4 ALLOY

We have done this model referring to the Class Diagram. The file taxi.axl can be found on our repository (<https://github.com/MassimoSchiavo/MyTaxiService-Schiavo-Cittar>) . We have divided signature, fact, assertion and predicate. In the last part there is the metamodel created with alloy analyzer and a world created with the predicate show.

**4.1.1 SIGNATURE:**

**sig** Integer{}

**sig** Strings{}

enum Boolean{YES,NO}

**sig** TaxiQueue{

locations: **set** Location,

drivers:**set** Driver}

**sig** Driver{

available:**one** Boolean,

taxi:**one** Taxi,

confirmedRequest: **set** Request}

**sig** User{

requests: **set** Request}

**sig** Time{

hour:**one** Integer,

minute:**one** Integer}

**sig** Date{

day:**one** Integer,

month:**one** Integer,

year:**one** Integer}

**sig** Taxi{

code:**one** Integer,

actualPosition:**one** Location,

}

**sig** Location{

address:**one** Strings,

civicNumber:**one** Integer}

**sig** Notification{

message: **one** Strings,

user:**one** User,

request:**one** Request}

**4.1.2.ABSTRACT SIGNATURE**

**abstract** **sig** Request{

id:**one** Integer,

confirmed:**one** Boolean,

startLocation:**one** Location}

**4.1.3 IMPLEMENTATION OF ABSTRACT SIGNATURE**

**sig** SimpleRequest **extends** Request{}

**sig** DetailedRequest **extends** Request{

endLocation:**one** Location,

date:**one** Date,

time:**one** Time}

**4.1.4 FACT:** This is the fact part that defines the costraint of the class.

**fact** noEmptyDate{

**all** d:Date|(#d.day=1)**and**(#d.month=1)**and**(#d.year=1)

}

**fact** noEmpyTime{

**all** t:Time|(#t.hour=1) **and** (#t.minute=1)

}

**fact** noEmptyDriver{

**all** d:Driver|(#d.taxi=1)

}

**fact** noEmptyTaxi{

**all** t:Taxi|(#t.code=1) **and** (#t.actualPosition =1)

}

**fact** noEmptyRequest{

**all** r:Request|(#r.id=1) **and** (#r.startLocation=1) **and** (#r.confirmed=1)

}

**fact** noEmptyDetailedRequest{

**all** rd:DetailedRequest|(#rd.endLocation=1) **and** (#rd.date=1) **and** (#rd.time=1)

}

**fact** noEmptyLocation{

**all** l:Location|(#l.address=1) **and** (#l.civicNumber=1)

}

**fact** noEmptyTaxiQueue{

**all** tq:TaxiQueue|(#tq.locations>0)

}

//Every location is insert into one and only one TaxiQueue and a TaxiQueue is formed at least by one location

**fact** noDuplicatedTaxiQueue{

**all** l:Location|**no** **disj** tq1,tq2:TaxiQueue|(l **in** tq1.locations) **and** (l **in** tq2.locations)

**all** l:Location{**some** tx:TaxiQueue|l **in** tx.locations}

}

//Every taxi has only one driver and any driver drives only one taxi

**fact** oneDriverOneTaxi{

**all** d:Driver|**no** **disj** t1,t2:Taxi|(t1 =d.taxi) **and** (t2 = d.taxi)

**all** t:Taxi|**no** **disj** d1,d2:Driver| (t =d1.taxi) **and** (t = d2.taxi)

}

//Every request is unique and exits only if a user has done it

**fact** noDuplicatedRequest{

**no** **disj** r1,r2:Request|r1.id=r2.id

**all** r:Request|**no** **disj** u1,u2:User|(r **in** u1.requests) **and** (r **in** u2.requests)

**all** r:Request|**one** u:User| r **in** u.requests

}

//Every Taxi has different code between each other

**fact** noDuplicatedTaxi{

**no** **disj** t1,t2:Taxi|t1.code=t2.code

}

//All drivers available are isert into a queue in corrispondance of his taxi's location

//Drivers not available are not into a Taxi Queue

**fact** avaiableToQueue{

**all** d:Driver|d.available=YES => {**one** tq:TaxiQueue|(d **in** tq.drivers) **and** (d.taxi.actualPosition **in** tq.locations)}

