La norme XACML

XACML [39] est une norme de l'OASIS. Elle consiste essentiellement en un langage XML qui permet d'encoder, d'une part, des politiques de contrôle d'accès de type RBAC étendu et, d'autre part, des requêtes/réponses aux décisions d'autorisation. La norme décrit aussi un modèle de flux de données (figure 1 de [39]) pour une application et un gestionnaire de mise en œuvre d'une politique XACML. Notre travail s'inspire en partie de ce diagramme de flux (voir la section 3.1.3) pour développer le modèle générique décrit au chapitre 4.

3.1.1 Structure d'un document XACML

La figure 3.1 présente le diagramme de classes d'un modèle XACML qui illustre les concepts du langage utiles pour la description d'un document XACML. Un document XACML définit un ensemble de politiques (`PolicySet`). Chaque politique

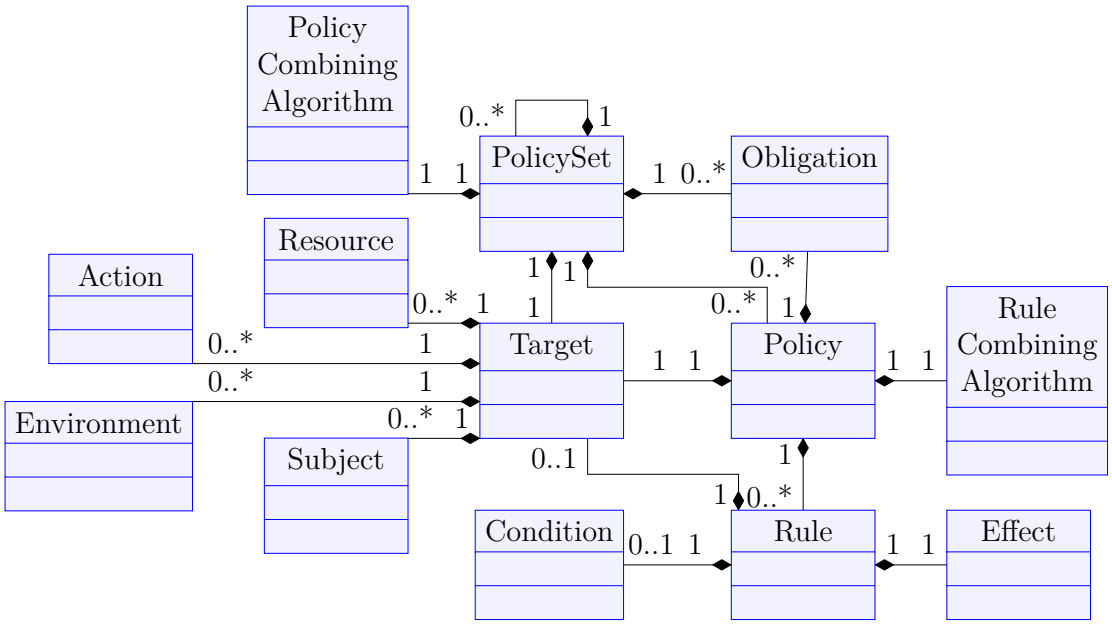

Figure 3.1 – Modèle d'une politique XACML [39]

(`Policy`) vise une cible (`Target`) et comporte des règles (`Rule`) qui sont évaluées dans le contexte courant de la requête d'autorisation reçue de l'environnement. Les règles d'une politique sont évaluées seulement lorsque la cible de celle-ci correspond à la cible de la requête d'autorisation et, les résultats de cette évaluation sont combinés pour fournir un résultat unique pour la politique. La combinaison de ces résultats est faite suivant un algorithme de combinaison (`Rule Combining Algorithm`) précisé par la politique. La politique comporte également des obligations (`Obligation`). XACML définit les obligations comme des actions à exécuter par le gestionnaire de mise en œuvre lorsque la requête d'autorisation traitée est autorisée. Cette fonctionnalité est utilisée par les règles, les politiques et les ensembles de politiques.

Une règle XACML comporte une cible, une condition (`Condition`) et un effet (`Effect`). Tout comme dans le cas des ensembles de politiques et des politiques, la cible indique pour quelles ressources (`Resource`) du système la règle doit être évaluée. Dans le cas d'une requête d'autorisation pour laquelle la cible correspond à la cible précisée par la règle (une requête vise une ou plusieurs ressources du système),

la condition est évaluée. La condition d'une règle est un prédicat qui porte sur les attributs des ressources de la requête et aussi de l'environnement (`Environment`). Les valeurs de retour possibles lors de l'évaluation d'une condition de règle sont *vrai* (*True*) et *faux* (*False*). Suivant la valeur de l'évaluation de la condition de la règle, cette dernière retourne un effet. L'effet correspond à un résultat partiel de la politique qui contient la règle. Les effets possibles sont, entre autres, *permis* (*Permit*) et *non applicable* (*NotApplicable*). Au niveau de la politique, les effets de toutes les règles sont combinés par un algorithme. Par exemple, la combinaison des effets *Permit* et *NotApplicable* par l'algorithme `permit-overrides` retourne l'effet *Permit*.

