Model Driven Security:
from UML Models to Access Control Infrastructures

Prof. David Basin
Jürgen Doser
Torrsten Lodderstedt

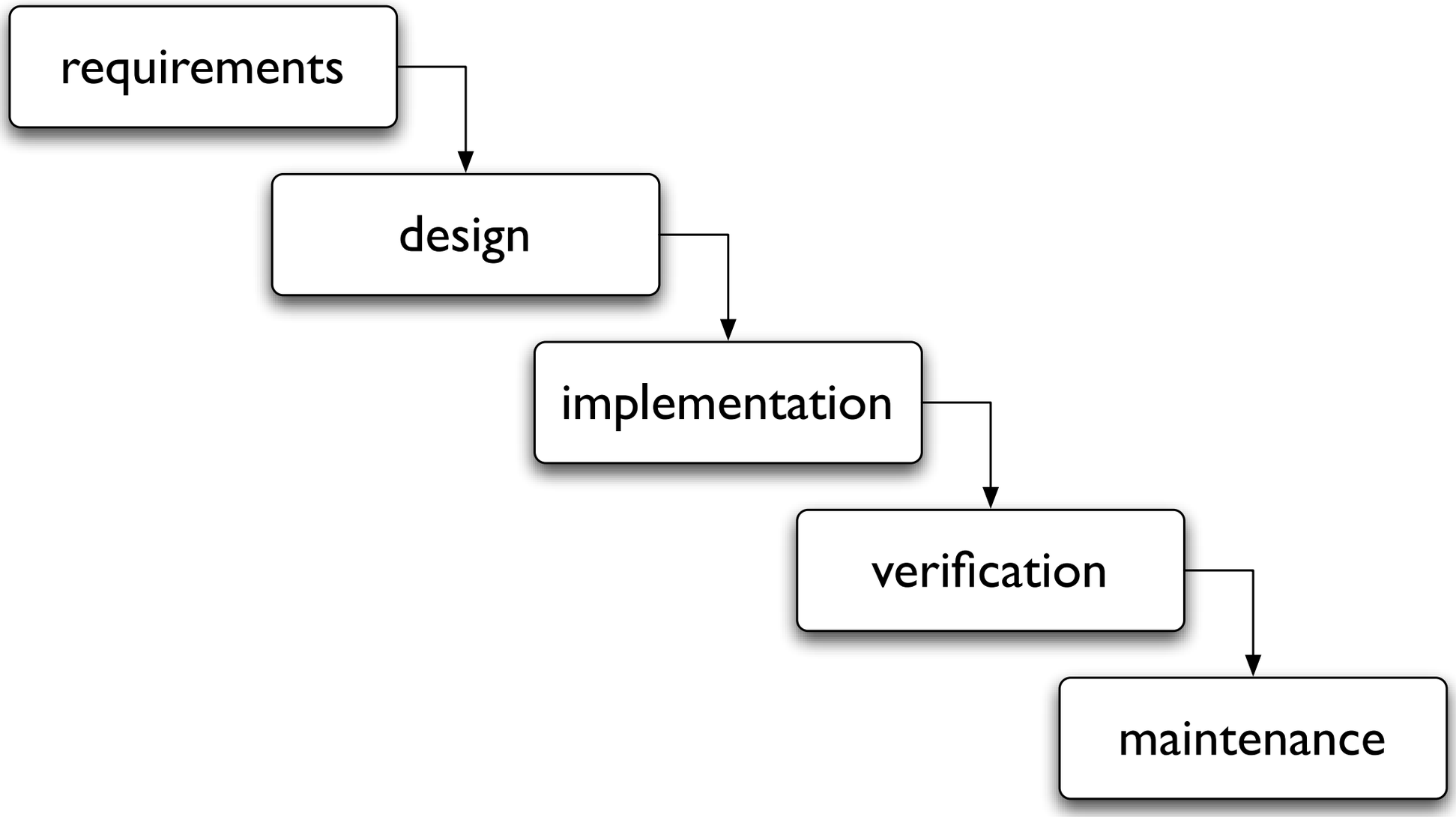
Supervisor: Raphael Eidenbenz

Jan-Filip Zagalak

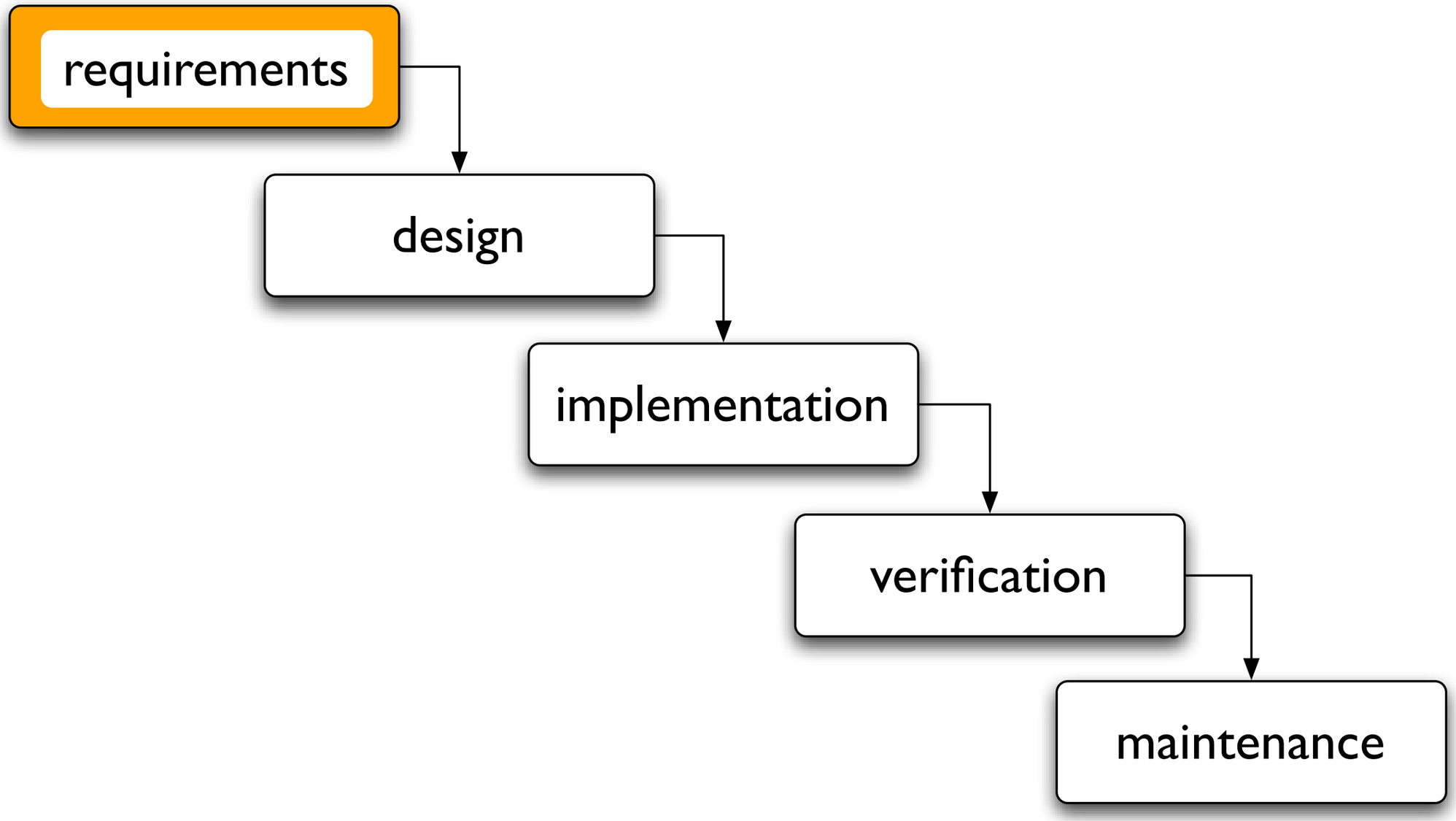
outline:

- problem domain / problem solving
- approach
- example
- bottom line

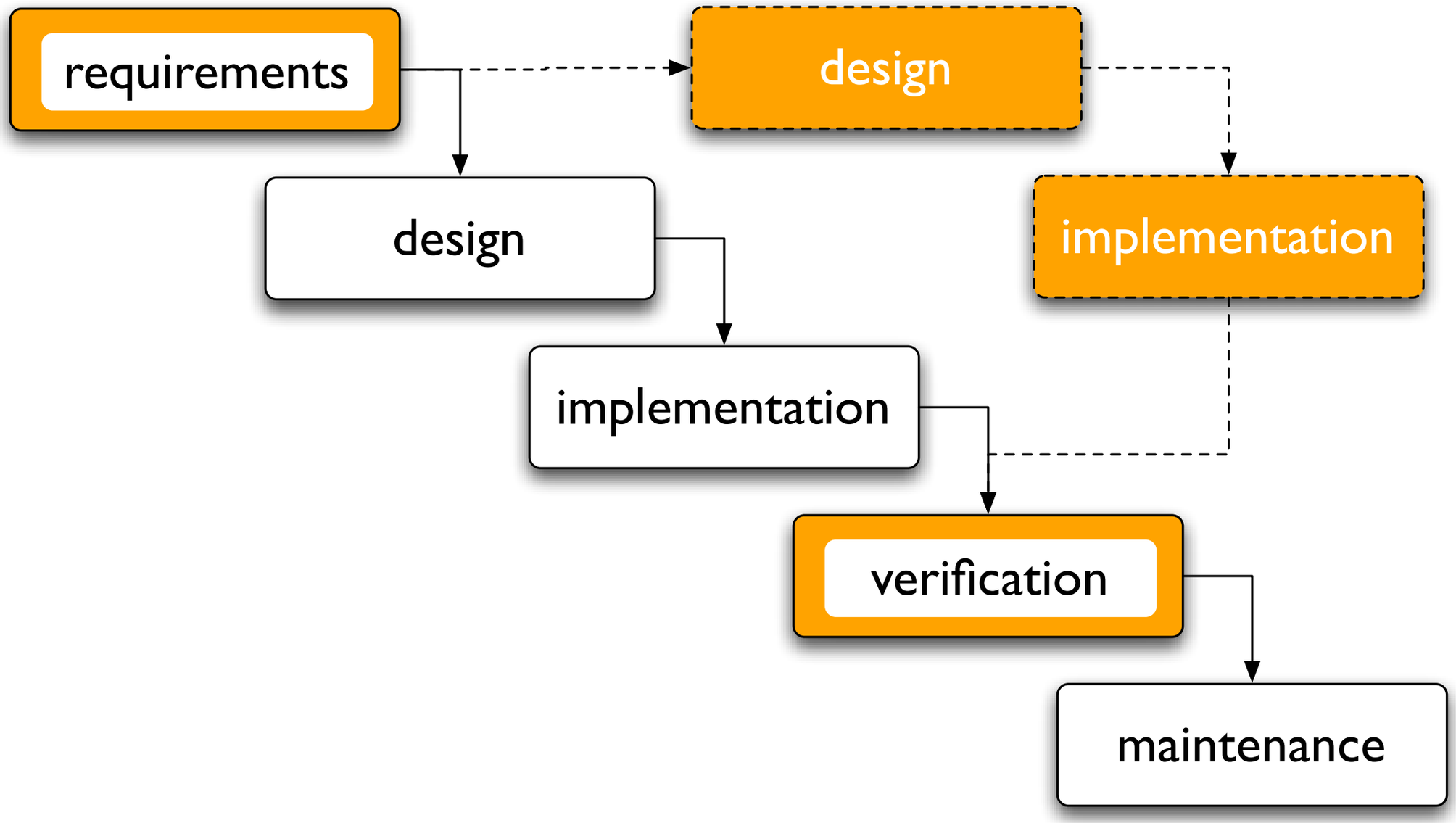
common software engineering process



security requirements



security requirements



development of security requirements

- very late ad hoc integration of implemented security mechanisms
- hard to keep track of security requirements through development

➡ different representations of system / security

problem solving

- one representation for system and security
- manual implementation is ambiguous: remove ambiguity

outline:

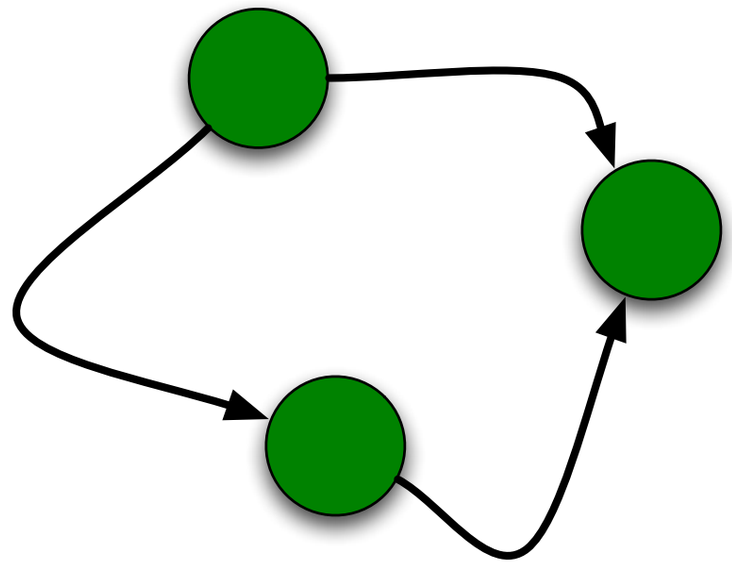
- problem domain / problem solving
- approach
- example
- bottom line

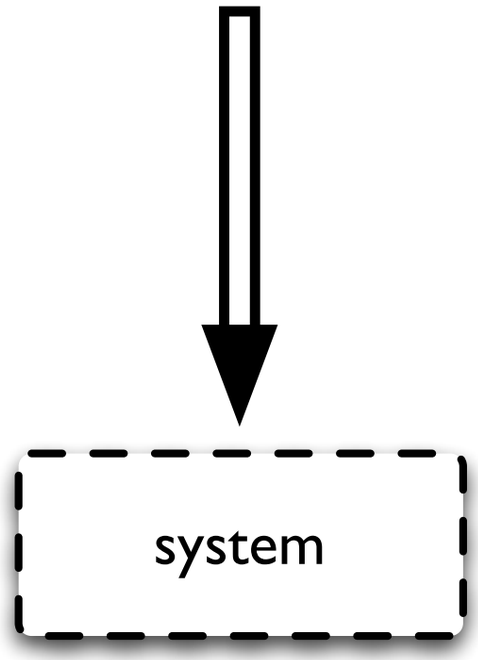
MDA: Model Driven Architecture

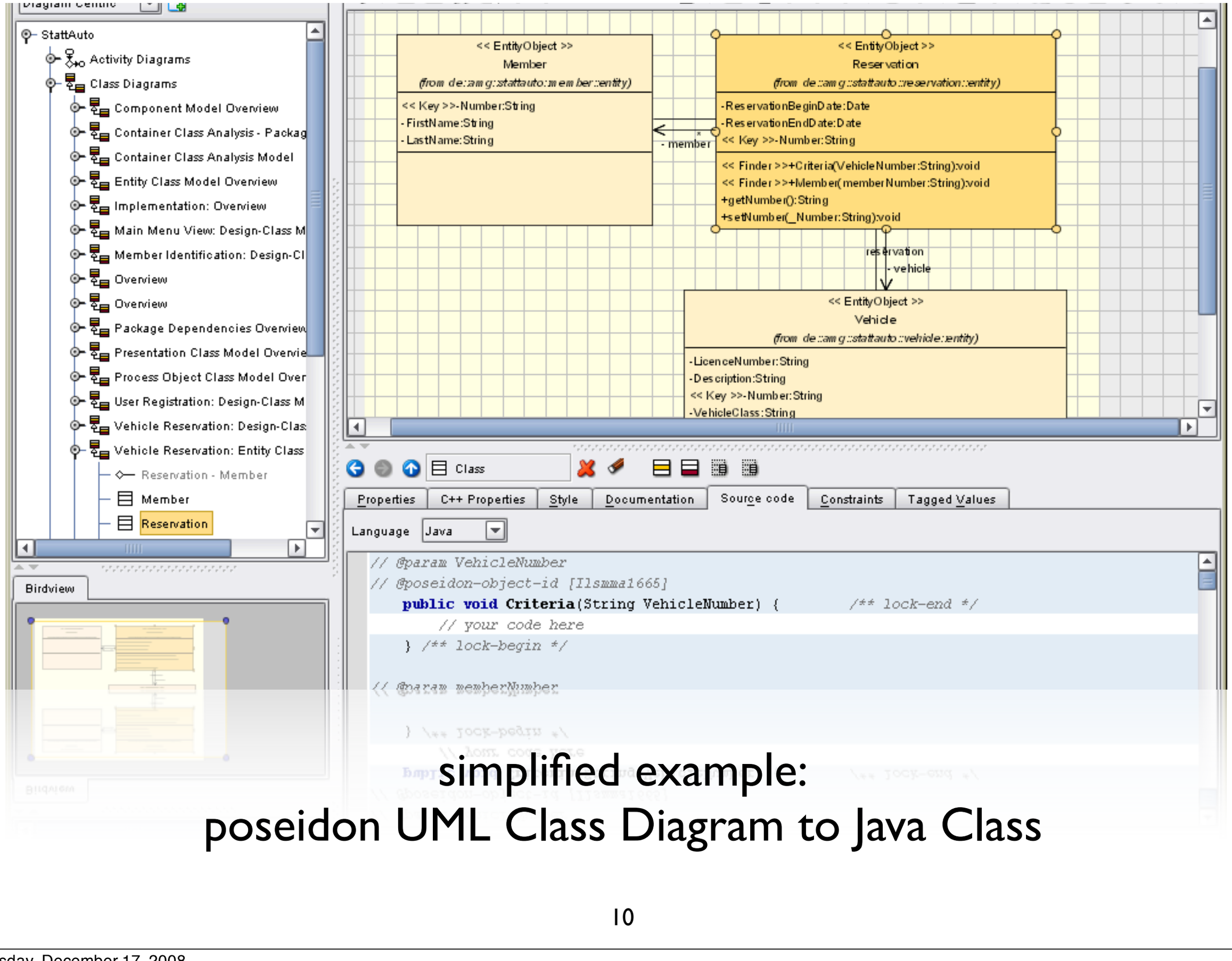
- specify system in abstract model

- apply transformation functions

- result:
  system specified in target platform
  e.g. EJB, .NET ...
  (only architecture, no business logic)



