalRole_Label),
              (Patient__GetClosed_Label|->Patient_closed_Label),
              (Patient__GetData_Label|->Patient_data_Label),
              (Patient__GetName_Label|->Patient_name_Label),
              (Diagnosis__GetWording_Label|->Diagnosis_wording_Label),
              (ManagementAct__GetDateTime_Label|->ManagementAct_dateTime_Label),
              (ManagementAct__GetValidated_Label|->ManagementAct_validated_Label),
              (ManagementAct__GetValidationDateTime_Label|->ManagementAct_dateTime_Label),
              (ManagementAct__GetInvalidationDateTime_Label|->ManagementAct_dateTime_Label),
```

```
            (ManagementAct__GetInvalidationReason_Label|->ManagementAct_invalidationReason_Labe
            (MedicalAdvice__GetAdvice_Label|->MedicalAdvice_advice_Label),
            (Management__GetStartingDateTime_Label|->Management_startingDateTime_Label),
            (Management__GetEndingDateTime_Label|->Management_endingDateTime_Label),
            (Management__GetEndingPatientState_Label|->Management_endingPatientState_Label),
            (Team__GetCreationDateTime_Label|->Team_creationDateTime_Label),
            (Team__GetEndOfMissionDateTime_Label|->Team_endOfMissionDateTime_Label),
            (Team__GetName_Label|->Team_name_Label),
            (Intervention__GetStartingDateTime_Label|->Intervention_startingDateTime_Label),
            (Intervention__GetEndingDateTime_Label|->Intervention_endingDateTime_Label),
            (Intervention__GetClosed_Label|->Intervention_closed_Label),
            (Care__GetData_Label|->Care_data_Label),
            (PreEstablishedTeam__GetName_Label|->PreEstablishedTeam_name_Label),
            (Person__GetName_Label|->Person_name_Label)}
||
PermissionAssignement := {  (RegulatorMARead|->(Regulator|->MedicalAdvice_Label)),
                            (Patient_Perms|->(CallCenterMember|->Patient_Label)),
                            (RegParmMAPerm|->(Regulator|->ManagementAct_Label)),
                            (RegulatorCareRead|->(Regulator|->Care_Label)),
                            (Intervention_Perms|->(CallCenterMember|->Intervention_Label)),
                            (ParmDiagnosisRead|->(PARM|->Diagnosis_Label)),
                            (ParmAdviceMA|->(PARM|->MedicalAdvice_Label)),
                            (TeamMemberMARead|->(TeamMember|->MedicalAdvice_Label)),
                            (Management_Perms|->(CallCenterMember|->Management_Label)),
                            (AdminPreTeamPerm|->(Administrator|->PreEstablishedTeam_Label)),
                            (AdminPersonPerm|->(Administrator|->Person_Label)),
                            (AdminPreActorPerm|->(Administrator|->PreHospitalActor_Label)),
                            (TeamMemberMAPerm|->(TeamMember|->ManagementAct_Label)),
                            (TeamMemberDiagnosisRead|->(TeamMember|->Diagnosis_Label)),
                            (TeamMemberMA|->(TeamMember|->Care_Label)),
                            (RegulatorInstructionMA|->(Regulator|->Diagnosis_Label)),
                            (Team_Perms|->(CallCenterMember|->Team_Label)),
                            (TeamDoctorMA|->(TeamDoctor|->ManagementAct_Label)),
                            (ParmCareRead|->(PARM|->Care_Label)),
                            (RegulatorDiagnosisRead|->(Regulator|->Diagnosis_Label)),
                            (ParmMARead|->(PARM|->MedicalAdvice_Label)),
                            (TeamMemberCareRead|->(TeamMember|->Care_Label))}
||


EntityActions := {  (RegulatorMARead|->{read}),
                    (Patient_Perms|->{create,modify,read}),
                    (RegParmMAPerm|->{read}),
                    (RegulatorCareRead|->{read}),
                    (Intervention_Perms|->{create,modify,read}),
                    (ParmDiagnosisRead|->{read}),
                    (ParmAdviceMA|->{create,modify}),
                    (TeamMemberMARead|->{read}),
                    (Management_Perms|->{create,modify,read}),
                    (AdminPreTeamPerm|->{}),(AdminPersonPerm|->{}),
                    (AdminPreActorPerm|->{}),
                    (TeamMemberMAPerm|->{read}),
                    (TeamMemberDiagnosisRead|->{read}),
                    (TeamMemberMA|->{create,modify}),
                    (RegulatorInstructionMA|->{create,modify}),
                    (Team_Perms|->{create,modify,read}),
                    (TeamDoctorMA|->{create,modify}),
                    (ParmCareRead|->{read}),
```