3.1.2 Requête et réponse d'autorisation

La norme XACML précise un format XML de données pour les requêtes d'autorisation. Un exemple de requête est donné au programme 3.1. Chaque requête précise tous les éléments requis pour que le mécanisme de mise en œuvre puisse calculer un effet. Le sujet (`Subject`) qui est à l'origine de la requête initiant la demande d'autorisation est précisé en premier. Son identifiant est donné par la valeur de l'attribut `subject-id`. Les autres attributs présents dans la requête d'autorisation explicitent des valeurs pour identifier respectivement la ressource qui fait l'objet de l'accès par le sujet, l'action que ce dernier veut exécuter sur la ressource et l'environnement d'exécution de l'action. L'environnement contient des informations pertinentes à l'évaluation de la décision d'autorisation. Un exemple d'une telle information est le lieu, géographique ou institutionnel, à partir duquel le sujet initie l'action. Les réponses à ces requêtes utilisent elles aussi un format XML. Une réponse typique indique essentiellement l'effet de la décision calculée, comme l'illustre le programme 3.2. Elle peut cependant être accompagnée d'obligations à considérer par le gestionnaire de mise en œuvre. Dans les deux programmes, les préfixes et les attributs d'espaces de noms sont omis pour alléger le texte.

3.1.3 Composants d'un PEM

La norme XACML décrit un flux de données dans une application. Ce flux de données ne fait pas partie intégrante de la norme. Cependant le modèle de flux de données

```
<Request>
 <Subject>
  <Attribute
   AttributeId="subject-id"
   DataType="rfc822Name">
   <AttributeValue>
    john@doe.com
   </AttributeValue>
  </Attribute>
 </Subject>
 <Resource>
  <Attribute
   AttributeId="resource-id"
   DataType="http://...#anyURI">
   <AttributeValue>
    file://.../special-file
   </AttributeValue>
  </Attribute>
 </Resource>
 <Action>
  <Attribute
   AttributeId="action-id"
   DataType="http://...#string">
   <AttributeValue>
    read
   </AttributeValue>
  </Attribute>
 </Action>
 <Environment/>
</Request>
```

Programme 3.1 – Requête XACML

```
<Response>
 <Result>
  <Decision>
   NotApplicable
  </Decision>
 </Result>
</Response>
```

Programme 3.2 – Réponse XACML

(voir la figure 3.2) fait ressortir divers composants nécessaires à la spécification d'une politique XACML et à sa mise en œuvre dans une application.

Le premier composant qui intervient dans le flux de données est le « Policy Administration Point » (PAP). Il est le composant responsable de la spécification des politiques XACML. Il fournit les fonctions d'édition et de mise à jour des politiques. La norme suggère que plusieurs composants PAP peuvent coopérer pour la spécification de règles ou de politiques pour une cible. Le PAP doit aussi, lorsque la nature des données référencées dans la politique sous édition le requiert, appliquer des mesures de contrôle d'accès et assurer la confidentialité d'informations sensibles.

Les politiques spécifiées avec le PAP sont accédées par le PDP. Cet accès peut être réalisé au moyen d'un dépôt sécurisé. Le PDP calcule des décisions d'autorisation

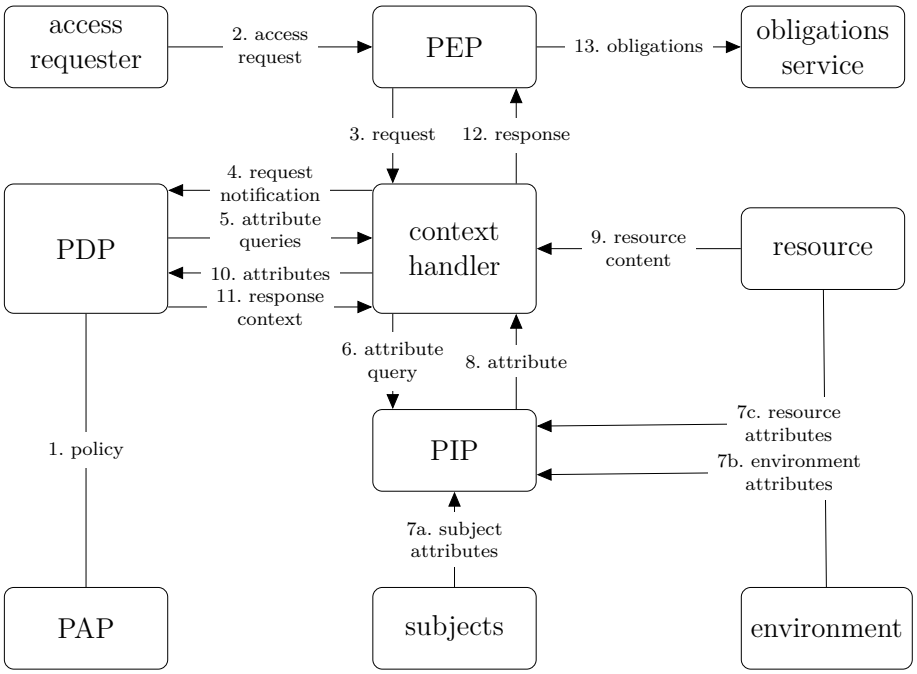

Figure 3.2 – Diagramme de flux de données XACML [39]

pour les politiques déployées par le PAP. La décision d'autorisation est calculée à la suite d'une requête d'autorisation. Les politiques qui correspondent à la cible de la requête d'autorisation sont utilisées pour faire le calcul. En plus des informations de la requête d'autorisation, le composant PDP peut nécessiter d'autres informations. Celles-ci, suivant la norme XACML, sont des attributs des objets du système. Ainsi, des attributs de la ressource accédée ou encore du sujet de la requête peuvent être demandées par le PDP pour arriver à une décision d'autorisation. Le composant responsable de la collecte des valeurs de ces attributs est le « Policy Information Point » (PIP). Il a la responsabilité de fournir au PDP, de façon indirecte, les valeurs des attributs sur les sujets, l'environnement et les ressources du système.