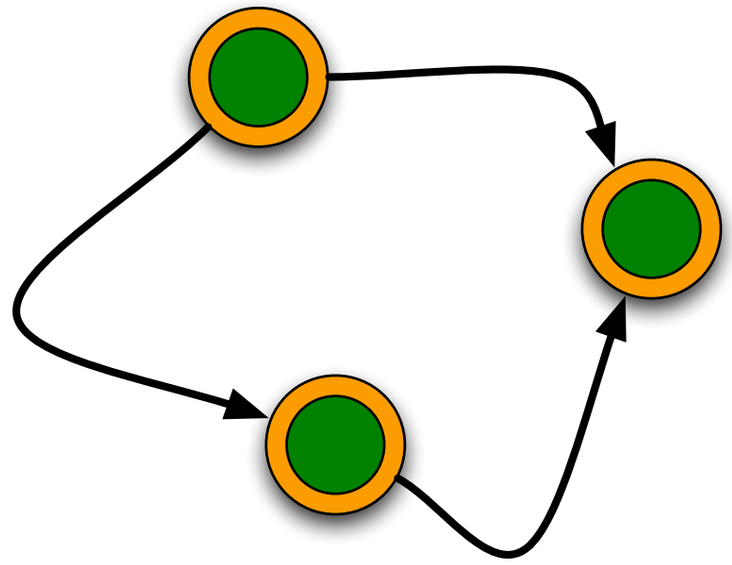




simplified example: poseidon UML Class Diagram to Java Class

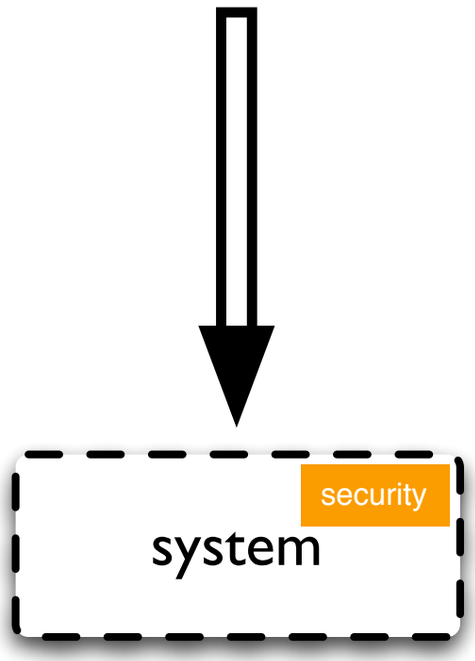
MDS: Model Driven Security

- specify system and security together in an abstract model



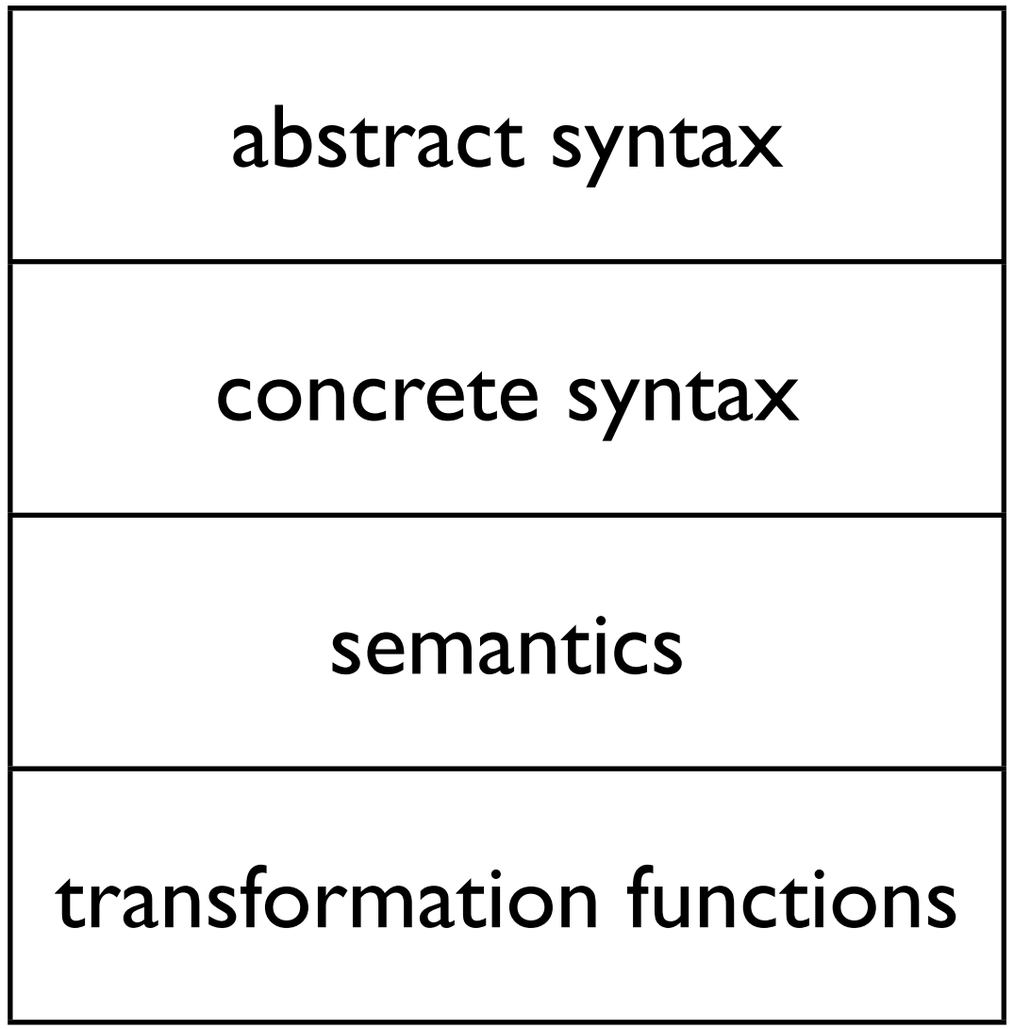
- apply transformation functions

- result:
  security aware system
  specified in target platform
  e.g. EJB, .NET ...
  (only architecture, no business logic)



... but how to build a model?

- modeling language



... but how to build a modeling language for MDS?

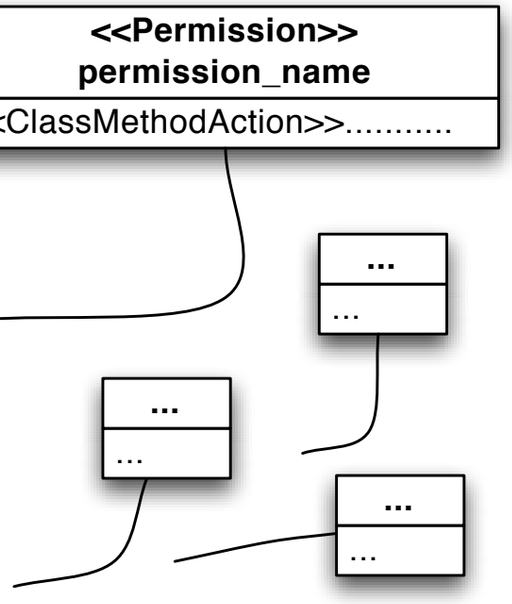
modeling language combination schema

modeling language combination schema

system design modeling language

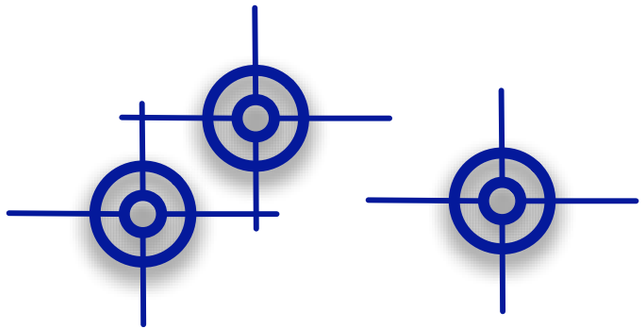
modeling language combination schema

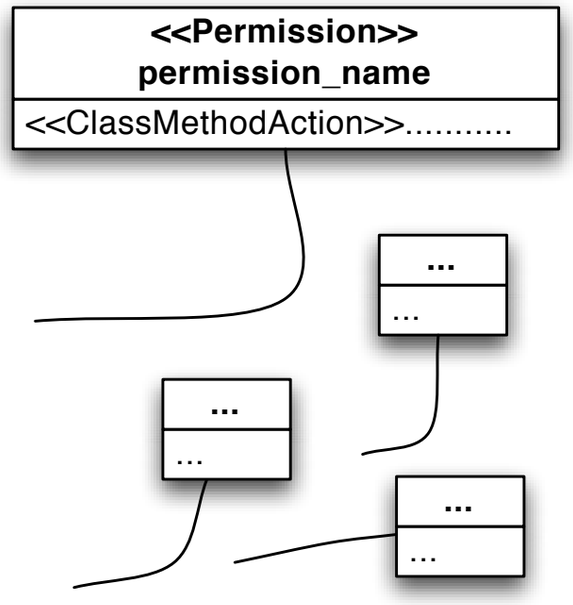
system design modeling language



security modeling language

modeling language combination schema



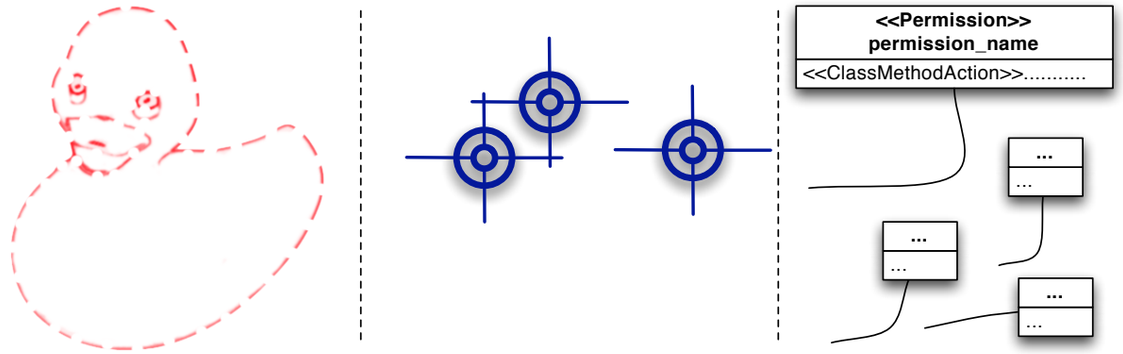


system design modeling language

dialect

security modeling language

modeling language combination schema



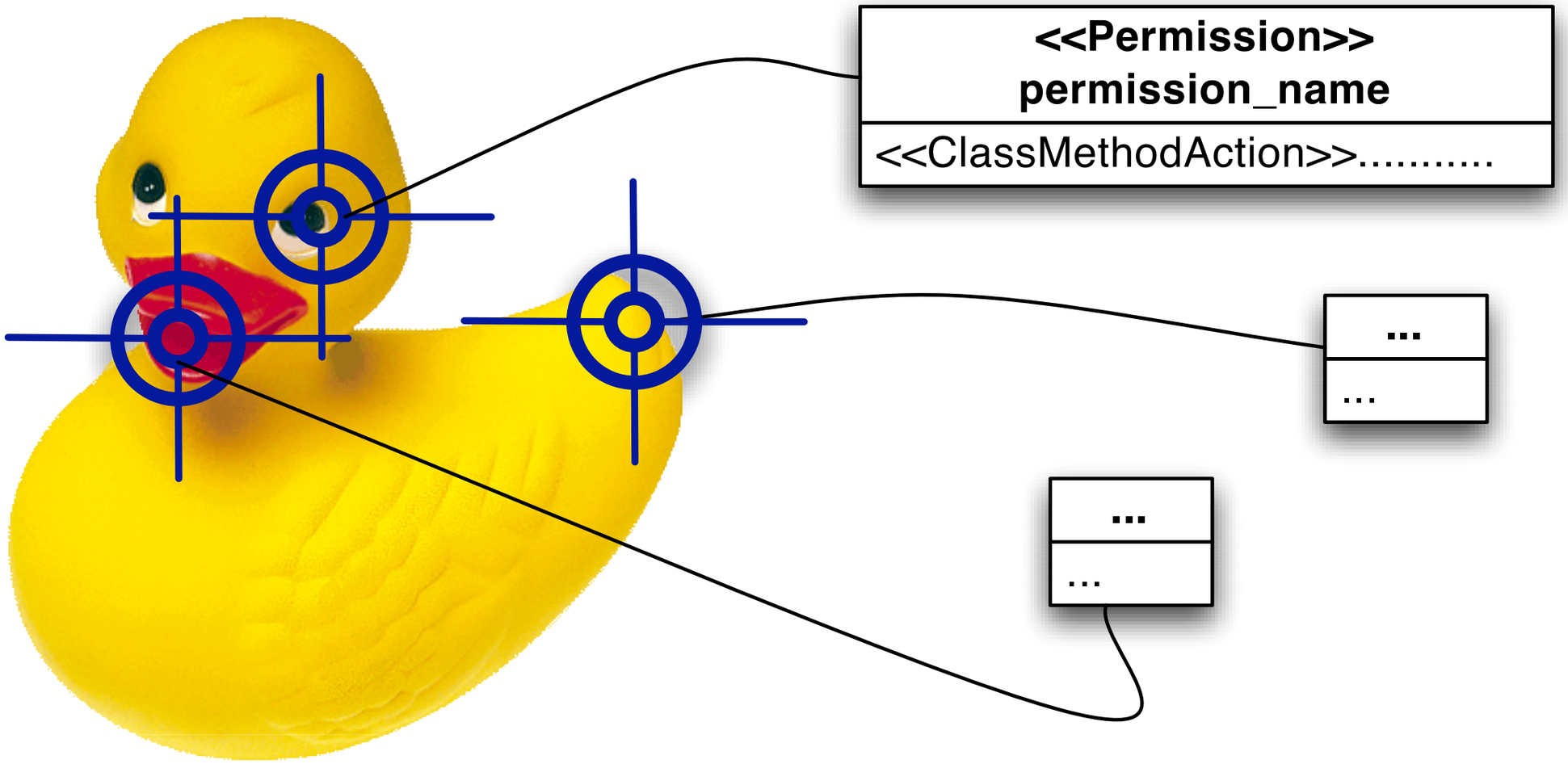
system design modeling language

dialect

security modeling language

security design language

modeling language combination schema



system and security
modeled with security design language

outline:

- problem domain / problem solving
- approach
- example
- bottom line

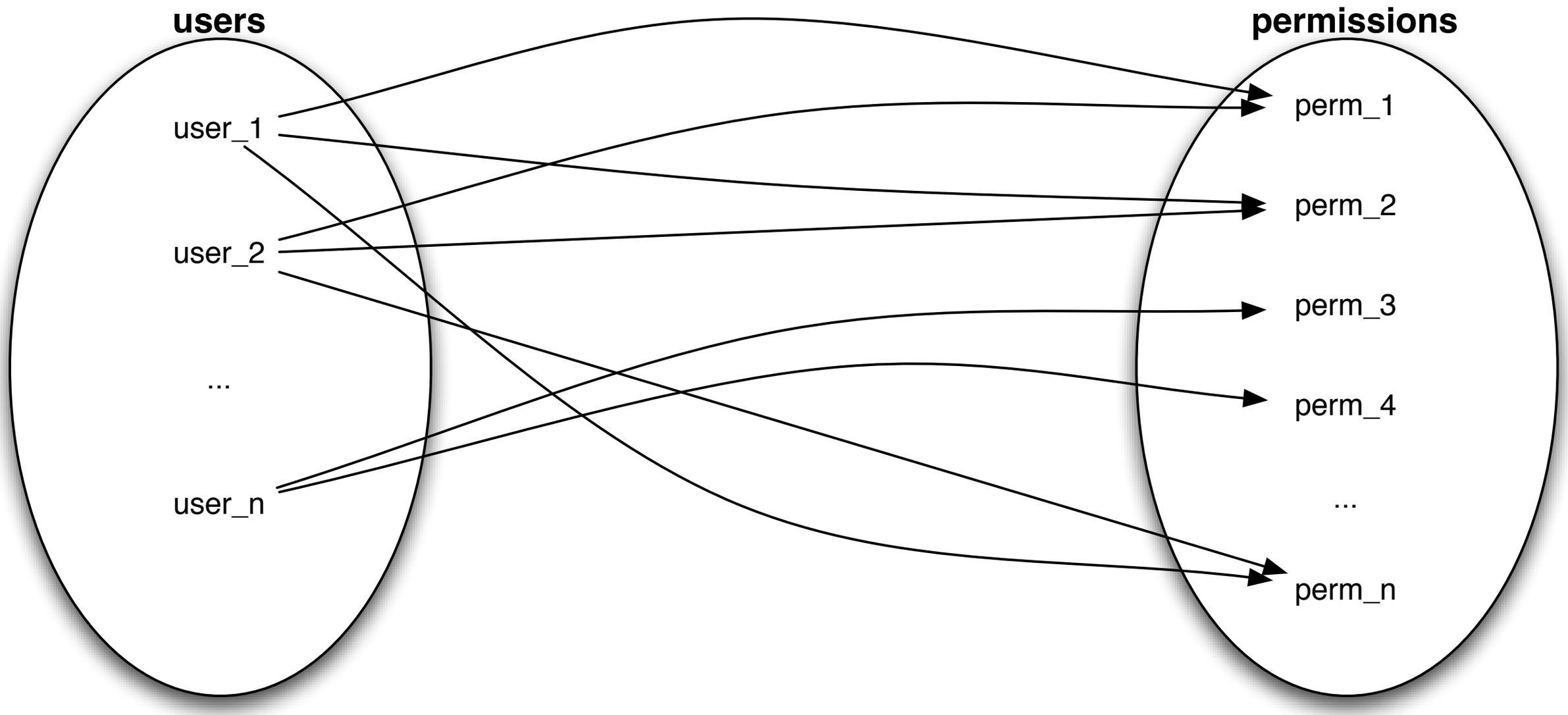
example

21

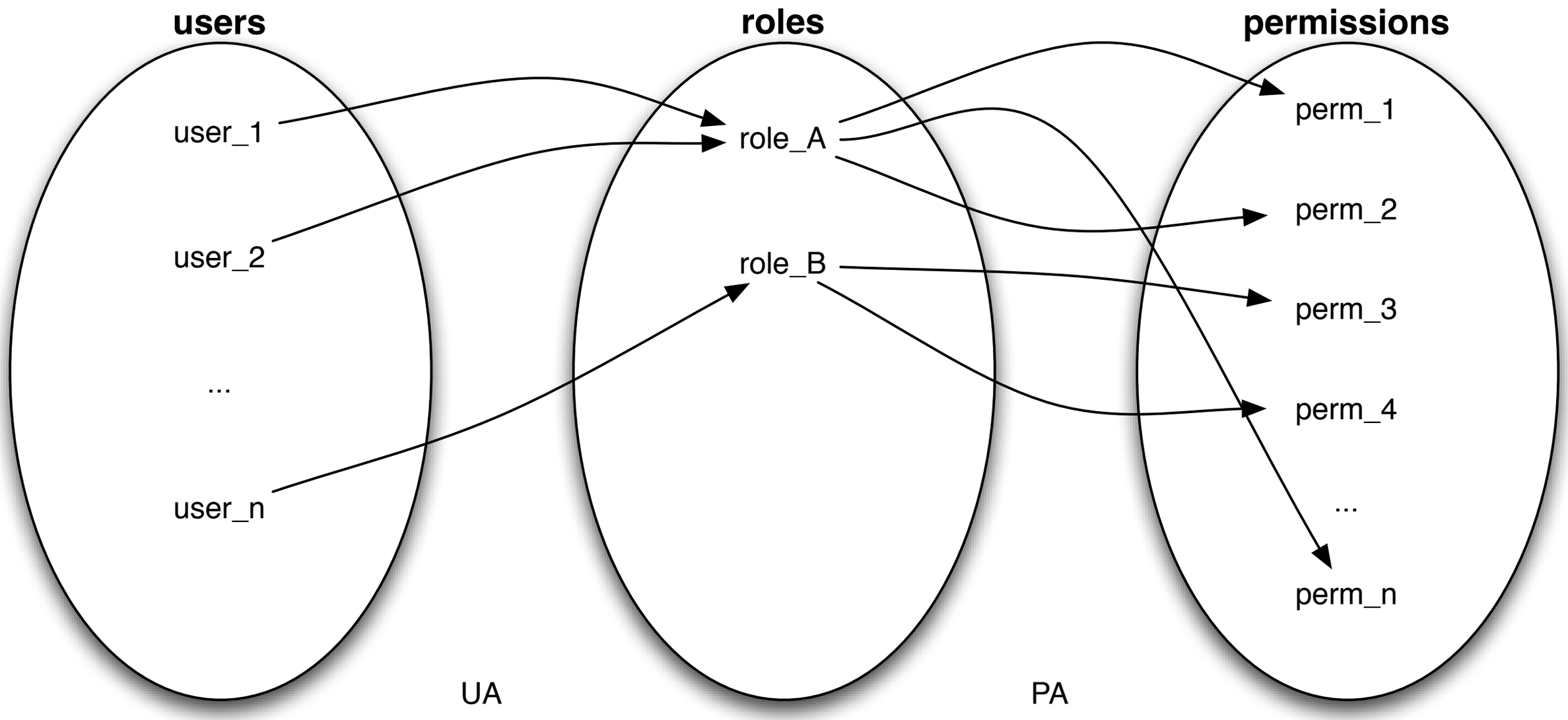
Briefing with "M"

- I need mi6 to get a new system
- I like my cars: protect them with RBAC
- I want everything deployed as EJBs

Role Based Access Control



Role Based Access Control



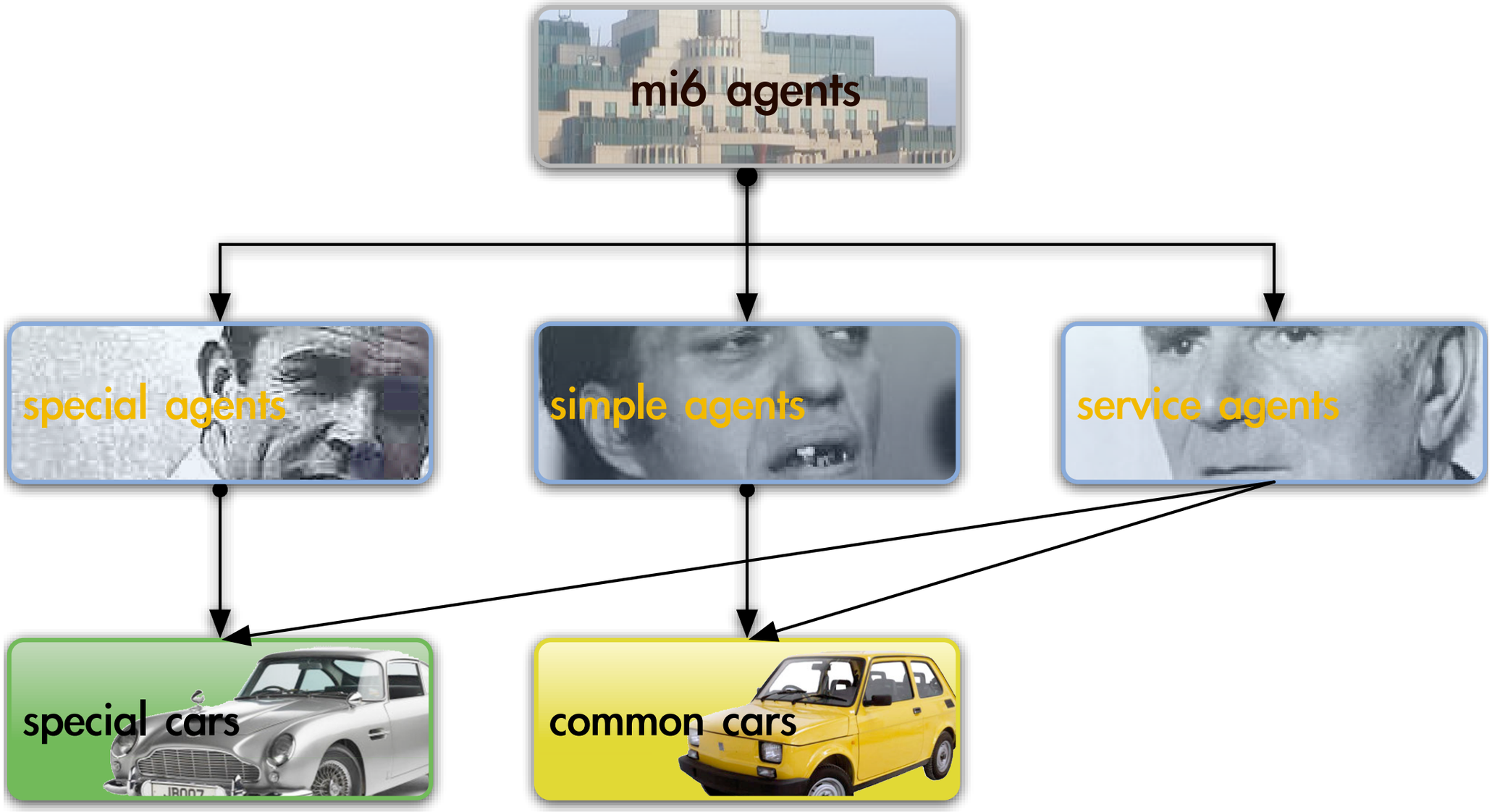
EJB: Enterprise Java Beans

- **Enterprise JavaBeans™** (**EJB**) is a managed, server-side component architecture for modular construction of enterprise applications.

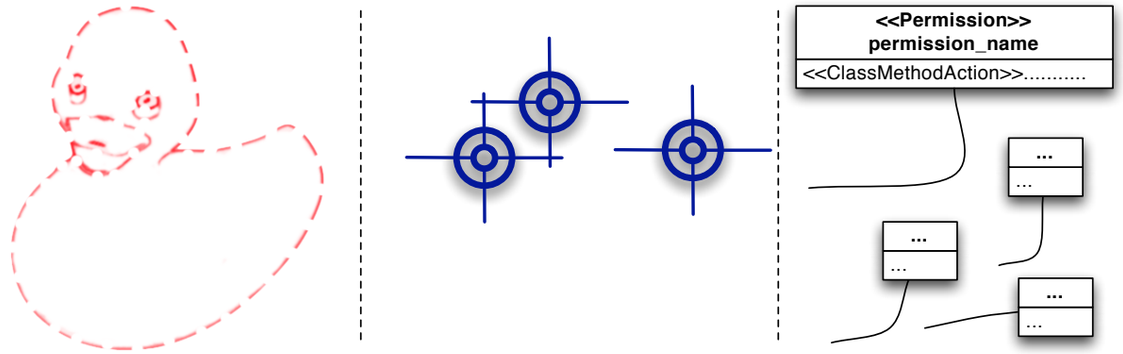
EJB: Security - role based access control

```
<method-permission>
    <role-name>employee</role-name>
    <method>
        <ejb-name>AardvarkPayroll</ejb-name>
        <method-name>findByPrimaryKey</method-name>
    </method>

    <method>
        <ejb-name>AardvarkPayroll</ejb-name>
        <method-name>getEmployeeInfo</method-name>
    </method>

    <method>
        <ejb-name>AardvarkPayroll</ejb-name>
        <method-name>updateEmployeeInfo</method-name>
    </method>
</method-permission>
```