**all** d:Driver|d.available=**NO** => {**all** tq:TaxiQueue|(d **not** **in** tq.drivers)}

}

//A notification exist only if the notified has done a request that has already been accepted

**fact** notify{

**all** n:Notification|**one** u:User|(u = n.user) **and**{**one** r:Request|(r **in** n.request) **and** (r **in** u.requests) **and** (r.confirmed=YES)}

**all** r:Request|(r.confirmed=YES)=>**one** u:User|r **in** u.requests **and**{**one** n:Notification|(n.user=u) **and** (n.request=r)}

}

//StartLocation and endLocation of a detailed request must be two different place

**fact** diffLocation{

**all** dt:DetailedRequest|dt.startLocation!=dt.endLocation

}

//Two Taxi Queue with a shared driver must not exist

**fact** oneQueuePerDriver{

**all** d:Driver|**no** **disj** tq1,tq2:TaxiQueue|(d **in** tq1.drivers) **and** (d **in** tq2.drivers)

}

//driver can accept request only if are in taxiqueue in which the startLocation is included

**fact** Acceptance{

**all** r:Request|r.confirmed=YES <=> {**one** tq:TaxiQueue|(r.startLocation **in** tq.locations)**and**{**one** d:Driver|(d **in** tq.drivers) **and**(r **in** d.confirmedRequest)}}

**all** r:Request|r.confirmed=NO <=> {**one** tq:TaxiQueue|(r.startLocation **in** tq.locations)**and**{**all** d:Driver|(d **in** tq.drivers) **and** (r **not** **in** d.confirmedRequest)}}

**all** r:Request|**no** **disj** d1,d2:Driver|(r **in** d1.confirmedRequest) **and** (r **in** d2.confirmedRequest)

}

**4.1.5 ASSERTION** These are the asserts used to verify the model.

//Every location can't be insert into two different taxis queues

**assert** LocationI**none**TaxiQueue{

**all** l:Location|**no** **disj** tq1,tq2:TaxiQueue|(l **in** tq1.locations) **and** (l **in** tq2.locations)

}

**check** LocationI**none**TaxiQueue **for** 10

//Once a request is confirmed by a driver it cannot be confirmed by anyone anymore

**assert** RequestsConfirmedByOnlyOneDriver{

**all** d1,d2:Driver|**no** r:Request|(d1!=d2) **and** (r **in** d1.confirmedRequest) **and** (r **in** d2.confirmedRequest)

}

**check** RequestsConfirmedByOnlyOneDriver **for** 10

//Every request is unique and belongs to only one user

**assert** oneOwner{

**all** u1,u2:User|**no** r:Request|(u1!=u2) **and** (r **in** u1.requests) **and** (r **in** u2.requests)

}

**check** oneOwner **for** 10

//Verify acceptance fact.

**assert** Acceptance{

**all** sr:SimpleRequest|lone d:Driver|(sr.confirmed=YES) **and** (sr **in** d.confirmedRequest)

**all** r:Request|**no** d:Driver|(r.confirmed=NO) **and** (r **in** d.confirmedRequest)

**all** r:Request|r.confirmed=YES implies **one** tq:TaxiQueue|(r.startLocation **in** tq.locations) **and** {**one** d:Driver|(d **in** tq.drivers) **and** (r **in** d.confirmedRequest) **and** (d.taxi.actualPosition **in** tq.locations)}

}

**check** Acceptance **for** 5

//Verify Notification fact.

**assert** Notification{

**no** n:Notification|n.request.confirmed=**NO** **and** n.user=**none**

}

**check** Notification **for** 6

//Verify available to Queue

**assert** availableToQueue{

**all** tq:TaxiQueue|**no** d:Driver|(d **in** tq.drivers) **and** (d.available=NO)

**all** tq:TaxiQueue|**all** d:Driver|(d **in** tq.drivers)=>(d.available=YES)

}

**check** availableToQueue **for** 6

//Verify DiffLocation fact.

**assert** DiffLocation{

**all** dt:DetailedRequest|**no** l:Location|dt.startLocation=l **and** dt.endLocation=l

}

**check** DiffLocation **for** 6

//Verify beAvailable pred.

**assert** beAvailable{

**all** d:Driver|(d.available=**NO**) **and** beAvailable[d] implies (d.available=YES)

}

**check** beAvailable **for** 6

//Verify accept pred.