Du point de vue de la norme XACML, le lien entre le système à sécuriser et les autres composants du mécanisme de contrôle d'accès se fait à travers le PEP. Les requêtes aux ressources du système passent par le PEP. À la suite de la décision d'autorisation calculée par le composant PDP, le PEP autorise ou pas l'accès à la

ressource visée. Dans le cas où la réponse permet l'accès demandé à la ressource, le PEP exécute non seulement l'action spécifiée par la requête originale, mais aussi les éventuelles obligations attachées à la décision d'autorisation rendue par le PDP. L'exécution des obligations est la responsabilité d'un *service des obligations*. Entre le PEP et le PDP, le *gestionnaire de contexte* transmet les requêtes en effectuant la conversion nécessaire entre le format de la requête initiale du sujet et le format de requête XACML. Le gestionnaire de contexte achemine aussi les requêtes et réponses des valeurs des attributs entre le PDP et le PIP.

3.1.4 Discussion

La solution proposée dans cette thèse met d'avantage l'emphase sur les composants PEP, PDP et PIP, plus particulièrement sur leurs interactions qui sont différentes de celles présentes dans la partie non normalisée du document [39]. Contrairement au modèle XACML, dans lequel un seul type de politiques est défini de manière générale, notre modèle de politiques de contrôle d'accès comporte plusieurs spécialisations de politiques, ce qui entraîne une différentiation importante au niveau des interactions des composants. Notre solution permet aussi le déploiement de PDP hiérarchiques de façon à prendre en compte la priorité des politiques, ce qui n'est pas possible avec l'architecture XACML. Des aspects comme les algorithmes de combinaison sont aussi inclus dans notre solution, bien qu'ils constituent des éléments de moindre importance.

3.2 Implémentations du contrôle d'accès

Dans cette section, deux approches du contrôle d'accès basées sur RBAC sont présentées. Celles-ci sont complètes car elles comprennent à la fois un langage ou une notation pour spécifier les politiques de contrôle d'accès et un mécanisme de mise en œuvre adapté à la notation. Les implémentations des autres modèles de contrôles d'accès, comme les ACL, les matrices de contrôle d'accès et le modèle de Bell-LaPadula, sont omis puisqu'elles sont généralement intégrées dans des systèmes d'exploitation. Dans tous ces derniers cas, ces méthodes sont hors de notre champ d'application.

3.2.1 Le cadre d'applications X-GTRBAC

Bhatti et al. [6] ont développé « XML Generalized Temporal Role-Based Access Control » (X-GTRBAC), un langage pour l'expression de politiques de contrôle d'accès. Il est basé sur le modèle « Generalized Temporal Role-Based Access Control » (GTRBAC) [33] qui lui-même est fondé sur la norme RBAC. GTRBAC ajoute des contraintes temporelles généralisées à RBAC décrit à la section 2.6. X-GTRBAC est essentiellement une grammaire hors-contexte qui décrit le format de documents XML pour l'écriture de politiques de contrôle d'accès GTRBAC.

GTRBAC reprend en effet tous les éléments d'une politique de contrôle d'accès RBAC. Cela inclut les utilisateurs, les rôles, les permissions et les relations entre, d'une part, les utilisateurs et les rôles et, d'autre part, les rôles et les permissions. Il supporte aussi les contraintes de type *séparation des devoirs* (statique et dynamique) de la norme. GTRBAC ajoute des contraintes de temps à tous ces éléments. Une permission peut être assignée à un rôle pour une période de temps précise. Cette contrainte de temps peut inclure aussi d'autres conditions en fonction du contexte. Une telle contrainte signifie que le rôle possède la permission en question seulement pour l'intervalle de temps spécifié et si seulement l'éventuelle condition additionnelle s'évalue à vrai. De même GTRBAC permet de définir un intervalle de temps pendant lequel un rôle est assigné à un utilisateur, ainsi qu'une contrainte additionnelle qui doit s'évaluer à vrai pour que l'assignation se fasse.

Les rôles, avec la notation GTRBAC, peuvent aussi être spécifiés pour être autorisés ou pas à certaines périodes de temps. Lorsqu'un rôle n'est pas autorisé pendant un intervalle de temps, cela signifie que ce rôle existe bien mais qu'il ne peut être activé par aucun utilisateur durant cet intervalle de temps. Tout comme les relations, l'intervalle de temps utilisé pour les rôles peut être associé à des contraintes exprimées en fonction du contexte d'exécution. La notation GTRBAC permet la spécification d'événements pendant la phase d'exécution des systèmes. L'*activation* ou la *désactivation* d'un rôle par un utilisateur au cours d'une session sont des exemples de tels événements. Il est possible d'exprimer des relations de cause à effet entre les événements. GTRBAC permet ainsi de spécifier des événements futurs en fonction

d'événements antérieurs : ce sont des *déclencheurs*, avec l'exception que l'activation d'un rôle — qui est un événement — ne peut être réalisée que par un utilisateur.

X-GTRBAC permet de représenter tous les concepts définis avec GTRBAC. Ainsi il est possible avec X-GTRBAC de spécifier dans un format XML les rôles (avec leurs justificatifs d'identité et les ensembles de séparations des devoirs auxquels ces rôles appartiennent) et les permissions (comportant chacune l'objet et l'opération pour lesquels elle symbolise un privilège). Il permet également de spécifier les relations entre, d'une part, les rôles et les permissions et, d'autre part, les utilisateurs et les rôles. La spécification des instances de chacune de ces entités se conforme au format défini par X-GTRBAC et les instances sont regroupées par entités dans des *feuilles*. Le format prend aussi en charge la spécification des contraintes temporelles qui peuvent être attachées aux rôles (pour leur autorisation ou leur activation) et aux relations précédemment mentionnées. Enfin, le format permet aussi de spécifier les déclencheurs qui peuvent être activés pendant l'exécution.