mi6 - car access policy



modeling language combination schema



system design modeling language

dialect

security modeling language

security design language

modeling language combination schema

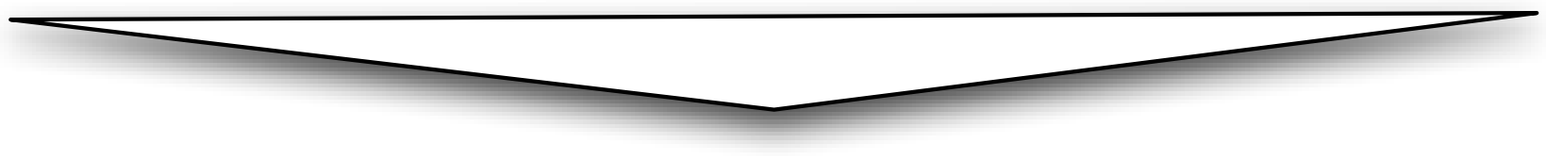
mi6UML

SecureUML

system design modeling language

dialect

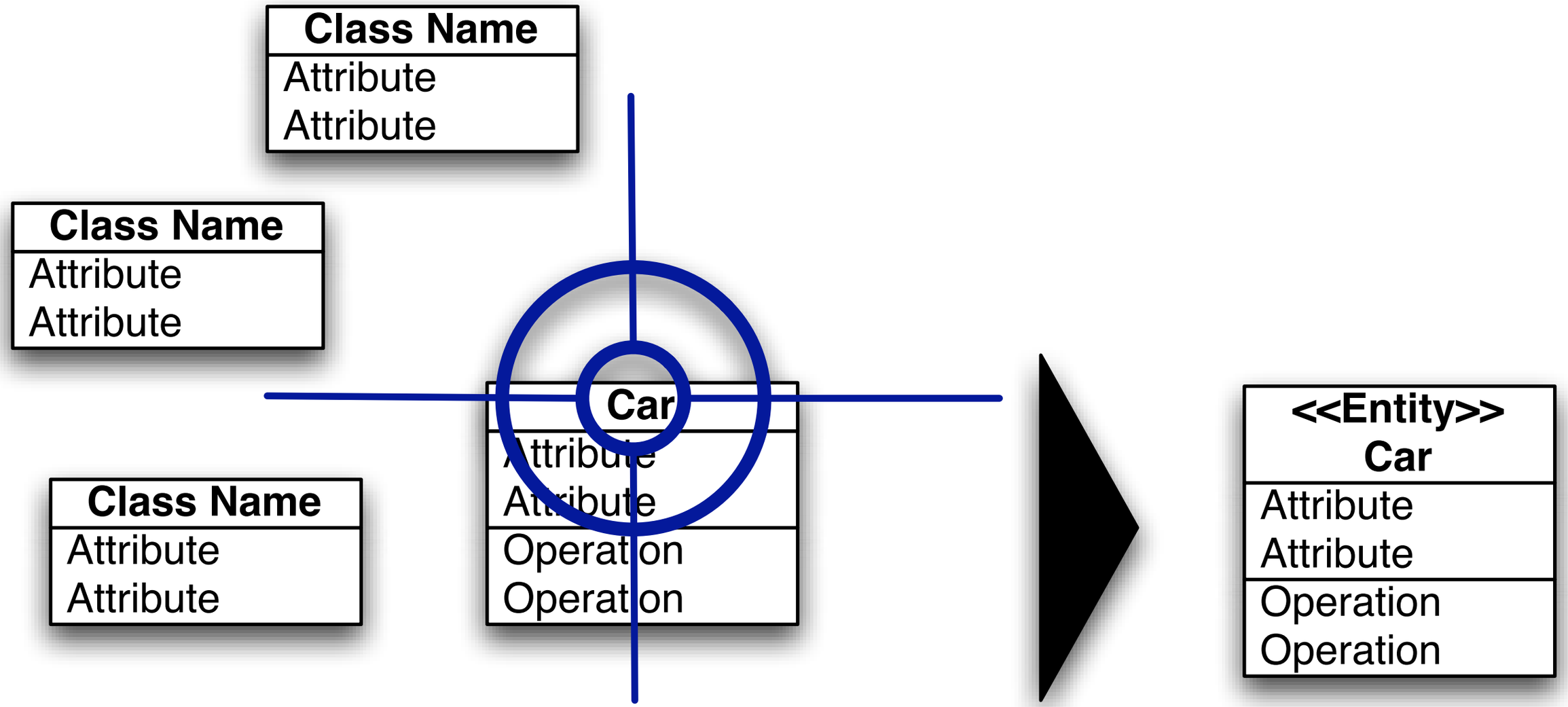
security modeling language



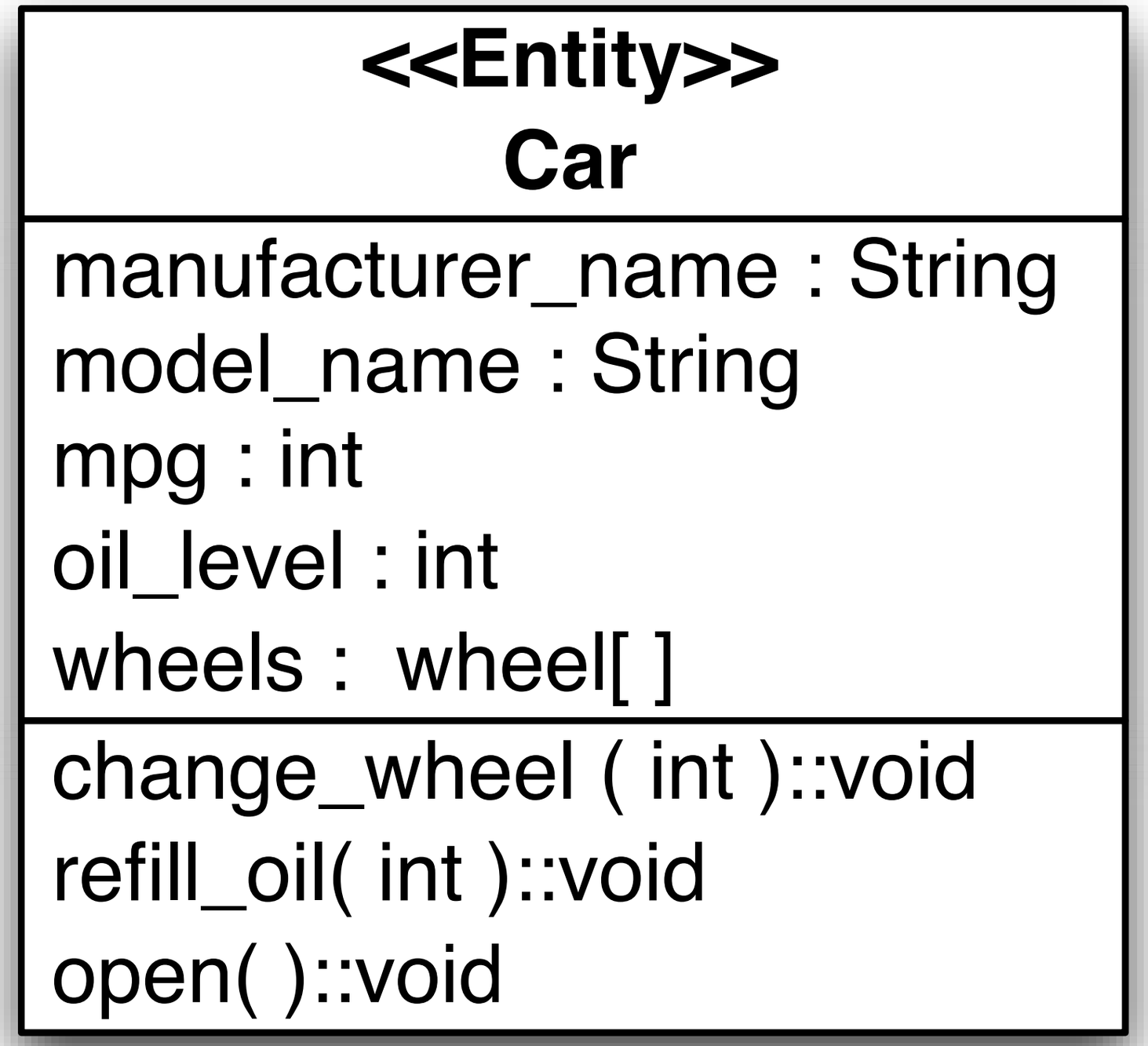
security design language

Securemi6UML

system: protected resources

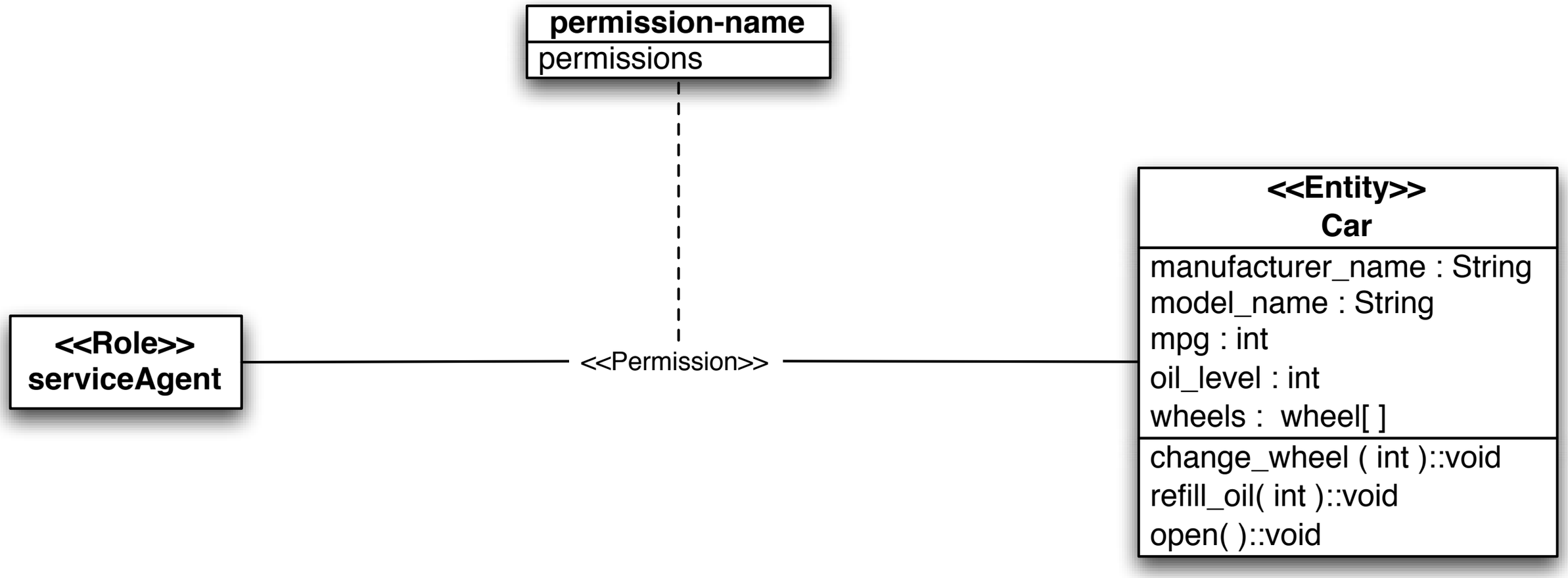


a car modeled with Securemi6UML

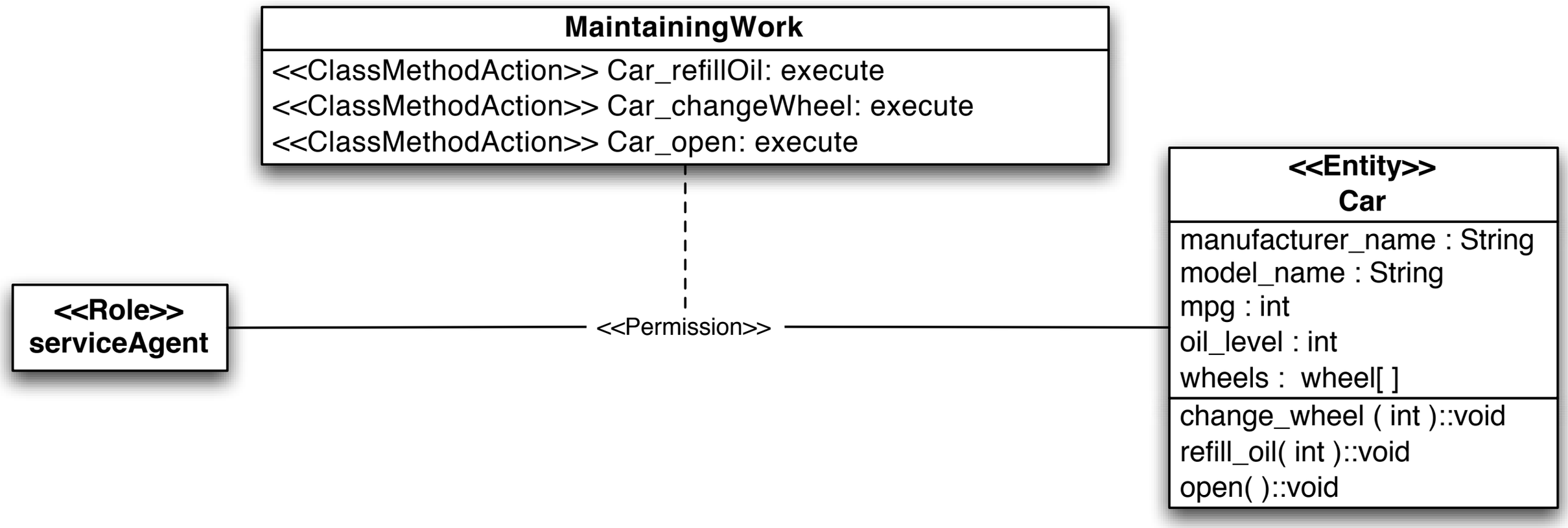


role and entity

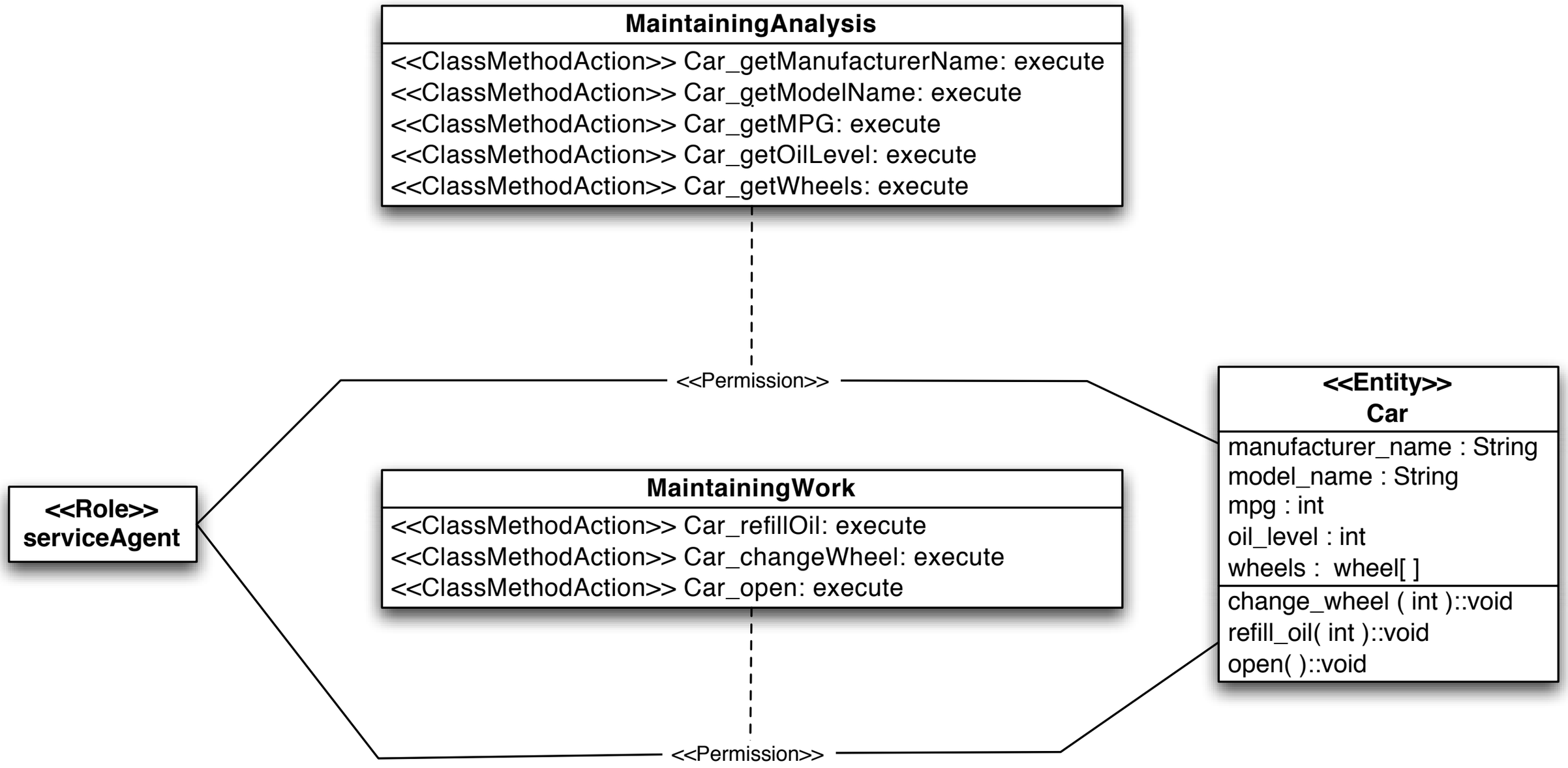
permissions as association class



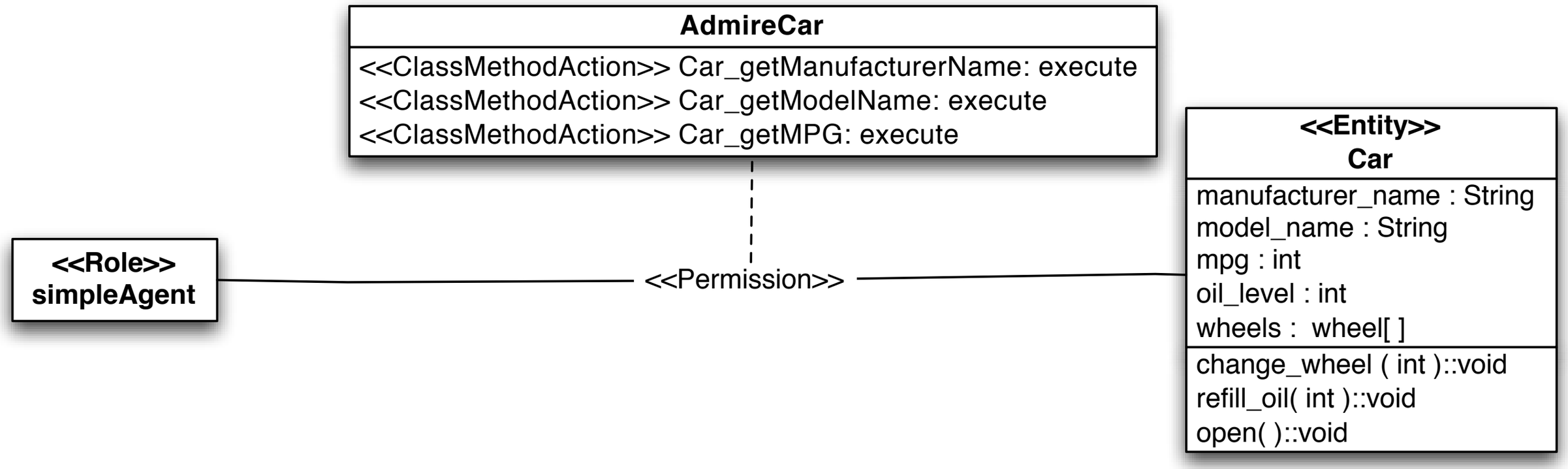
role: serviceAgent - permission 1



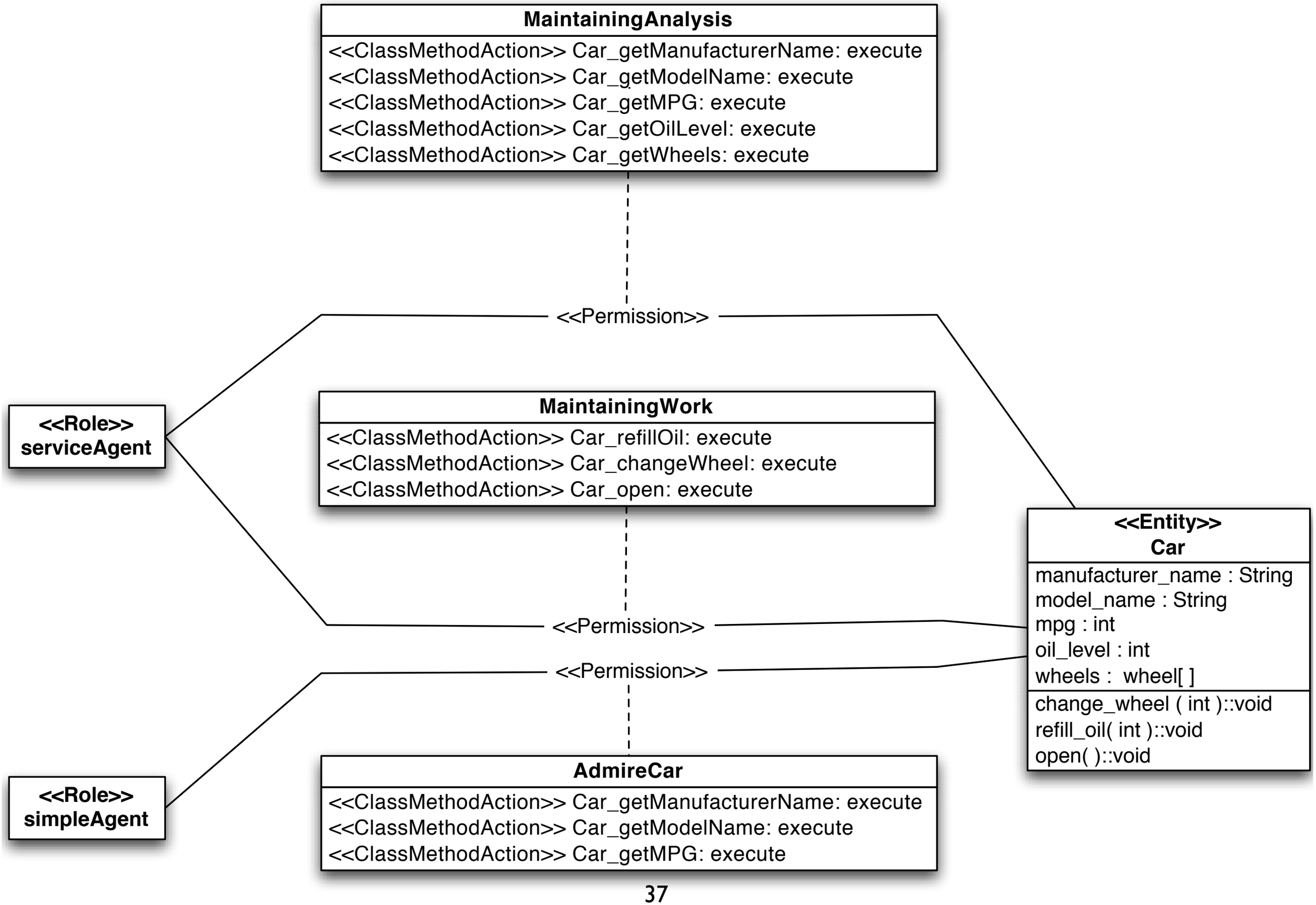
role: serviceAgent - permission II



role: simpleAgent - permission I



model

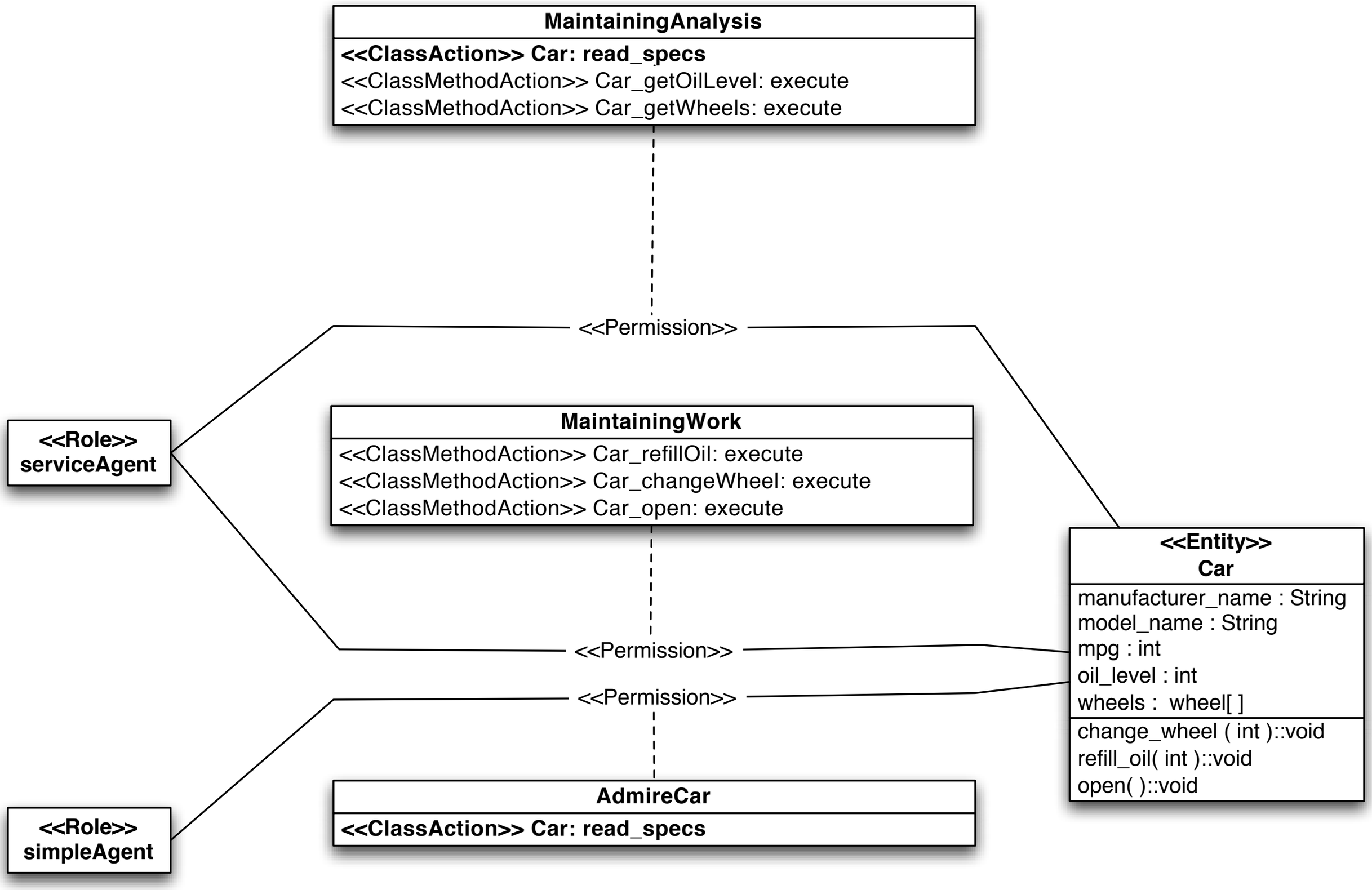

MaintainingAnalysis
<<ClassMethodAction>> Car_getManufacturerName: execute
<<ClassMethodAction>> Car_getModelName: execute
<<ClassMethodAction>> Car_getMPG: execute
<<ClassMethodAction>> Car_getOilLevel: execute
<<ClassMethodAction>> Car_getWheels: execute
<<Permission>>
<<Role>>
serviceAgent
MaintainingWork
<<ClassMethodAction>> Car_refillOil: execute
<<ClassMethodAction>> Car_changeWheel: execute
<<ClassMethodAction>> Car_open: execute
<<Entity>>
Car
manufacturer_name : String
model_name : String
mpg : int
oil_level : int
wheels : wheel[ ]
change_wheel (int)::void
refill_oil(int)::void