**assert** accept{

**all** d1,d2:Driver|**all** r:Request| ((d1=d2) **and**(d1.available=YES) **and** (r.confirmed=NO) **and** accept[d1,d2,r])implies((r **in** d2.confirmedRequest) **and** d2.available=NO **and** r.confirmed=YES)

}

**check** accept **for** 6

**4.1.6 PREDICATES** These are the predicates used with the previous assert to verify the model.

**pred** beAvailable(d:Driver){

(d.available=NO) implies (d.available=d.available - NO + YES)

}

**run** beAvailable **for** 6

**pred** accept(d1,d2:Driver,r:Request){

(r **not** **in** d1.confirmedRequest **and** d1.available=YES **and** d1=d2) implies (d2.confirmedRequest=d1.confirmedRequest+r **and** r.confirmed=YES **and** d2.available=d1.available-YES+NO )

}

**run** accept **for** 4

**pred** addSimpleRequest(u1,u2:User,sr:SimpleRequest){

(sr **not** **in** u1.requests) implies (u2.requests=u1.requests+sr)

}

**run** addSimpleRequest **for** 4

**pred** addDetailedRequest(u1,u2:User,dt:DetailedRequest){

(dt **not** **in** u1.requests) implies (u2.requests=u1.requests+dt)

}

**run** addDetailedRequest **for** 4

**pred** show(){

#TaxiQueue>1

#User>1

#Driver>1

#Taxi>1

#Request>1

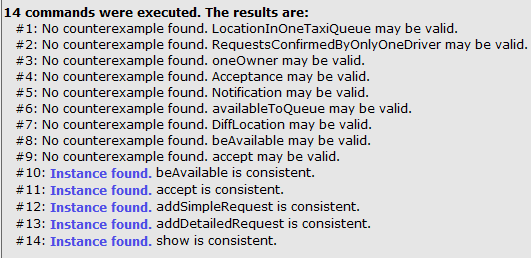
#Location>1

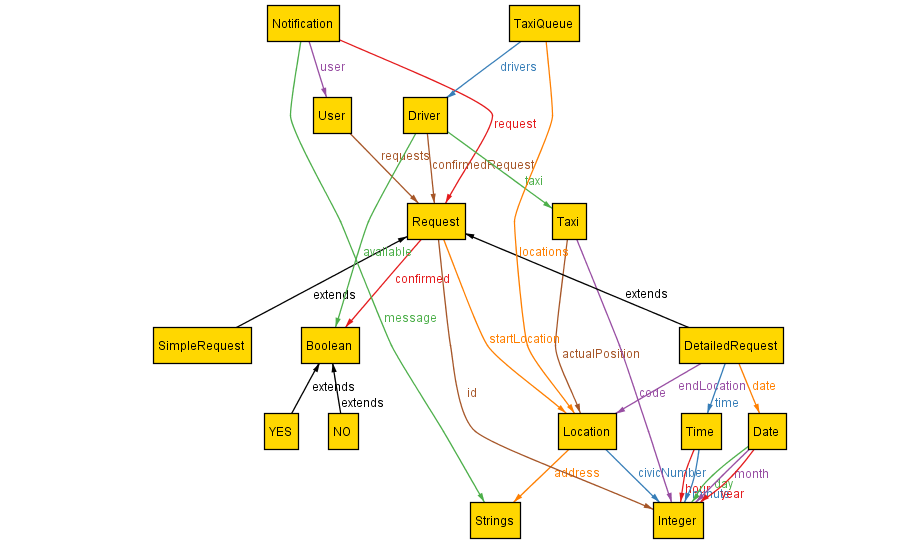
}

**run** show **for** 4

**4.1.7 RESULTS and METAMODEL**

Results obtained with the alloy analyzer that show the consistency of the model in all parts.





A screenshot that represents the metamodel created.

**4.1.8 Generated world**

The first screeshot represent the world generated with the alloy analyzer using the predicate show() for 2 case. The second using the predicate show() for 2 case with exactly five Location. The third instead using show for 4 case with exactly two SimpleRequest and nine Location. With more cases the scheme would become impossible to read and understand for the presence of too many arrows so we decided not to put more cases in this document.