X-GTRBAC fournit non seulement un format de document XML pour représenter les politiques de contrôle d'accès, mais aussi une architecture de composants pour le calcul de décisions d'autorisation. Le système ainsi modélisé prend en paramètres une politique de contrôle X-GTRBAC et une requête d'accès, et calcule une décision d'autorisation qui sera utilisée par le mécanisme de mise en œuvre pour autoriser ou pas l'accès à la ressource visée. L'architecture de composants est illustrée à la figure 3.3. Les administrateurs du système utilisent le module de composition de documents pour spécifier les politiques de contrôle d'accès X-GTRBAC. Les politiques ainsi spécifiées sont ensuite chargées dans le module X-GTRBAC pour le calcul de décision. Ce module est constitué d'un processeur XML et d'un processeur GTRBAC. Le processeur XML analyse des politiques de contrôle d'accès et construit un arbre « Document Object Model » (DOM) correspondant à celles-ci. Avant le traitement par le processeur XML, les documents des politiques sont chargés et validés respectivement par un module de chargement et un module de validation. La validation vérifie par exemple que les utilisateurs référencés dans la feuille de la relation entre les utilisateurs et les rôles existent dans la feuille des utilisateurs du système.

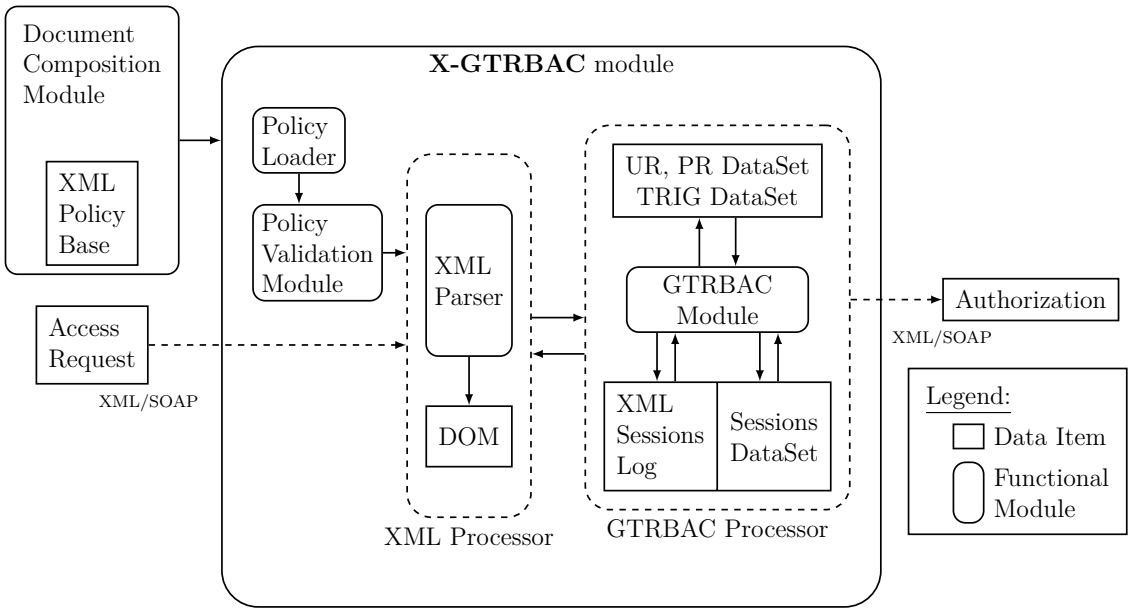

Figure 3.3 – Architecture de composants pour une décision X-GTRBAC [6]

L'arbre construit par le processeur XML est transmis au processeur GTRBAC. Le module GTRBAC enregistre l'arbre DOM dans sa propre base de données et procède à des vérifications de cohérence supplémentaires. L'information contenue dans cette base de données est ensuite utilisée par le module GTRBAC pour gérer les sessions des utilisateurs lorsqu'ils se connectent et pour calculer les décisions d'autorisation en fonction des requêtes d'accès aux ressources du système. Le module X-GTRBAC, illustré à la figure 3.3, est implémenté comme une application Java avec une interface graphique nécessaire pour afficher les données de la politique de contrôle d'accès telles que les utilisateurs ou les rôles.

3.2.2 Cadre d'applications W-RBAC pour les workflows

L'expression « Workflow Role-Based Access Control » (W-RBAC) [50] désigne un cadre d'applications pour le développement d'applications de workflows sécurisés. Un *workflow* est un ensemble partiellement ordonné de tâches dans lequel chaque tâche est une unité atomique de travail à réaliser par un intervenant humain ou par un ou plusieurs systèmes automatiques. W-RBAC est constitué de deux modèles. Le premier

modèle, W0-RBAC, combine un modèle de politiques de contrôle d'accès basé sur RBAC à un système d'exécution de workflow. Le second modèle, W1-RBAC, définit un mécanisme de gestion des exceptions qui permet d'outrepasser les contraintes spécifier par le modèle de base pour le fonctionnement normal du système.