open( )::void
<<Permission>>
<<Permission>>
<<Role>>
simpleAgent
AdmireCar
<<ClassMethodAction>> Car_getManufacturerName: execute
<<ClassMethodAction>> Car_getModelName: execute
<<ClassMethodAction>> Car_getMPG: execute

CompositeAction

CompositeAction: read_specs

<<ClassMethodAction>> Car_getManufacturerName: execute
<<ClassMethodAction>> Car_getModelName: execute
<<ClassMethodAction>> Car_getMPG: execute

model


MaintainingAnalysis
<<ClassAction>> Car: read_specs
<<ClassMethodAction>> Car_getOilLevel: execute
<<ClassMethodAction>> Car_getWheels: execute
<<Permission>>
<<Role>>
serviceAgent
MaintainingWork
<<ClassMethodAction>> Car_refillOil: execute
<<ClassMethodAction>> Car_changeWheel: execute
<<ClassMethodAction>> Car_open: execute
<<Entity>>
Car
manufacturer_name : String
model_name : String
mpg : int
oil_level : int
wheels : wheel[ ]
change_wheel (int)::void
refill_oil(int)::void
open( )::void
<<Permission>>
<<Permission>>
AdmireCar
<<ClassAction>> Car: read_specs
<<Role>>
simpleAgent

CompositeAction

CompositeAction:
read_all

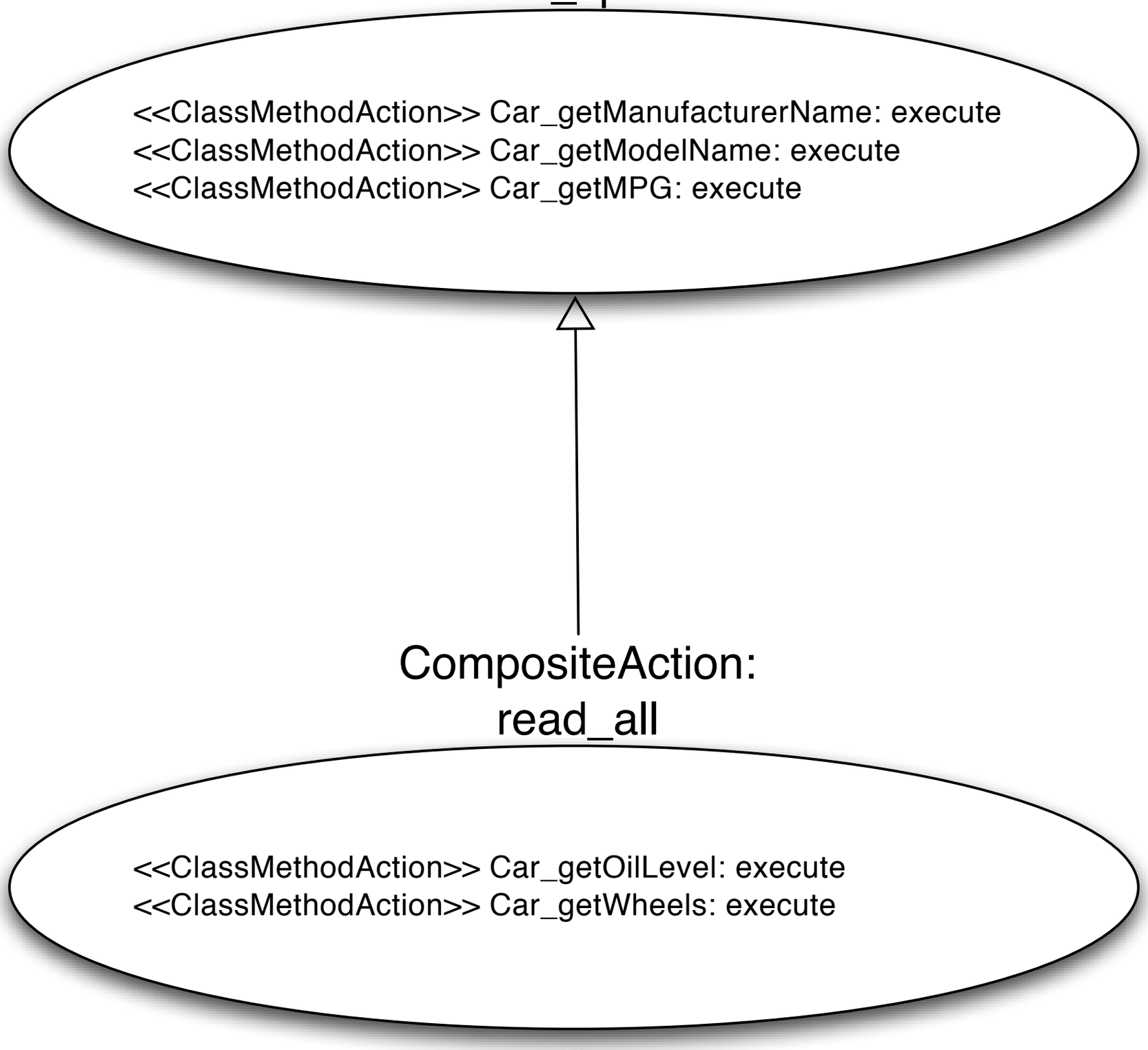
<<ClassMethodAction>> Car_getManufacturerName: execute
<<ClassMethodAction>> Car_getModelName: execute
<<ClassMethodAction>> Car_getMPG: execute
<<ClassMethodAction>> Car_getOilLevel: execute
<<ClassMethodAction>> Car_getWheels: execute

CompositeAction

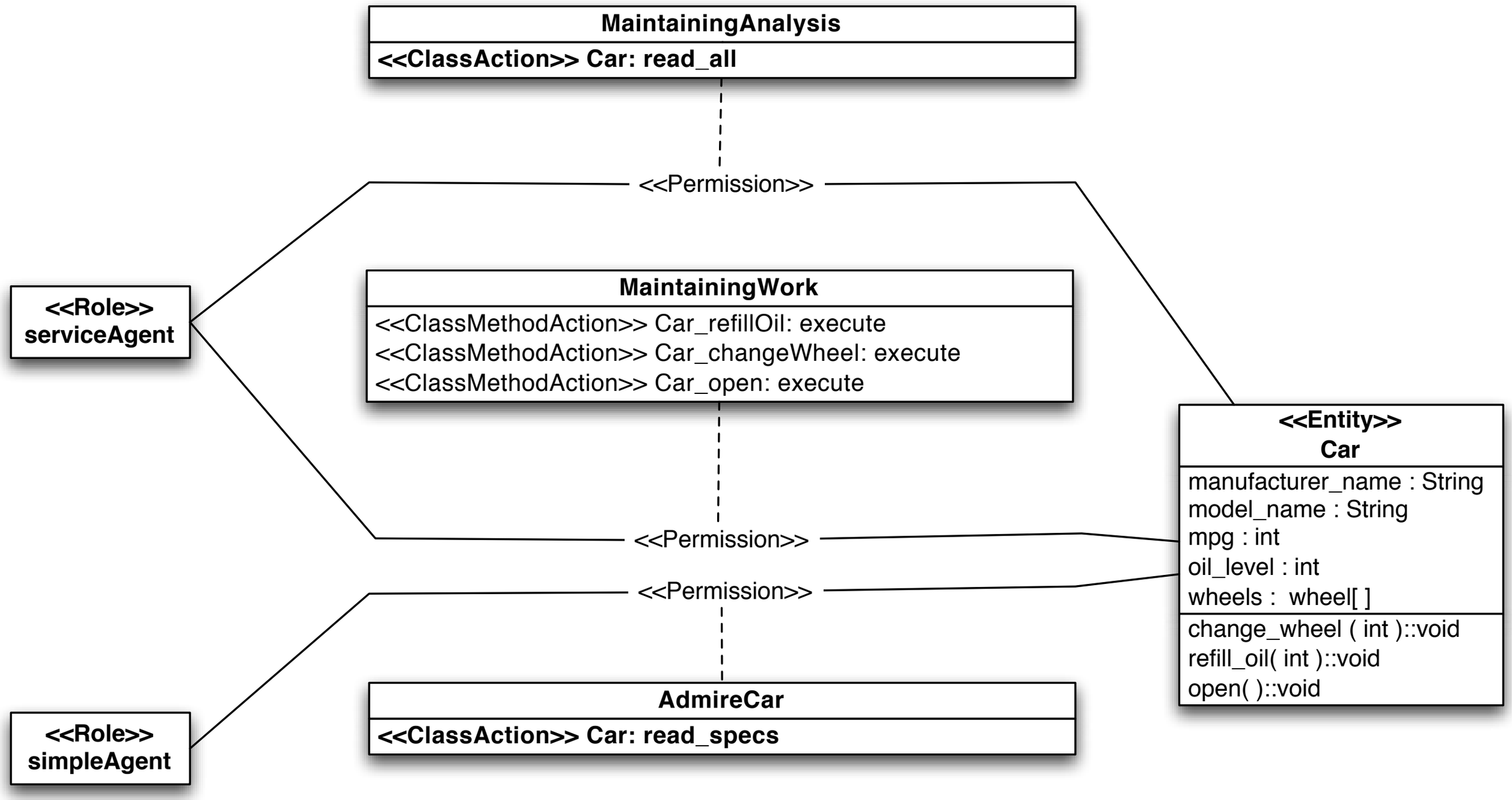
CompositeAction: read_all

<<ClassAction>> Car: read_specs
<<ClassMethodAction>> Car_getOilLevel: execute
<<ClassMethodAction>> Car_getWheels: execute

