Une classe possède des propriétés qui peuvent être impliquées dans des invariants de cette classe.

##### Support

Une propriété est d’un certain type.

Toutes les propriétés du type A où A est une classe forment le « support » de A.

Réciproquement, la classe A est aussi appelée support de ces propriétés.

Considérons les invariants de la classe Flight :

- crew : captain <> copilot

- captainIsCertified : captain.certificates.planeModel → includes(self.plane.planeModel)

- captainLivesInParis : captain.address.town = « Paris »

Le support de la classe Pilot contient les propriétés :

- captain (appartenant à la classe Flight)

- copilot (appartenant à la classe Flight)

Le support de la classe PlaneModel contient les propriétés

- planeModel (appartenant à la classe Certificate)

- planeModel (appartenant à la classe Plane)

##### Requête

Toute propriété p dans le support d’une classe A est associée à une requête dans la class A.

Ainsi les requêtes getAllPilotsAsCaptainFor(Flight f) et getAllPilotsAsCopilotFor(Flight f) sont créées dans la classe Pilot.

La requête getAllPlaneAsPlaneFor(Flight f) est créée dans la classe Plane.

Elles seront traduites en méthodes Java et requêtes SQL dans les classes PilotDAO et PlaneDAO.

##### Filtre

Chaque requête contient des filtres.

Soit p la propriété associée à la requête ; il y a un filtre pour chaque invariant où intervient la propriété p au départ d’une navigation.

Ainsi, la requête getAllPilotsAsCaptainFor(Flight f) contient trois filtres :

- filterCaptainForCrew

- filterCaptainForCaptainIsCertified

- filterCaptainForCaptainLivesInParis

Remarque :

un même invariant peut générer plusieurs filtres dans plusieurs requêtes. Ainsi captainIsCertified génère les filtres

- filterCaptainForCaptainIsCertified dans la classe Pilot

- filterPlaneForCaptainIsCertified dans la classe Plane

##### Dépendances

Un filtre a des dépendances. Ce sont les navigations qui ne commencent pas par la propriété associée à la requête à laquelle appartient le filtre.

Ainsi le filtre filterCaptainForCaptainIsCertified a pour dépendance self.copilot, le filtre filterPlaneForCaptainIsCertified a pour dépendance self.captain.certificate.planeModel.

##### SQL associé à la requête

Chaque requête doit être ultérieurement traduite par une requête SQL. La requête contient des jointures pour toutes les navigations possibles dans tous les filtres.

Ainsi (le début de) la requête SQL correspondant à getAllPilotsAsCaptainFor(Flight flight) est :

SELECT DISTINCT v0.\* FROM PILOT AS v0

LEFT JOIN PILOT\_CERTIFICATE AS v1 ON v0.ID=v1.PILOT\_ID

LEFT JOIN CERTIFICATE AS v2 ON v2.ID=v1.CERTIFICATES\_ID

LEFT JOIN PLANEMODEL AS v3 ON v3.ID=v2.PLANEMODEL\_ID

LEFT JOIN ADDRESS AS v4 ON v4.ID=v0.ADDRESS\_ID

Pour réaliser cela, la requête conserve les navigations et leur associe un numéro :

Pilot → Certificate 2

Certificate → PlaneModel 3

Pilot → Address 4

On voit que dans cette liste, il y a un « saut » dans la navigation, du fait de la table intermédiaire PILOT\_CERTIFICATE.

Chaque jointure est représentée par une variable. La variable de la jointure finale du filtre sera utilisée dans les clauses WHERE associées à ce filtre.

##### SQL associé au filtre

Chaque filtre est traduit en SQL sous forme d’une clause WHERE.

Ainsi le filtre filterCaptainForCrew est traduit par

(WHERE/AND) v0.ID <> :copilot

Le filtre filterCaptainForCaptainIsCertified est traduit par

(WHERE/AND) v3.ID = :plane\_planeModel

##### Fonction Java (ou autre) associée à la requête

Lors de la génération du code, chaque requête est traduite par une fonction dans la classe DAO de la classe support.

Cette fonction générée demande au dialecte SQL la requête de base (sans les clauses where) telle que définie au §5.

Ensuite le template génère le calcul des dépendances de chacun des filtres.

Enfin le template génère un code vérifiant que toutes les dépendances d’un filtre sont présentes et dans ce cas appelant la fonction associée au filtre.

Ainsi on obtient la fonction java suivante :

public List<Pilot> getAllPilotsAsCaptainFor(Flight flight){

EntityManager em = getEntityManager();

StringBuilder builder = new StringBuilder(

« SELECT v0.\* FROM ...ON v4.ID=v0.ADDRESS\_ID »);

OptionalPilot copilot = flight.getCopilot$();

String where = « WHERE »;

if (copilot.isPresent()) {

filterCaptainForCrew(builder, where);

where = « AND »;

}

OptionalPlaneModel plane\_planeModel

= flight.getPlane$().getPlaneModel();

if (plane\_planeModel.isPresent()) {

filterCaptainForCaptainIsSelected(builder, where);

where = « AND »;

}

filterCaptainForCaptainLivesInParis(builder, where);

where = « AND »;

Query query = em.createNativeQuery(builder.toString());

if (copilot.isPresent())

query.setParameter(«copilot», copilot.get());

if (plane\_planeModel.isPresent())

query.setParameter(«plane\_planeModel»,

plane\_planeModel.get());

return query.resultList();

}

##### Fonction Java (ou autre) associée au filtre

Le template doit aussi générer une fonction par filtre.

La fonction associée

##### Processus

###### Visite des propriétés

Pour chaque propriété, on crée une requête dans le support de la propriété ; c’est la Map<Propriete, Requete> de la classe EnhancedMofClassImpl

###### Visite des invariants

On visite ensuite chaque invariant ; pour chaque navigation, on ajoute un filtre dans la requête associée à la propriété de la navigation initiale de cette expression. On mémorise aussi les navigations dans l’invariant.

À la fin de la visite de l’invariant, on ajoute les dépendances aux différents filtres : ce sont toutes les navigations différentes de celle associée au filtre (en fait à la requête du filtre).

###### Visite des supports

On numérote les navigations dans les filtres d’une requête.