Un système de gestion de workflows (Workflow Management System (WfMS) en anglais) utilise les spécifications des workflows pour instancier des cas particuliers. Cette instanciation est semblable à la création de n-uplets dans une base de données relationnelles suivant son schéma. L'admission à une université est un exemple de workflow et les demandes d'admission des étudiants prospecteurs sont des cas de ce workflow. Les tâches à exécuter immédiatement par des intervenants humains sont attribuées aux intervenants qui disposent de la compétence nécessaire. Par exemple, des tâches comptables seront affectées aux employés du service de comptabilité présents. Lorsqu'une tâche est complétée, d'autres tâches deviennent exécutables suivant la relation d'ordre partielle qui définit le workflow. W-RBAC prend en compte la nature spécifique de l'exécution des workflows.

Le modèle W0-RBAC est basée sur la norme RBAC. Une politique W0-RBAC comporte des utilisateurs, des rôles et des permissions, ainsi que les deux relations usuelles entre ces relations : l'assignation des rôles à des utilisateurs et l'assignation des permissions à des rôles. Les permissions dans le contexte des workflows sont les privilèges d'exécuter des tâches. En plus des entités du modèle RBAC de base, W0-RBAC ajoute les entités *cas* et *unité organisationnelle*. Un *cas* représente une instance particulière d'un workflow en cours d'exécution. Une *unité organisationnelle* regroupe des utilisateurs du système, de façon à mieux correspondre à la division du travail au sein d'une organisation. Cette entité et les relations qui sont définies à partir de celle-ci facilitent l'attribution des tâches aux utilisateurs en fonction de leur implication dans les unités de l'organisation. Par exemple, dans une entreprise un employé fait partie du département assurance qualité des produits matériels et il est impliqué pour quelques mois dans un projet de développement logiciel. Dans cet exemple, l'employé fait partie de deux unités organisationnelles : celle du département de la qualité et celle du projet de développement en cours. W0-RBAC définit les

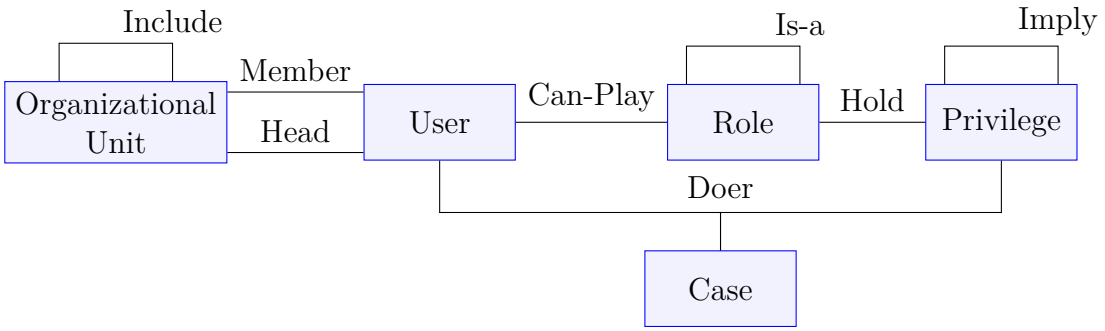

Figure 3.4 – Métamodèle W0-RBAC [50]

relations pour spécifier l'appartenance d'un utilisateur à une unité organisationnelle, et également pour définir le responsable de celle-ci. Il est aussi possible, à la manière de la hiérarchie sur les rôles, d'avoir une hiérarchie sur les unités organisationnelles par une relation réflexive sur les unités. Enfin, W0-RBAC définit une relation ternaire entre les utilisateurs, les permissions et les cas. Cette relation est utilisée pour savoir quel utilisateur a exécuté une certaine tâche pour un cas particulier. Le métamodèle qui comprend toutes ces relations est représenté à la figure 3.4.

À partir du métamodèle ainsi défini, W0-RBAC permet de raffiner le contrôle d'accès par des contraintes. Celles-ci sont écrites comme des programmes logiques par lesquels les états indésirables du système sont spécifiés. W0-RBAC distingue les *contraintes statiques* et les *contraintes dynamiques*. Les contraintes statiques définissent les modifications qui peuvent être faites aux données d'une politique. Ces données comprennent, en autres, les assignations des rôles aux utilisateurs tout comme les assignations des permissions aux rôles. Les contraintes dynamiques font intervenir les cas ou les tâches dans l'expression des programmes logiques. Elles sont plus larges que les contraintes statiques et sont utilisées pour préciser les conditions sous lesquelles des utilisateurs peuvent exécuter des tâches dans des cas de workflows. Les contraintes dynamiques peuvent exprimer par exemple des contraintes de type SoD.

W0-RBAC définit aussi un service de permission qui interagit avec le système de gestion de workflows. Le service de permission a un accès immédiat à toute l'informa-

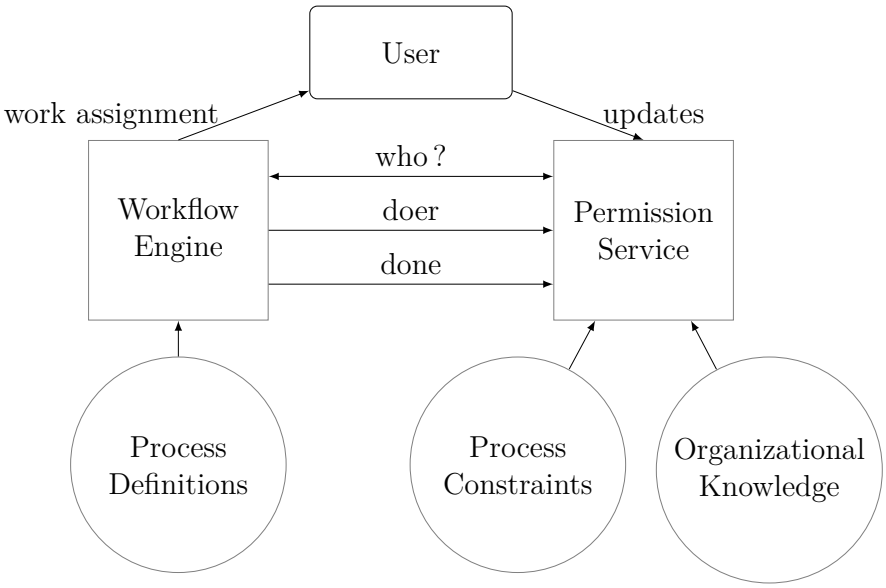

Figure 3.5 – Interactions entre les composants et les utilisateurs [50]