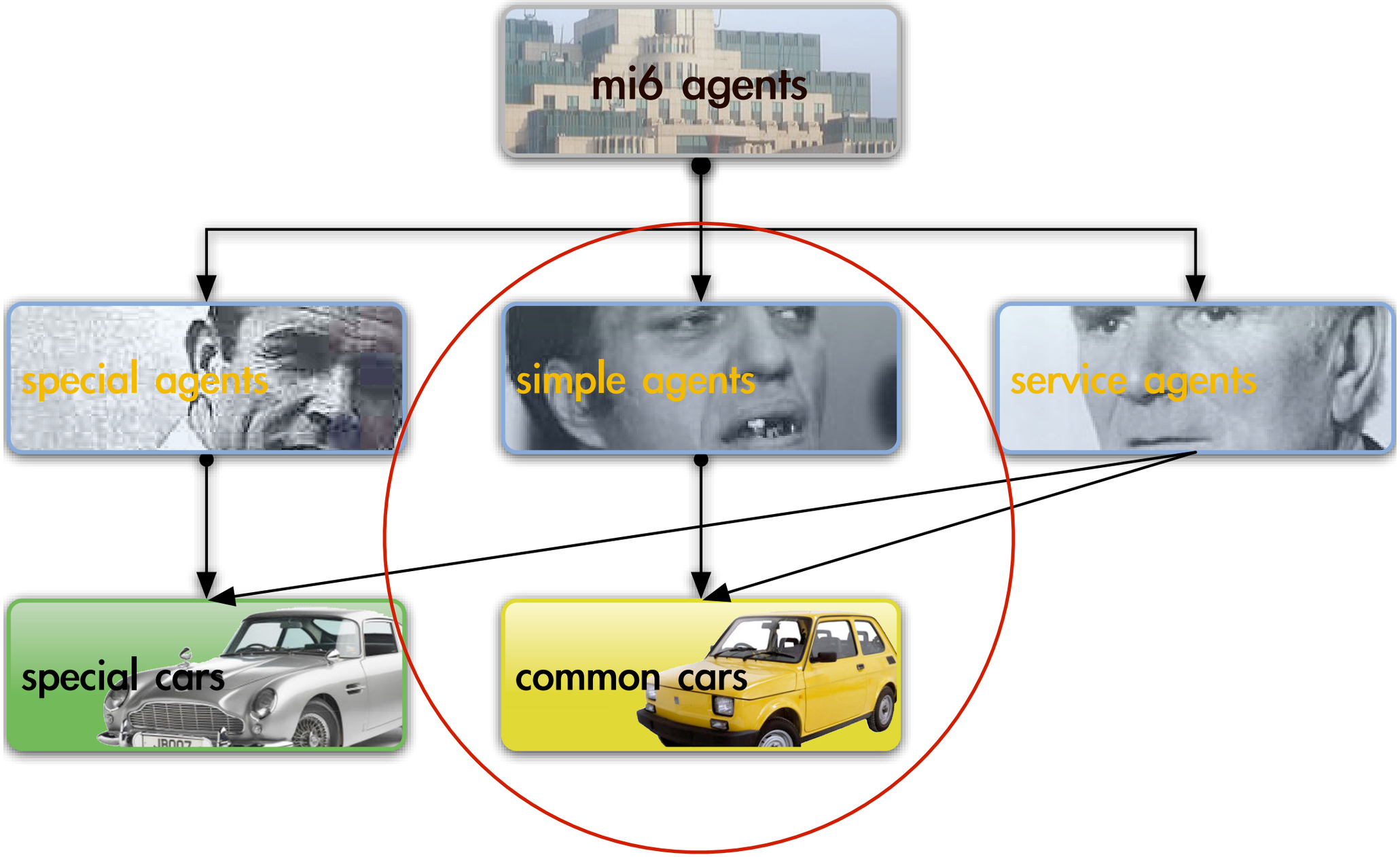
action hierarchy



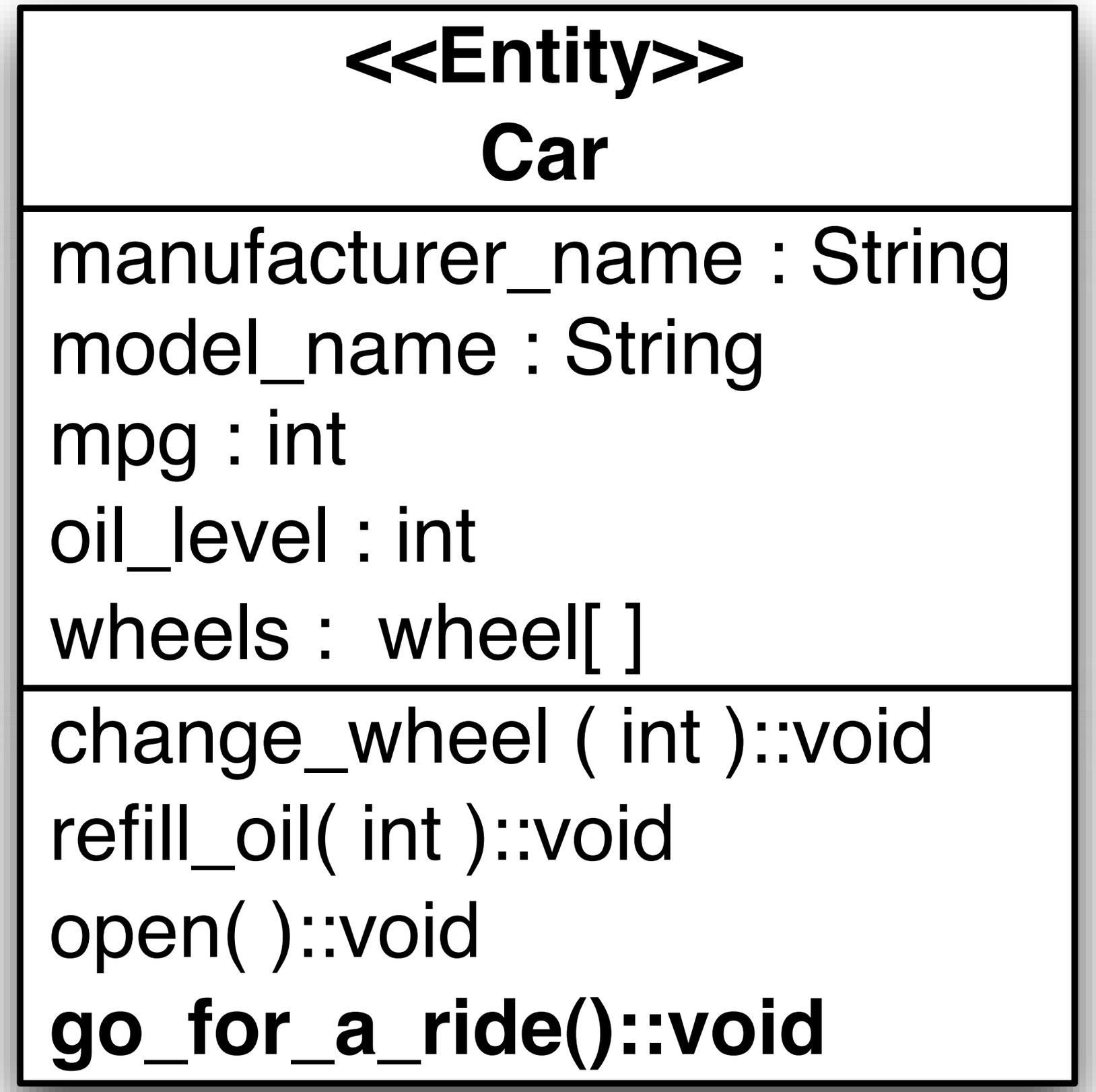
model


MaintainingAnalysis
<<ClassAction>> Car: read_all
<<Permission>>
<<Role>>
serviceAgent
MaintainingWork
<<ClassMethodAction>> Car_refillOil: execute
<<ClassMethodAction>> Car_changeWheel: execute
<<ClassMethodAction>> Car_open: execute
<<Permission>>
<<Permission>>
<<Entity>>
Car
manufacturer_name : String
model_name : String
mpg : int
oil_level : int
wheels : wheel[ ]
change_wheel (int)::void
refill_oil(int)::void
open( )::void
<<Role>>
simpleAgent
AdmireCar
<<ClassAction>> Car: read_specs

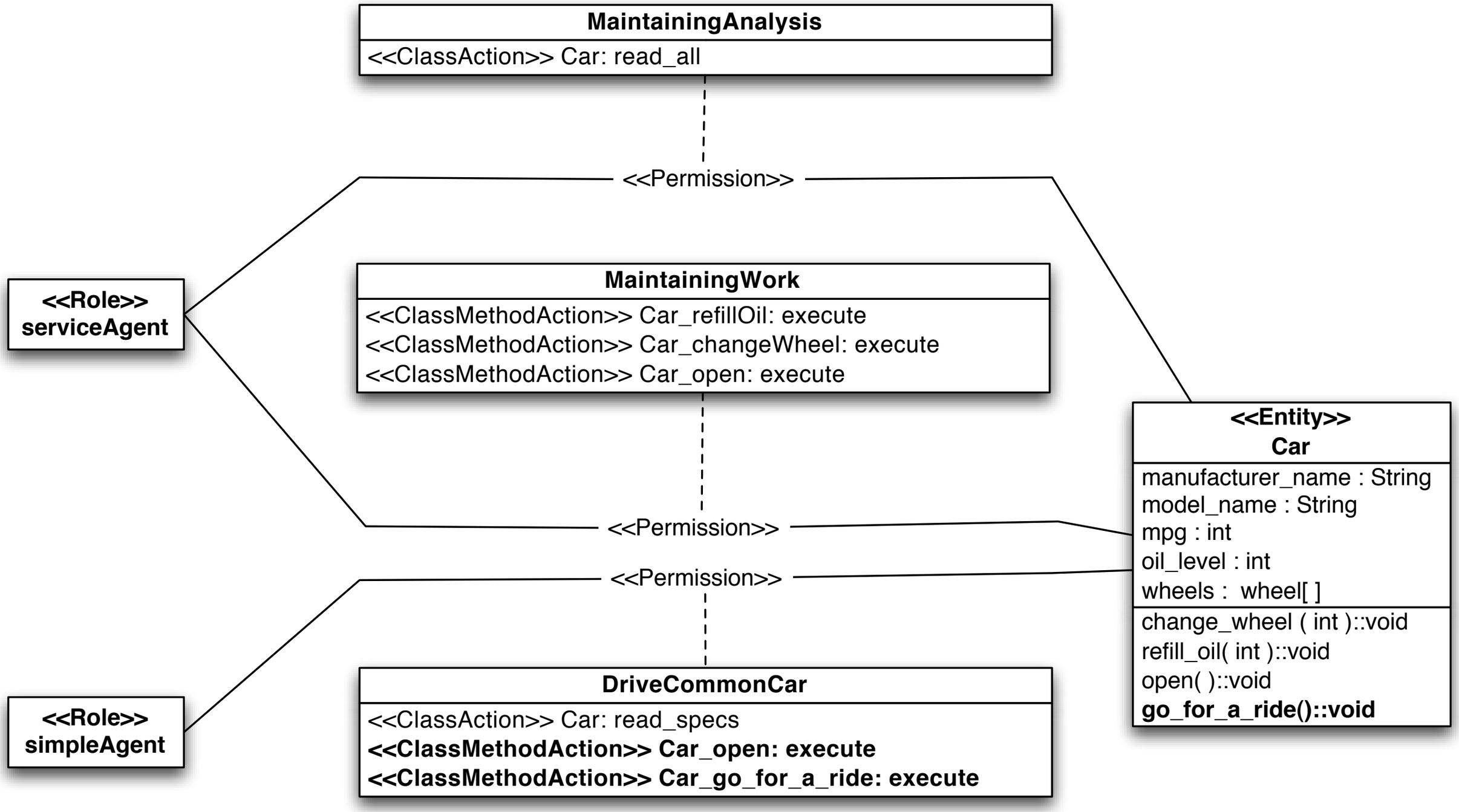
mi6 - car access policy



extend Car entity



model

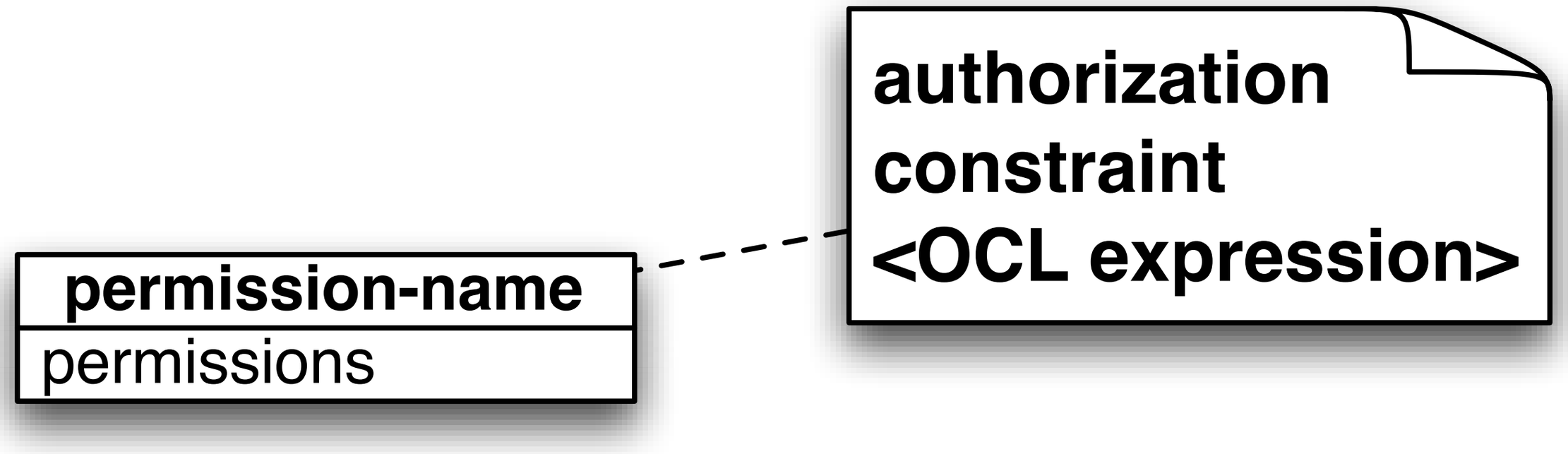

MaintainingAnalysis
<<ClassAction>> Car: read_all
<<Permission>>
<<Role>>
serviceAgent
MaintainingWork
<<ClassMethodAction>> Car_refillOil: execute
<<ClassMethodAction>> Car_changeWheel: execute
<<ClassMethodAction>> Car_open: execute
<<Entity>>
Car
manufacturer_name : String
model_name : String
mpg : int
oil_level : int
wheels : wheel[ ]
change_wheel (int)::void
refill_oil(int)::void
open( )::void
go_for_a_ride()::void
<<Permission>>
<<Permission>>
<<Role>>
simpleAgent
DriveCommonCar
<<ClassAction>> Car: read_specs
<<ClassMethodAction>> Car_open: execute
<<ClassMethodAction>> Car_go_for_a_ride: execute

MDS: access control decisions

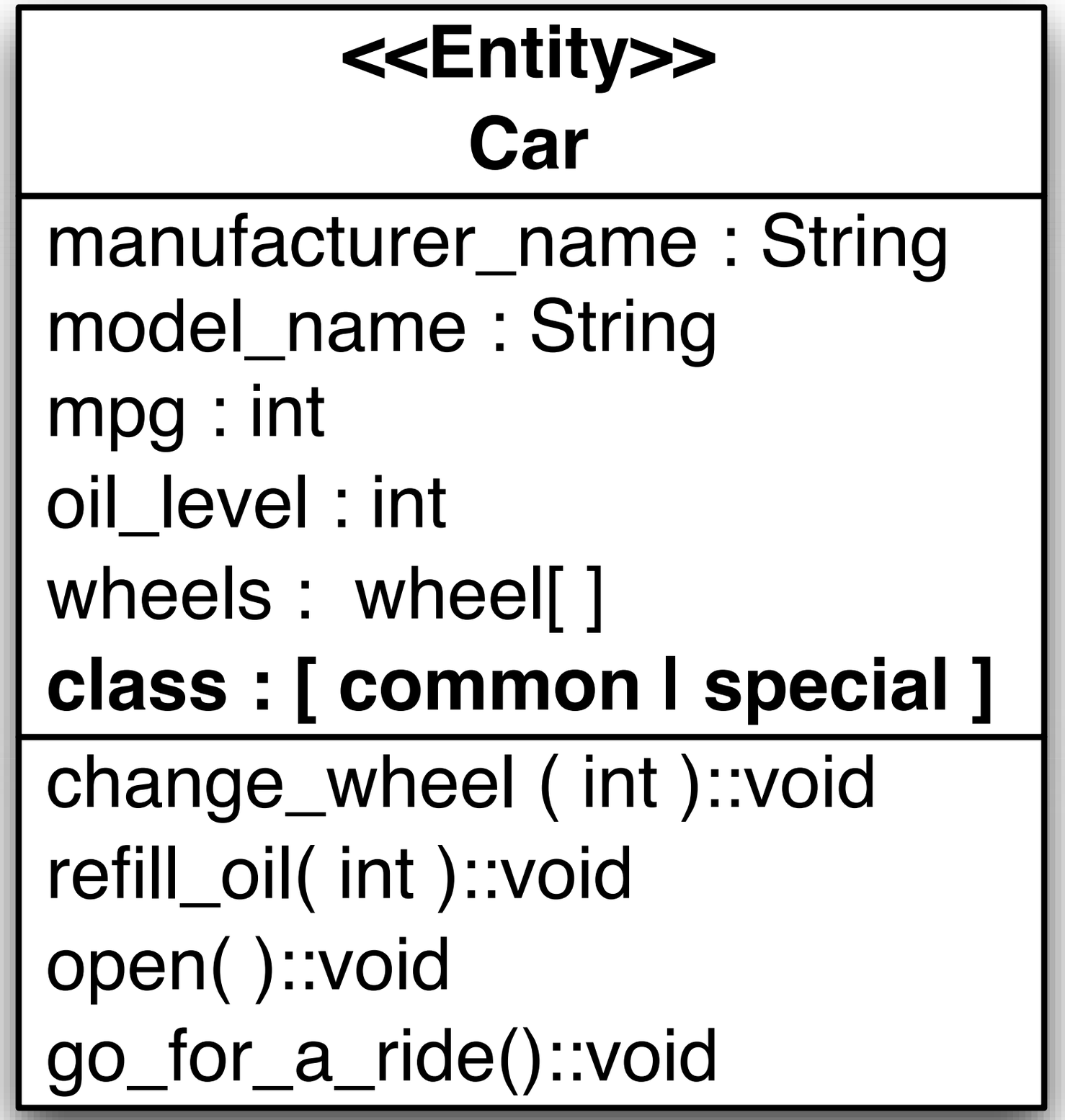
- declarative access control (static)
  ⇒ Permissions