```
                     (RegulatorDiagnosisRead|->{read}),
                     (ParmMARead|->{read}),
                     (TeamMemberCareRead|->{read})}
    ||
    MethodActions := {  (RegulatorMARead|->{}),
                     (Patient_Perms|->{}),
                     (RegParmMAPerm|->{}),
                     (RegulatorCareRead|->{}),
                     (Intervention_Perms|->{}),
                     (ParmDiagnosisRead|->{}),
                     (ParmAdviceMA|->{MedicalAdvice__NEW_ValidAdvice_Label}),
                     (TeamMemberMARead|->{}),(Management_Perms|->{}),
                     (AdminPreTeamPerm|->{}),(AdminPersonPerm|->{}),
                     (AdminPreActorPerm|->{}),(TeamMemberMAPerm|->{}),
                     (TeamMemberDiagnosisRead|->{}),(TeamMemberMA|->{}),
                     (RegulatorInstructionMA|->{}),
                     (Team_Perms|->{Team__addMembers_Label,Team__removeMembers_Label}),
                     (TeamDoctorMA|->{}),
                     (ParmCareRead|->{}),
                     (RegulatorDiagnosisRead|->{}),
                     (ParmMARead|->{}),
                     (TeamMemberCareRead|->{})}
||
    isPermitted := {}
```

A.3 Calcul des permissions : ensemble isPermitted

```
DEFINITIONS

/*Transformee de relation de EntityActions*/
allEntityActions == {pp, at | pp : PERMISSIONS & at : ActionsType
     & pp : dom(EntityActions) & at : EntityActions(pp)} ;

/* Les roles ayant des des droits de creation : renvoit des couples (role, entite)*/
    PermEntitiesCreation == ran({create} <| (allEntityActions~ ; PermissionAssignement)) ;
    /*les operations de creation permises : renvoit des couples (role, constructeur)*/
    PermOpCreation == (PermEntitiesCreation ; constructorOf~);

/* Les roles ayant des des droits de detruction : renvoit des couples (role, entite)*/
    PermEntitiesDestruction == ran({delete} <| (allEntityActions~ ; PermissionAssignement)) ;
    /*les operations de destruction permises : renvoit des couples (role, destructeur)*/
    PermOpDestruction == (PermEntitiesDestruction ; destructorOf~);

/* Les roles ayant des des droits de lecture : renvoit des couples (role, entite)*/
    PermEntitiesPRead == ran({privateRead} <| (allEntityActions~ ; PermissionAssignement)) ;
    /*les operations de lecture permises : renvoit des couples (role, getterprive)*/
    PermOpPRead == (PermEntitiesPRead ; (getterOf ; AttributeOf)~)  ;

    /*getters des attribus publics*/
    publicGetters == getterOf |> dom(AttributeKind |> {public});
    /* Les roles ayant des des droits de lecture : renvoit des couples (role, entite)*/
    PermEntitiesRead == ran({read} <| (allEntityActions~ ; PermissionAssignement)) ;
    /*les operations de lecture permises : renvoit des couples (role, getter public)*/
    PermOpRead == (PermEntitiesRead ; (publicGetters ; AttributeOf)~) ;

    /*Permissions sur les operations de modification*/
```

```
    PermEntitiesPModify == ran({privateModify} <| (allEntityActions~ ; PermissionAssignement)) ;
    PermOpPModify == (PermEntitiesPModify ; (setterOf ; AttributeOf)~) ;

    publicSetters == setterOf |> dom(AttributeKind |> {public});
    PermEntitiesModify == ran({modify} <| (allEntityActions~ ; PermissionAssignement)) ;
    PermOpModify == (PermEntitiesModify ; (publicSetters ; AttributeOf)~) ;

PermEntitiesAbsoluteRead == ran({privateRead, read} <| (allEntityActions~ ; PermissionAssignement))
PermEntitiesAbsoluteModify == ran({privateModify, modify} <| (allEntityActions~ ; PermissionAssign
PermOpReadOps == (PermEntitiesAbsoluteRead ; (StereotypeOps[{readOp}] <| OperationOf)~)  ;
PermOpModifyOps == (PermEntitiesAbsoluteModify ; (StereotypeOps[{modifyOp}] <| OperationOf)~)  ;

    PermOpMethodAction== {ro, op |ro: ROLES &  op <: Operations & op :ran(MethodActions) & ro : do

    PermOpMethodActions == {ro, op | ro : ROLES & op : Operations & op : union(PermOpMethodAction[

    /*classRoles perms*/
    PermOpAssoRoleRead == (PermEntitiesRead ; (AssoRoleAccessorOf ; AssoRoleOf)~) ;
PermOpAssoRoleModify == (PermEntitiesModify ; (AssoRoleMutatorOf ; AssoRoleOf)~) ;

currentRole == (Session[{currentUser}] \/ ran(Session[{currentUser}]<|closure1(Roles_Hierarchy)))

permissions == PermOpCreation \/
   PermOpDestruction \/
            PermOpPRead \/
   PermOpReadOps \/
            PermOpRead \/
            PermOpPModify \/
   PermOpModifyOps \/
   PermOpModify \/
   PermOpMethodActions
   \/PermOpAssoRoleRead
   \/PermOpAssoRoleModify
```