tion organisationnelle (par exemple, utilisateurs, rôles, unités organisationnelles) et aux contraintes sur les workflows, tandis que le service de gestion de workflows gère l'information sur les workflows et les cas en cours d'exécution. Cette architecture est illustrée à la figure 3.5. Le service de permission et le système de gestion des workflows interagissent par trois canaux. Le premier canal, nommé *who ?*, permet au composant de gestion des processeurs d'interroger le service de permission pour déterminer les utilisateurs qui peuvent exécuter une tâche donnée. Ce même canal est utilisé par le service de permission pour répondre à cette requête. La réponse est une liste ordonnée de groupes d'utilisateurs. Ces utilisateurs peuvent tous exécuter la tâche dans le cas passé en paramètre car ils ne violent aucune contrainte, en particulier celles inhérentes aux assignations des rôles aux utilisateurs et des permissions aux rôles. Ils sont groupés et listés par ordre de préférence décroissante. Le second canal, nommé *doer*, informe le service de permission sur l'identité de l'utilisateur qui a exécuté une tâche dans un cas d'un workflow. Cette information est utilisée pour mettre à jour la relation ternaire du modèle correspondant. Le dernier canal, nommé *done*, indique au service de permission que l'exécution d'un cas d'un workflow est complétée.

W1-RBAC vient compléter W0-RBAC pour la gestion des situations exceptionnelles. Il peut arriver souvent avec les workflows qu'un cas *bloque* parce que les contraintes exprimées sont restrictives à un point que la requête *who ?* retourne une liste vide. Dans ces situations, un utilisateur approprié du système peut outrepasser la politique établie pour débloquer le cas, par exemple en affectant un utilisateur à la tâche bloquée du système. W1-RBAC permet de représenter cette fonctionnalité avec deux éléments de spécification. Le premier élément est l'attribution d'un *niveau de priorité* aux contraintes de la politique de contrôle d'accès. Un niveau de priorité est tout simplement un entier positif non nul attaché à une contrainte. Ainsi les contraintes de même niveau de priorité peuvent être regroupées ensemble. Le second élément de spécification est la permission spéciale *override(n)* où n est un entier positif. Cette permission signifie qu'un utilisateur qui la possède peut outrepasser toutes les contraintes ayant un niveau de priorité inférieur ou égal à n.

W1-RBAC est intégré à l'architecture précédente illustrée à la figure 3.5 par l'ajout d'un canal spécial $assign?(u_1, u_2, t, c)$. Ce canal est utilisé par le système de gestion de workflows pour communiquer au service de permission que l'utilisateur u_1 veut forcer la réalisation de la tâche t dans le cas c d'un workflow par l'utilisateur u_2. Dans ce cas, l'utilisateur u_2 doit posséder le droit RBAC de réaliser t. De plus le métamodèle de la figure 3.4 doit être complété avec des éléments pour représenter la délégation ou le transfert de droits qui se produit entre les utilisateurs u_1 et u_2. Forcer la réalisation d'une tâche peut violer des contraintes exprimées au niveau W0-RBAC et l'utilisateur u_1 doit posséder par l'un de ses rôles actifs une permission *override(n)* telle que l'entier n soit plus grand que toutes les contraintes violées. Dans le cas où l'utilisateur u_1 ne possède pas une telle permission alors le canal spécial $assign?(u_1, u_2, t, c)$ retourne la valeur *false* au système de gestion de workflows pour indiquer que l'affectation de la tâche ne peut être exécutée. Dans le cas contraire, le service de permission met à jour la base de données avec la nouvelle affection et retourne le niveau de priorité de l'utilisateur u_1 qui lui permet d'outrepasser la politique de contrôle d'accès normale du système.

3.2.3 Discussion

Le cadre d'applications X-GTRBAC est propre à une extension de RBAC qui ajoute des contraintes temporelles. La mise en œuvre de politiques de contrôle d'accès X-GTRBAC n'utilise pas les composants du PEM de la norme XACML bien que Batthi et al. aient proposé avec la notation un moteur de calcul de décision. Leur prototype d'implémentation est réalisé en Java mais n'est pas accompagné d'une étude de performance. Les implémentations réalisées dans nos travaux sont différentes car elles ont une architecture SOA et sont accompagnées d'une étude de performance. De plus, les mécanismes de contrôle d'accès sont basés sur les composants introduits par la norme XACML.

Wainer et al. ont proposé W-RBAC qui consiste en une extension de la norme RBAC ajoutant la notion d'organisation et des programmes logiques pour exprimer des contraintes. Cette notation a été définie pour permettre le contrôle d'accès dans des WfMS. L'approche proposée par Wainer et al. diffère de nos travaux tout d'abord par la notation utilisée pour exprimer les politiques de contrôle d'accès. Le langage ASTD définit des traces acceptables pour le système tandis que W1-RBAC définit des états acceptables. D'autre part, les systèmes à sécuriser que nous considérons sont des applications SOA alors que Wainer et al. traitent des WfMS pour lesquels un prototype réalisé en Prolog a permis de valider les concepts mais pas l'efficacité en situation réelle. Dans nos travaux, des tests unitaires et de performance appuient les méthodes proposées.

