AspectOCL Constraints- Case Study 1

AspectOCL Constraint 1:

mapping mapNameUniqueness

{

let T:{ Branch, Branchtype, PerformanceIndicator, Country, CarModel, CarGroup, RentalDuration, ServiceDepot, Discount}

}

aspect NameUniqueness

{

import_mapping mapNameUniqueness

pointcut uniqueNames

context T :: nameIsKey () : Boolean

intro:

post uniqueNames

result= T.allInstances()-> **select**(t|t.name=self.name)->**size**()=1

}

AspectOCL Constraint 2:

```
mapping mapPickUpDropOff
{
        let T-> A: MakeRental::pickUpBranch():Branch->MakeRental ::pickUpId,
                MakeRental::dropOffBranch():Branch->MakeRental ::dropOffId
}

aspect PickUpDropOffBranch
{
        import_mapping mapPickUpDropOff
        pointcut PickDropBranch
        context T
        intro:
        post PickDropBranch:
        let branches:Set(Branch) = Branch.allInstances()-> select(name=self.A)
        in
        branches->size()=1 implies result=branches->any()
}
```

AspectOCL Constraint 3:

```
mapping mapUniqueID
{
        let T: {EU_RentPerson,    PendantCarOrder}
}

aspect UniqueAttributes
{
        import_mapping mapUniqueID
        pointcut uniqueID
        context T :: idIsKey () : Boolean
```

intro:

post uniqueID

result=T. allInstances()->**select**(t|t.id=self.id)->**size()**=1

}

AspectOCL Constraint 4:

mapping mapReservationStatus

{

let T->{S,A}:DemandXModel:: demand() : Integer ->

{CarModel ::requestedModel , ExistingCarModel::carModel},

DemandXGroup :: demand() : Integer ->

{CarGroup ::requestedGroup , ExistingCarGroup::carGroup},

DemandXModel :: demand() : Integer ->

{BikeModel::requestedModel , ExistingBikModel::BikeModel},

DemandXGroup :: demand() : Integer ->

{BikeGroup::requestedGroup , ExistingBikeGroup::BikeGroup}

}

aspect ReservationStatus

{

import_mapping mapReservationStatus

pointcut IntroduceNewConstraint:

context T

intro:

post IntroduceNewConstraint:

let pendantRes:Reservation= Reservation.allInstances()->

select(r|r.beginning.date()=tomorrow())->

select(r| r.pickUpBranch=self.branch and r.car-> **isEmpty()**)

in

result=pendantRes.S-> **select**(m|m=d.A)->**size()**

AspectOCL Constraint 5:

```
mapping mapOrder
{
        let T: {CarGroup   ,   RentalDuration}
}
aspect totalOrder
{
        import_mapping mapOrder
        pointcut IntroduceNewConstraint
        context T :: totalOrder (): Boolean
        intro:
        post IntroduceNewConstraint:
        let firstValue(w,b:T):Boolean= b.first=w or firstValue(w,b.first)
        let secondValue(b,w:T):Boolean=w.second=b or secondValue(b,w.second)
        in
                result= T.allInstances()->  one(t|t.first-> isEmpty()) and
                result=T.allInstances()->  one(t |t.second-> isEmpty()) and
                result= T.allInstances()->  forall(t1, t2| firstValue(t1, t2)
                implies not secondValue(t1, t2) and secondValue(t1, t2) implies
                notfirstValue(t1,t2))-> size() >0
}
```

AspectOCL Constraint 6:

```
mapping mapDuration
{
        let T ->{S,A,B}: ExistingRentalDuration::duration():RentalDuration  ->
                                {RentalDuration   ,   ExistingRentalDuration::durationName,
                                                      ExistingRentalDuration::durationLimit},
```

```
                    ExistingPerformanceIndicator::perfInd() : PerformanceIndicator  ->
                        {PerformanceIndicator  ,  ExistingPerformanceIndicator::name,
                                                ExistingPerformanceIndicator:: performanceLevel }
}
aspect  durationStatus
{
        import_mapping mapDuration
        pointcut IntroduceNewConstraint:
        context T
        intro:
        post IntroduceNewConstraint
        let temp:Set (S)=S.allInstances()-> select(var | var.name=self.A)
        in
        temp->notEmpty() implies result= temp-> any()
        and self. perf= B
}
```

AspectOCL Constraint 7:

```
mapping mapExisitngGroup
{
        let  S-> {A,B,TT} :  ExistingCar :: car() : Car  ->
                                { Car :: registrationNumber  ,  ExistingCar ::regNumber  ,  Car}
                        ExistingCarGroup  :: carG() : CarGroup  ->
                                { CarGroup ::name  ,  ExistingCarGroup ::carGroup  ,  CarGroup }
                        ExistingCarModel :: carM() : CarModel ->
                                { CarModel ::name  ,  ExistingCarModel ::carModel  ,  CarModel}
                        ExistingDiscount :: discount(): Discount ->
                                { Discount ::name  ,  ExistingDiscount ::discountName  ,  Discount}
                        ExistingRentalDuration:: duration() : RentalDuration ->
                                {RentalDuration:: name  ,  ExisitngRentalDuration::
                                                        durationName  ,  RentalDuration
                        ExistingPerformanceIndicator:: perfInd() : PerformanceIndicator ->
                                {PerformanceIndicator::name  ,  ExistingPerformanceIndicator::name,
                                                                PerformanceIndicator}
aspect ExistingGroup
```

```
{
        import_mapping mapExisitngGroup
        pointcut GroupConstraint
        context S
        intro:
        post GroupConstraint:
        let CarVal: Set(TT)=TT.allInstances()-> select(c|c.A = self.B )
        in
        CarVal ->notEmpty() implies result = CarVal ->any()
}
```

AspectOCL Constraint 8:

```
mapping mapPrice
{
        let T:  NewCarGroupDurationPrice:: apply(): CarGroup,
                NewCGDPForNewDuration :: apply() : CarGroup
}
aspect SetPrice
{
        import_mapping mapPrice
        pointcut PriceConstraint
        context T
        intro:
```

post PriceConstraint:

cgdp.**oclIsNew()** and cgdp.**oclIsTypeOf**(CarGroupDurationPrice)

and cgdp.price=self.price **and** cgdp.carGroup=self.carG **and** cgdp.rentalDuration=duration

}

AspectOCL Constraint 9:

mapping mapCarMove

{

let T -> {A, B}: RequestTransfer :: apply() : Boolean ->

{Branch :: otherBranch, Branch :: askingBranch}

DoTransfer :: apply() : Boolean ->

{ Branch :: askingBranch, Branch :: otherBranch}

}

aspect CorrectCarMove

{

pointcut IntroduceNewConstraint:

context T

intro:

post IntroduceNewConstraint:

self.**oclIsTypeOf**(MoveCars).^apply() **and** self.A.carsAvailable@pre->

intersection(self.otherBranch.car-> **select**(c|c.oclIsKindOf (BeingTransferredCar) **and**

c.**oclIsTypeOfType**(BeingTransferredCar).destination= self.B))->size()= movedCars

}