Soulignons aussi les travaux de Bourdier et al. [9] qui ont développé un cadre d'applications pour des systèmes sécurisés. Leur approche est basée sur des règles formelles de réécriture. D'une part, une politique de contrôle d'accès est un système de règles réécriture de termes, et d'autre part, l'état d'une politique de contrôle d'accès est un ensemble de faits. À chaque décision d'autorisation rendue, la base de faits est mise à jour par l'application des règles de réécriture. Ce cadre d'applications permet également la validation et vérification de propriétés de sécurité. L'approche est semblable à celle proposée par Wainer et al. dans [50] qui est elle cependant spécialisée pour les WfMS.

Enfin, Mao et al. [37] ont proposé une implémentation de la norme XACML adaptée au contrôle de l'accès à des documents numériques. Leur méthode résout, sans toutefois préciser comment, les problèmes de flot d'information solutionnés par Bell-LaPadula avec les règles « no read up » et « no write down ». La solution à ces problèmes repose sur la recherche des chemins potentiels de fuite d'information et l'attribution de contraintes aux rôles XACML pour empêcher ces fuites.

3.3 Transformation de modèles

La transformation de modèles est le procédé par lequel une spécification écrite dans un langage de départ est traduite en une spécification équivalente dans un langage d'arrivée. Cette technique est à la base de méthodologies de développement logicielle telles que « Model-Driven Engineering » (MDE) [45]. Les transformations peuvent être distinguées selon plusieurs critères. Quand les langages de départ et d'arrivée sont identiques, on parle de transformation *endogène*, dans le cas contraire la transformation est dite *exogène*. La transformation est *bidirectionnelle* lorsqu'il est possible d'obtenir le modèle de départ à partir du modèle d'arrivée. Sinon elle est simplement *unidirectionnelle*.

Pour modéliser une transformation, les langages d'entrée et de sortie doivent être spécifiés d'abord. En MDE, cette spécification prend couramment la forme d'un métamodèle pour chacun des langages. Ensuite, la transformation elle-même est spécifiée comme une correspondance entre les éléments du métamodèle du langage de départ et ceux du métamodèle du langage d'arrivée. Le plus souvent, chaque élément du métamodèle du langage de départ est mis en correspondance avec un ensemble d'éléments du métamodèle du langage d'arrivée. Par exemple dans l'application de la méthode de développement MDE, un modèle UML d'entités d'un système d'information est transformé en un modèle de données relationnelles pour un SGBD moderne et une bibliothèque d'objets Java pour la couche d'accès aux données d'une application multi-tiers. Ces transformations sont illustrées à la figure 3.6.

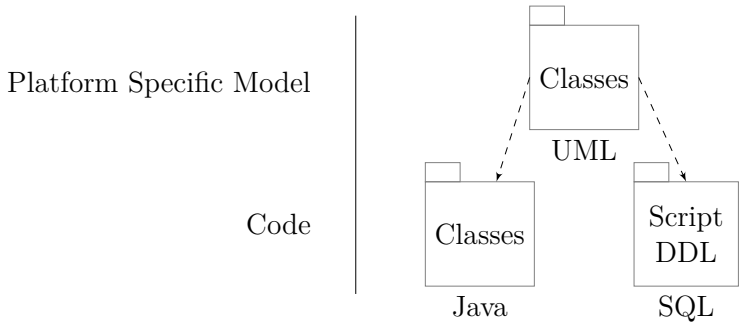

Figure 3.6 – Transformations de modèles dans MDE

Relativement à BPEL, de nombreux travaux ont été menés soit pour simplifier le développement de processus BPEL [54, 10], soit pour préciser leur sémantique [49] avec des réseaux de Petri [43], avec des réseaux de Petri colorés [53], avec Event-B [3], avec des automates [25, 52], avec des machines à états abstraites (Abstract-State Machines (ASM) en anglais) [20] ou encore des algèbres de processus [44]. Seuls [10, 3] sont présentés, car parmi tous ces travaux ce sont les seuls qui se rapprochent le plus des nôtres.

3.3.1 Vérification de processus BPEL

Ait-Sadoune [3] a proposé une transformation de processus BPEL en des modèles Event-B. Cette transformation a pour but la vérification et la validation des processus vis-à-vis de propriétés. Elle s'effectue en deux étapes. La première, dite statique, transforme la définition des services WSDL associés à un processus BPEL vers un **CONTEXT** d'un modèle Event-B. La seconde phase transforme le processus BPEL en un modèle Event-B. Ces deux étapes ont été implémentées dans l'outil BPEL2B.

L'étape statique traite des documents WSDL et XSD associés à des processus BPEL. Les types de données, les messages, les opérations et les types de port définis par un fichier WSDL sont traduits dans un **CONTEXT** Event-B. Par exemple, un message WSDL est transformé en un ensemble (élément **SETS** de Event-B) et chaque partie du message (élément **part** de WSDL) est transformé en une constante (élément **CONSTANT** de Event-B). Ces nouveaux éléments Event-B sont liés ensuite

par des axiomes (élément **AXIOMS** de Event-B). Un axiome spécifie que l'ensemble correspondant au message n'est pas vide et pour chaque partie du message, un axiome spécifie que la constante associée à la partie du message est une fonction. Une opération dans un document WSDL devient une constante Event-B et un axiome spécifie que cette constante est une fonction de l'ensemble encodant le type du paramètre d'entrée vers l'ensemble encodant le type paramètre de sortie. Dans l'éventualité où l'opération n'a pas de paramètre d'entrée ou de paramètre de sortie, l'ensemble correspondant est remplacé par l'ensemble spécial `Void`.

L'étape dynamique construit une machine Event-B à partir d'un processus BPEL. Cette machine utilise le contexte construit à partir du document WSDL de l'interface du processus BPEL. Ait-Sadoune distingue deux groupes d'activités BPEL : les activités simples et les activités structurées. Les activités simples comportent les activités de communication avec d'autres services Web, les activités de manipulation de données et les activités de contrôle telles que **wait**, **empty** et **exit**. Leur transformation dans la machine du processus produit un événement. Les activités structurées quant à elles sont les activités qui permettent de composer des activités de base, telles que les activités itératives, conditionnelles (**if**) ou de contrôle de flot. Leur transformation produit, dans les événements créés pour les activités englobées, des éléments pour encoder le comportement de l'activité structurée concernée. Par exemple, pour une séquence qui contient n activités simples, une variable entière est créée et sa valeur initialisée à n. Cette variable est décrémentée par l'exécution des événements produits pour chaque activité simple de la séquence et de plus, ces exécutions sont gardées par une valeur spécifique de la variable. L'événement de la $k^{ème}$ activité de la liste ordonnée des activités de la séquence BPEL ne peut s'exécuter que si la variable associée à la séquence a la valeur $n - k + 1$. Dans le cas général, cette condition d'exécution relative à la séquence est ajoutée à d'autres conditions inhérentes à la traduction de l'activité simple ou d'autres activités structurées englobantes.

Avec cette transformation, Ait-Sadoune définit une approche de conception verticale de processus BPEL. Dans celle-ci un processus BPEL est développé de façon incrémentale : une version abstraite du processus est enrichie de détails en même temps

que la machine Event-B correspondante est obtenue par raffinement de la machine de l'itération précédente. Cette approche permet l'introduction graduelle d'obligations de preuve à un niveau adéquat d'itération de la transformation, simplifiant ainsi la décharge de celles-ci.

3.3.2 Développement de processus BPEL corrects

Chirichiello et Salaün [10] ont proposé une méthode de développement de processus BPEL dans laquelle une algèbre de processus est utilisée pour spécifier le comportement attendu d'un processus. Cette approche diffère de la précédente dans laquelle le but est de valider le comportement d'un processus BPEL existant. Dans l'approche de Chirichiello et Salaün, le comportement désiré de l'interaction est spécifié et les outils disponibles pour l'algèbre de processus sont utilisés pour vérifier des propriétés désirables pour l'interaction. La transformation proposée utilise le langage LOTOS comme langage de départ et produit non seulement le processus BPEL mais aussi un document WSDL pour en décrire l'interface.

Dans [10], un guide de la traduction de LOTOS vers BPEL est décrit. Les émissions et réceptions sur des portes LOTOS sont traduites respectivement en des activités BPEL **`reply`** et **`receive`**. Lorsque l'émission et la réception sont spécifiées en séquence (opérateur « ; » de LOTOS), alors elles sont toutes deux traduites en une seule activité **`invoke`**. Ces activités de communication sont créées avec comme paramètres des opérations correspondant aux portes de l'émission ou réception correspondante. Les variables de ces activités BPEL correspondent aux variables utilisées dans LOTOS pour communiquer sur les portes. De ces opérations de communication LOTOS est aussi extraite la déclaration de l'interface du processus, notamment les messages et les opérations qui correspondent aux portes LOTOS. L'opération de séquence LOTOS est traduite en une activité BPEL **`sequence`**. Une action cachée du processus LOTOS correspond à une activité non visible de l'extérieur du processus, par exemple une activité **`assign`**. L'opérateur garde de LOTOS est traduit en l'activité **`case`** (**`if`** dans la nouvelle version de la norme BPEL). Un opérateur choix LOTOS avec des gardes pour chaque option du choix est traduit en une activité **`case`** avec différentes branches **`switch`**. Dans la nouvelle version de la norme BPEL, c'est l'activité BPEL

if qui est utilisée avec autant de branches conditionnelles **elseif** que nécessaires. Un choix LOTOS avec des réceptions sur des portes est traduit en une activité **pick** avec autant de branches **onMessage** que de réceptions sur des portes.

3.3.3 Discussion

Les travaux de Chirichiello et Salaün [10] ont pour but de construire des processus BPEL corrects vis-à-vis d'une spécification. C'est aussi l'une des démarches effectuées dans cette thèse, notamment au chapitre 5. Mais contrairement à leur approche qui constitue plutôt un guide pour la traduction de spécifications LOTOS en processus BPEL, notre approche est systématique puisqu'elle ne requiert pas d'intervention humaine. Une autre différence se manifeste dans le langage utilisé pour la spécification du comportement attendu des processus et aussi dans la finalité de ces processus BPEL. Dans nos travaux nous nous intéressons au langage ASTD, alors que dans [10] LOTOS est la notation privilégiée. De plus les processus BPEL visés dans cette thèse ont la finalité de servir à la mise en œuvre de politiques de contrôle d'accès. Nos travaux se basent aussi sur l'interprétation de spécifications formelles pour achever le même résultat, ce qui n'est pas le cas dans [10]. La pertinence des travaux de Ait-Sadoune [3] se manifeste dans la correspondance qu'il établit entre les éléments du langage BPEL et un autre langage, mais BPEL est dans ce cas le langage de départ et non le langage cible. La transformation doit cependant être guidée dans le processus de conception verticale. Au chapitre 5, les structures du langage ASTD sont mises en correspondance avec les activités du langage BPEL.