- programmatic access control (dynamic)
  ⇒ AuthorizationConstraints

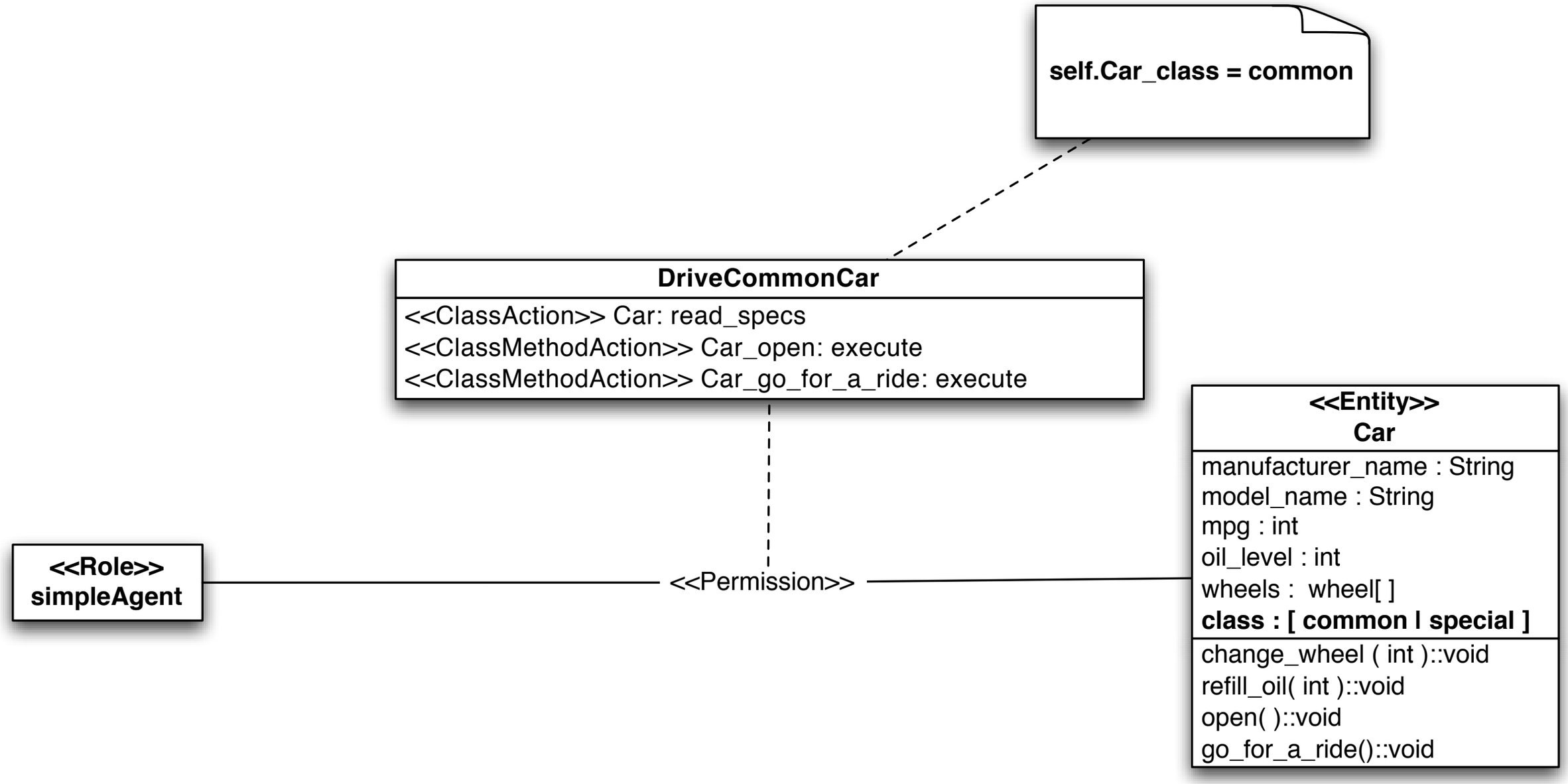
programmatic access control



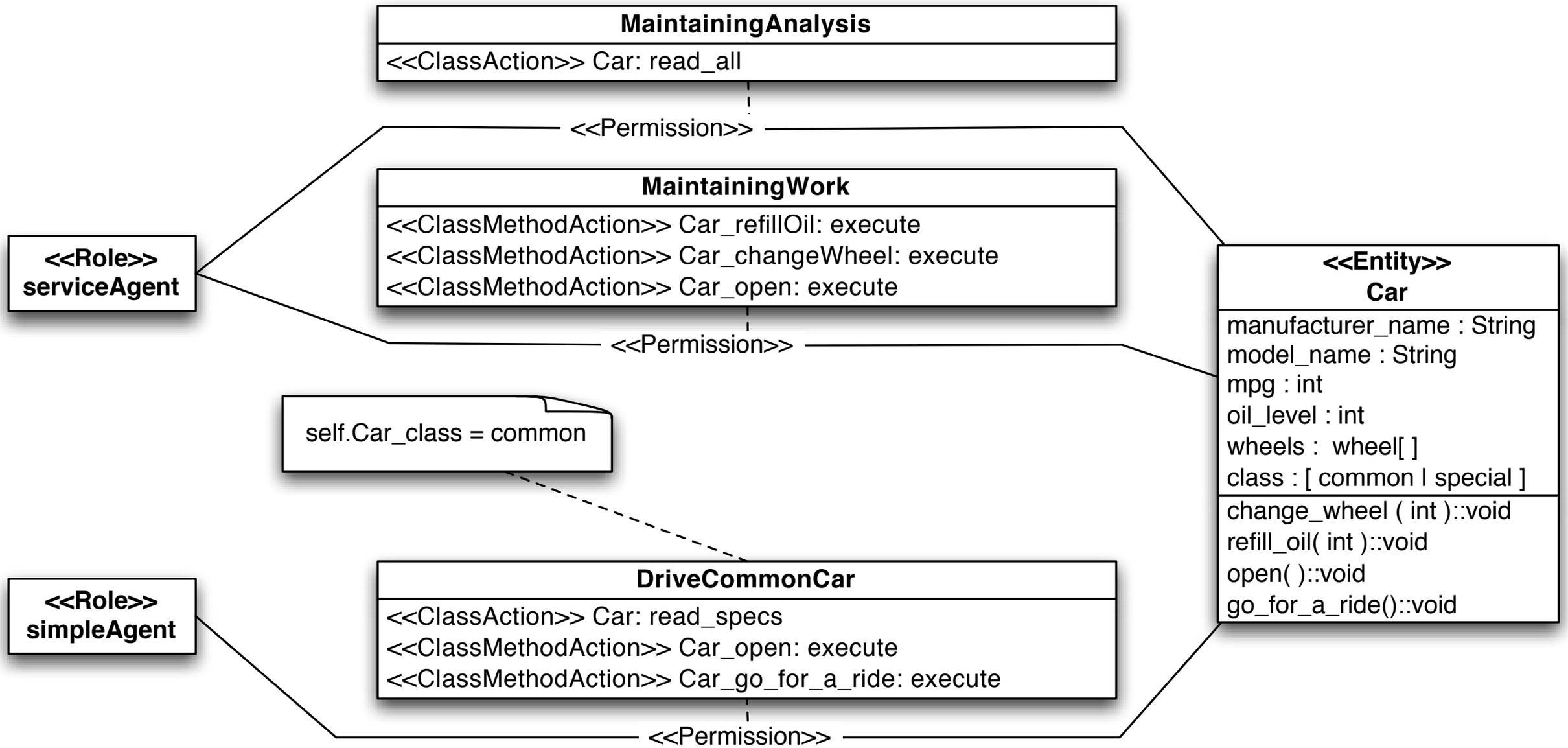
extend Car Entity



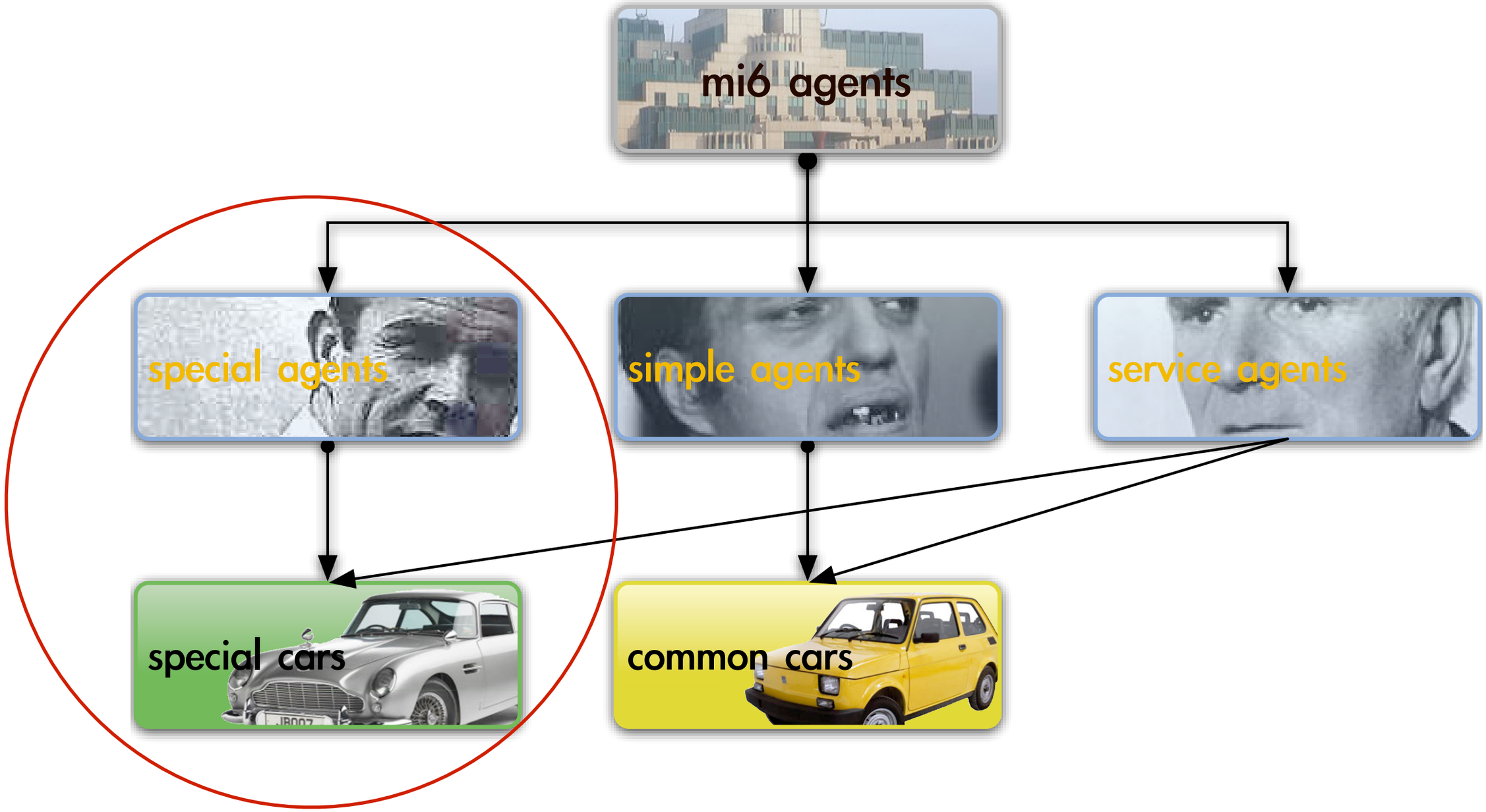
simpleAgent: may only drive common cars



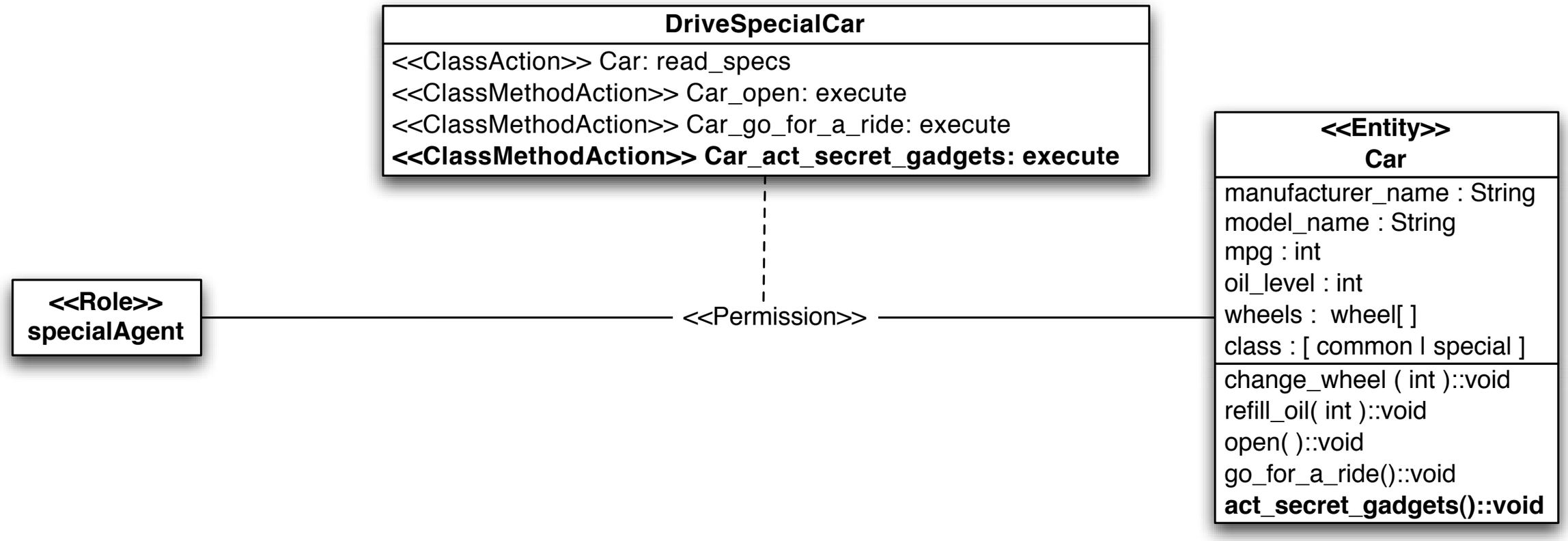
model


MaintainingAnalysis
<<ClassAction>> Car: read_all
<<Permission>>
MaintainingWork
<<ClassMethodAction>> Car_refillOil: execute
<<ClassMethodAction>> Car_changeWheel: execute
<<ClassMethodAction>> Car_open: execute
<<Permission>>
<<Role>>
serviceAgent
<<Entity>>
Car
manufacturer_name : String
model_name : String
mpg : int
oil_level : int
wheels : wheel[ ]
class : [common l special]
change_wheel (int)::void
refill_oil(int)::void
open( )::void
go_for_a_ride()::void
self.Car_class = common
DriveCommonCar
<<ClassAction>> Car: read_specs
<<ClassMethodAction>> Car_open: execute
<<ClassMethodAction>> Car_go_for_a_ride: execute
<<Role>>
simpleAgent
<<Permission>>

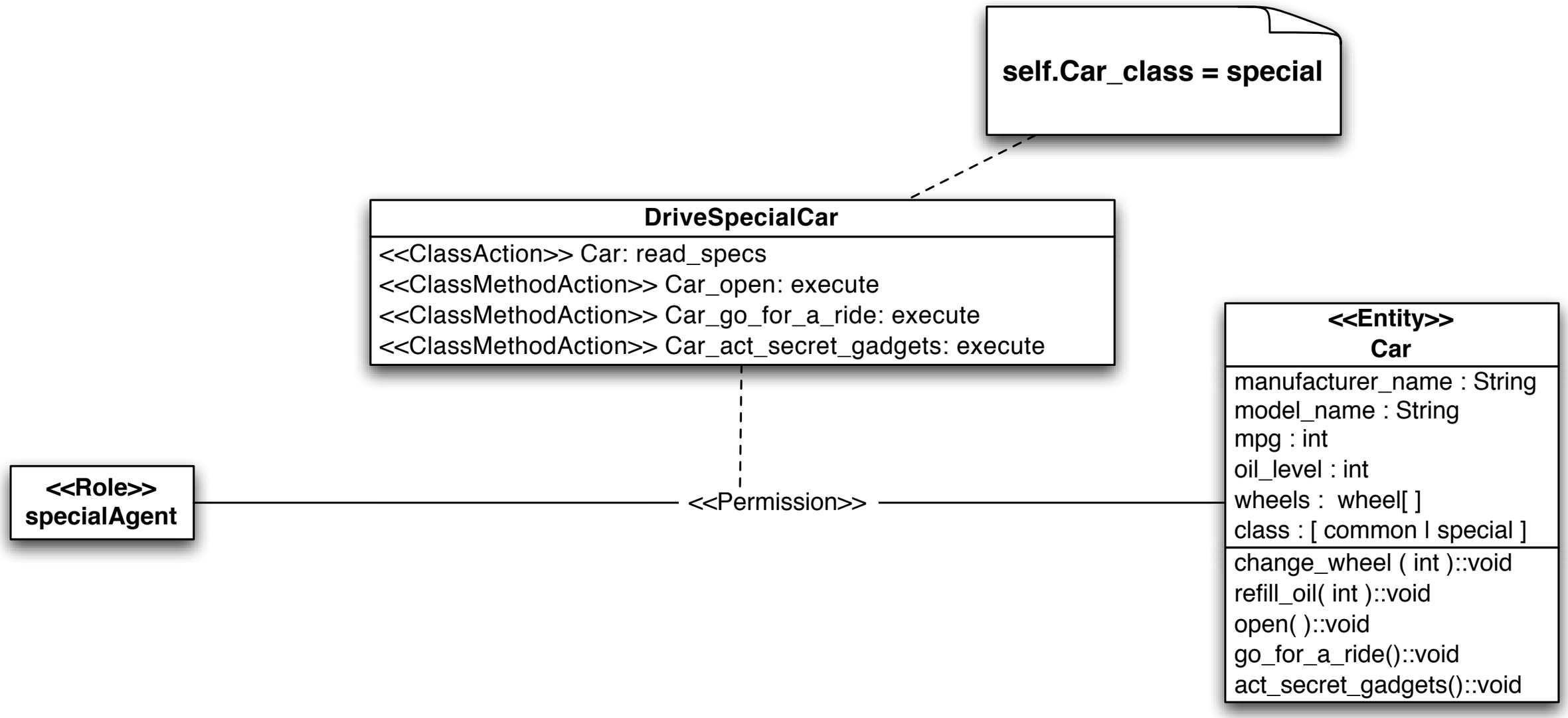
mi6 - car access policy



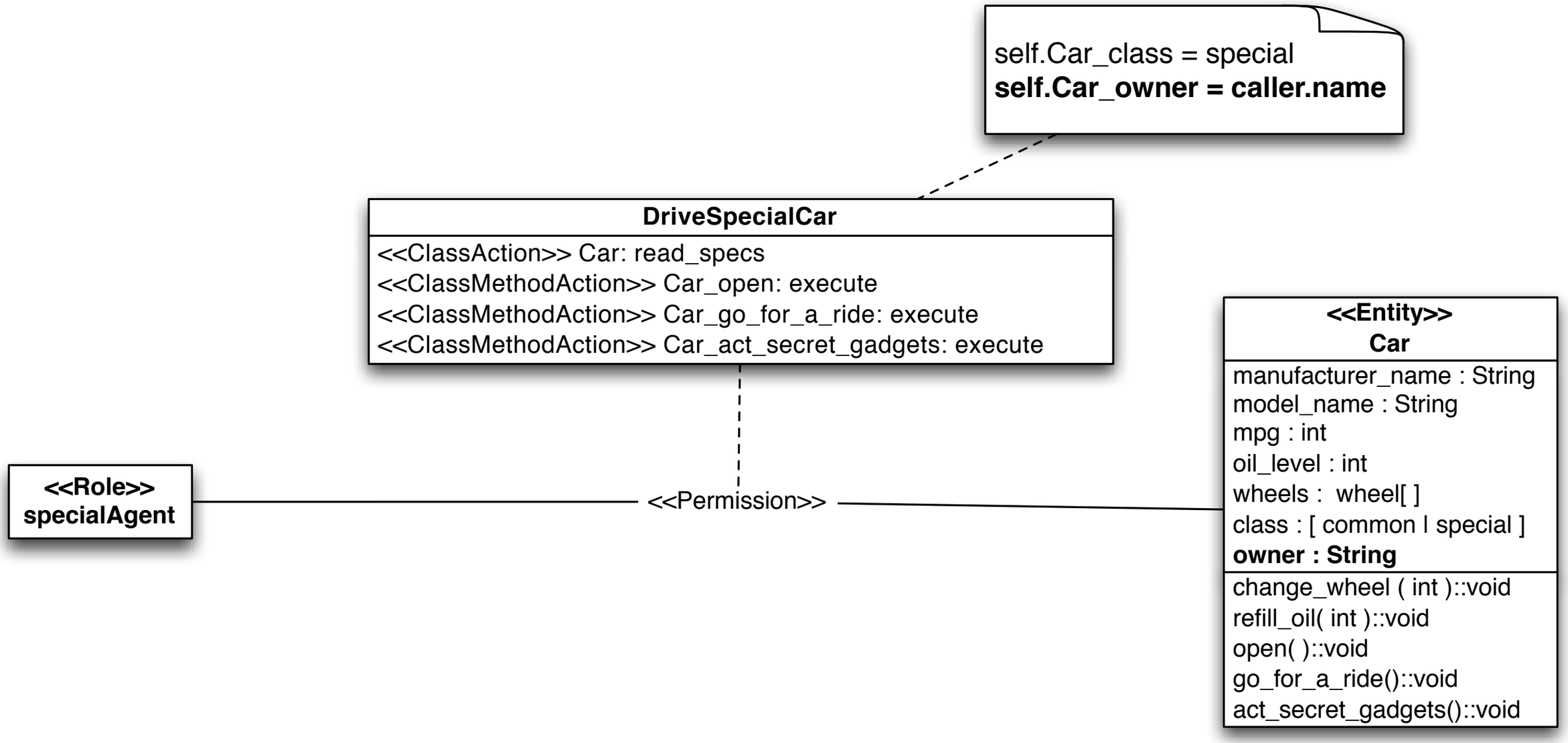
specialAgent:



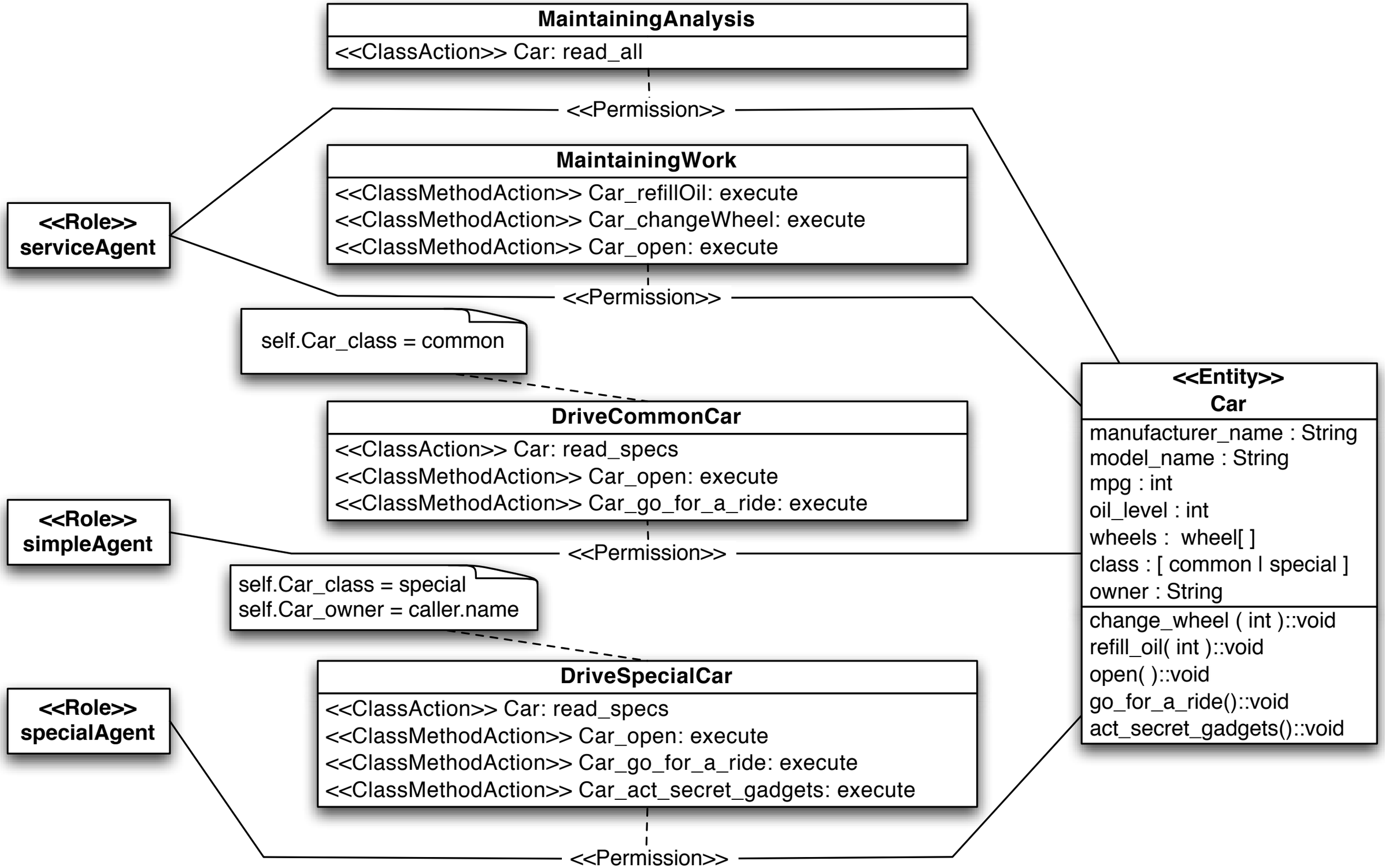
specialAgent: "may" only drive super cars



specialAgents: don't do carsharing



model

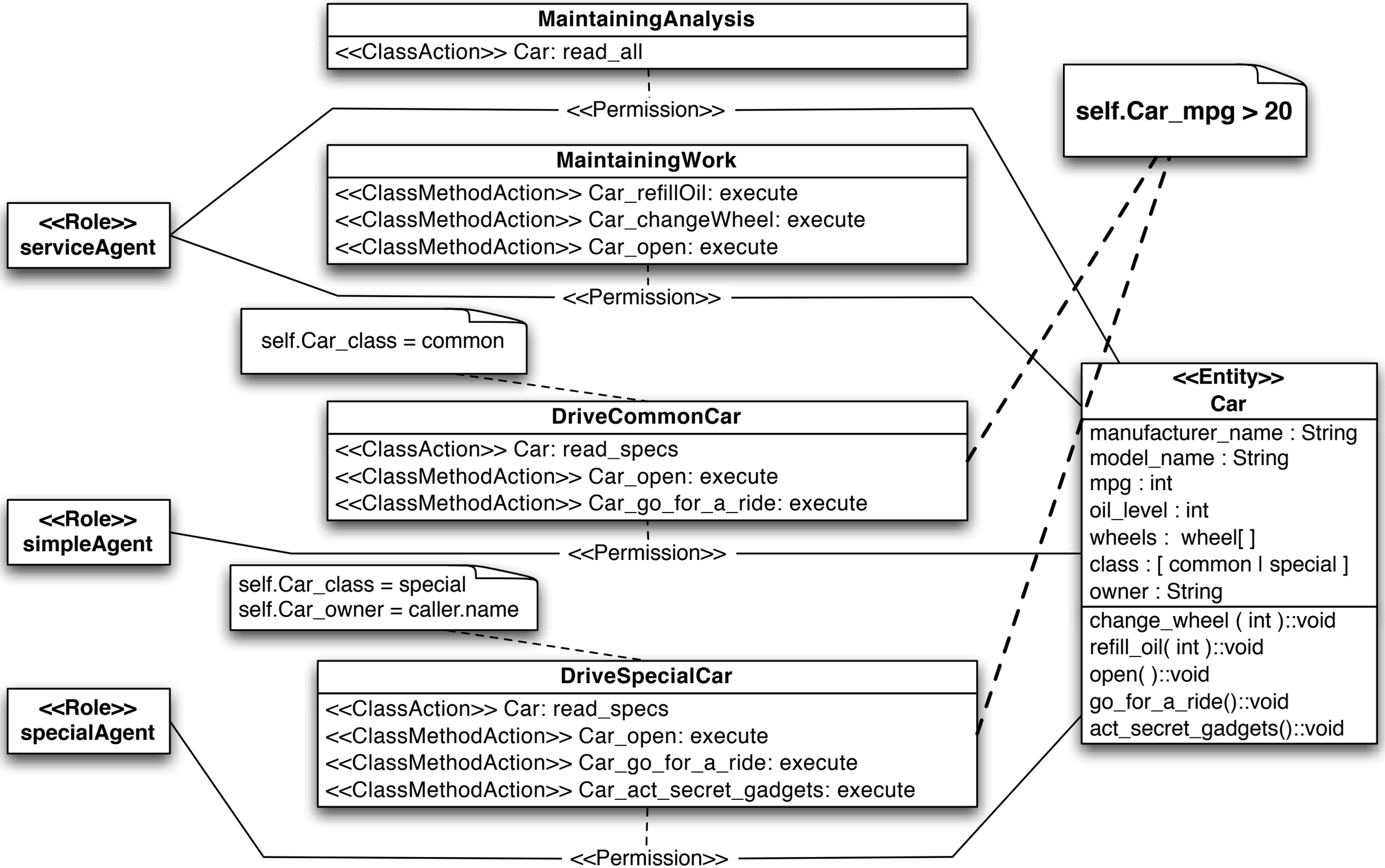

MaintainingAnalysis
<<ClassAction>> Car: read_all
<<Permission>>
MaintainingWork
<<ClassMethodAction>> Car_refillOil: execute
<<ClassMethodAction>> Car_changeWheel: execute
<<ClassMethodAction>> Car_open: execute
<<Role>>
serviceAgent
<<Permission>>
self.Car_class = common
<<Entity>>
Car
manufacturer_name : String
model_name : String
mpg : int
oil_level : int
wheels : wheel[ ]
class : [common I special]
owner : String
change_wheel (int)::void
refill_oil(int)::void
open( )::void
go_for_a_ride()::void
act_secret_gadgets()::void
DriveCommonCar
<<ClassAction>> Car: read_specs
<<ClassMethodAction>> Car_open: execute
<<ClassMethodAction>> Car_go_for_a_ride: execute
<<Role>>
simpleAgent
<<Permission>>
self.Car_class = special
self.Car_owner = caller.name
DriveSpecialCar
<<ClassAction>> Car: read_specs
<<ClassMethodAction>> Car_open: execute
<<ClassMethodAction>> Car_go_for_a_ride: execute
<<ClassMethodAction>> Car_act_secret_gadgets: execute
<<Role>>
specialAgent
<<Permission>>

change request

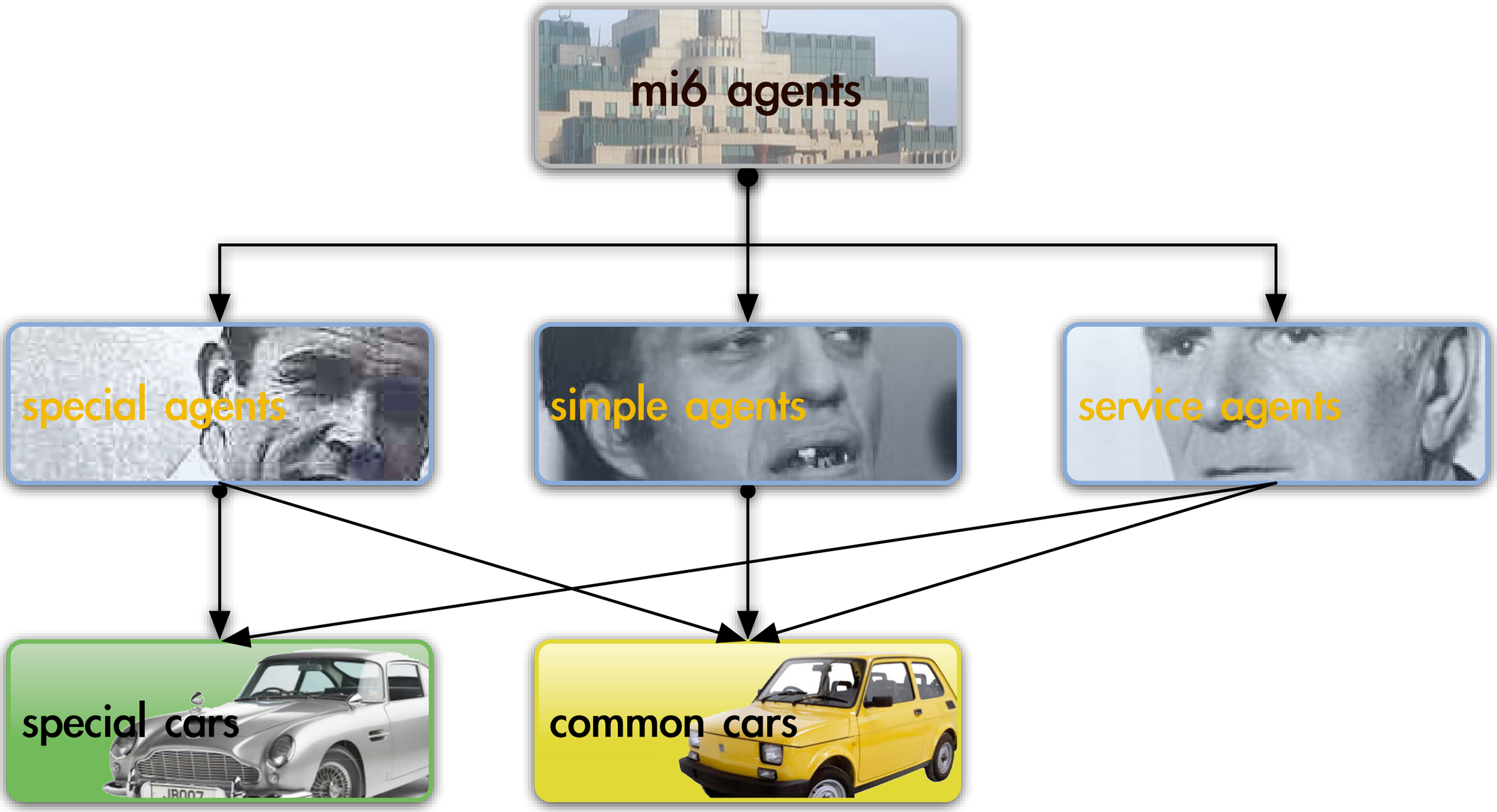
- I must reduce the CO2 emissions of our car fleet

- no car below 20 mpg may be used from now on

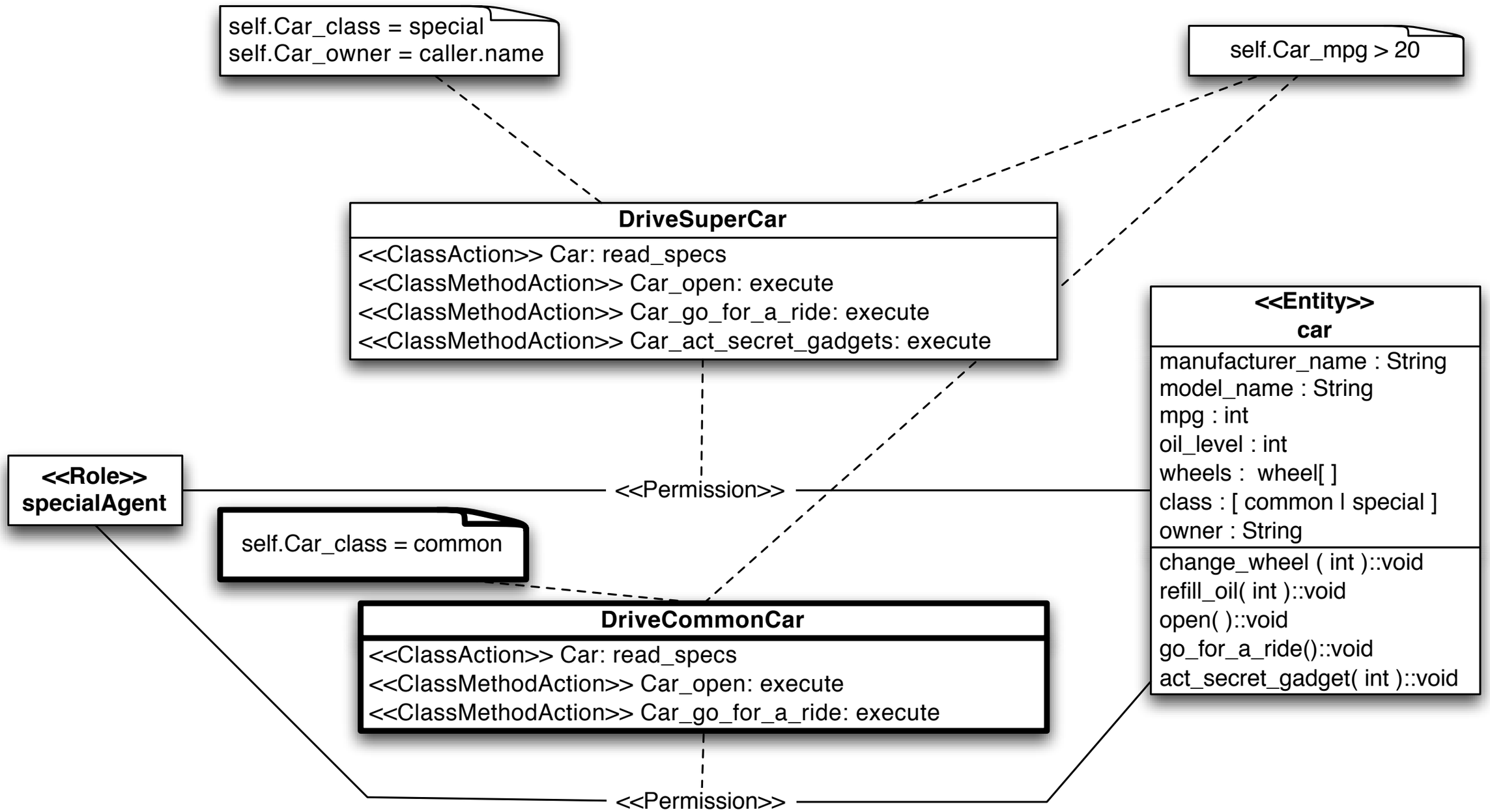
model


MaintainingAnalysis
<<ClassAction>> Car: read_all
<<Permission>>
self.Car_mpg > 20
MaintainingWork
<<ClassMethodAction>> Car_refillOil: execute
<<ClassMethodAction>> Car_changeWheel: execute
<<ClassMethodAction>> Car_open: execute
<<Role>>
serviceAgent
<<Permission>>
self.Car_class = common
<<Entity>>
Car
manufacturer_name : String
model_name : String
mpg : int
oil_level : int
wheels : wheel[ ]
class : [common | special]
owner : String
change_wheel (int)::void
refill_oil(int)::void
open( )::void
go_for_a_ride()::void
act_secret_gadgets()::void
DriveCommonCar
<<ClassAction>> Car: read_specs
<<ClassMethodAction>> Car_open: execute
<<ClassMethodAction>> Car_go_for_a_ride: execute
<<Role>>
simpleAgent
<<Permission>>
self.Car_class = special
self.Car_owner = caller.name
DriveSpecialCar
<<ClassAction>> Car: read_specs
<<ClassMethodAction>> Car_open: execute
<<ClassMethodAction>> Car_go_for_a_ride: execute
<<ClassMethodAction>> Car_act_secret_gadgets: execute
<<Role>>
specialAgent
<<Permission>>

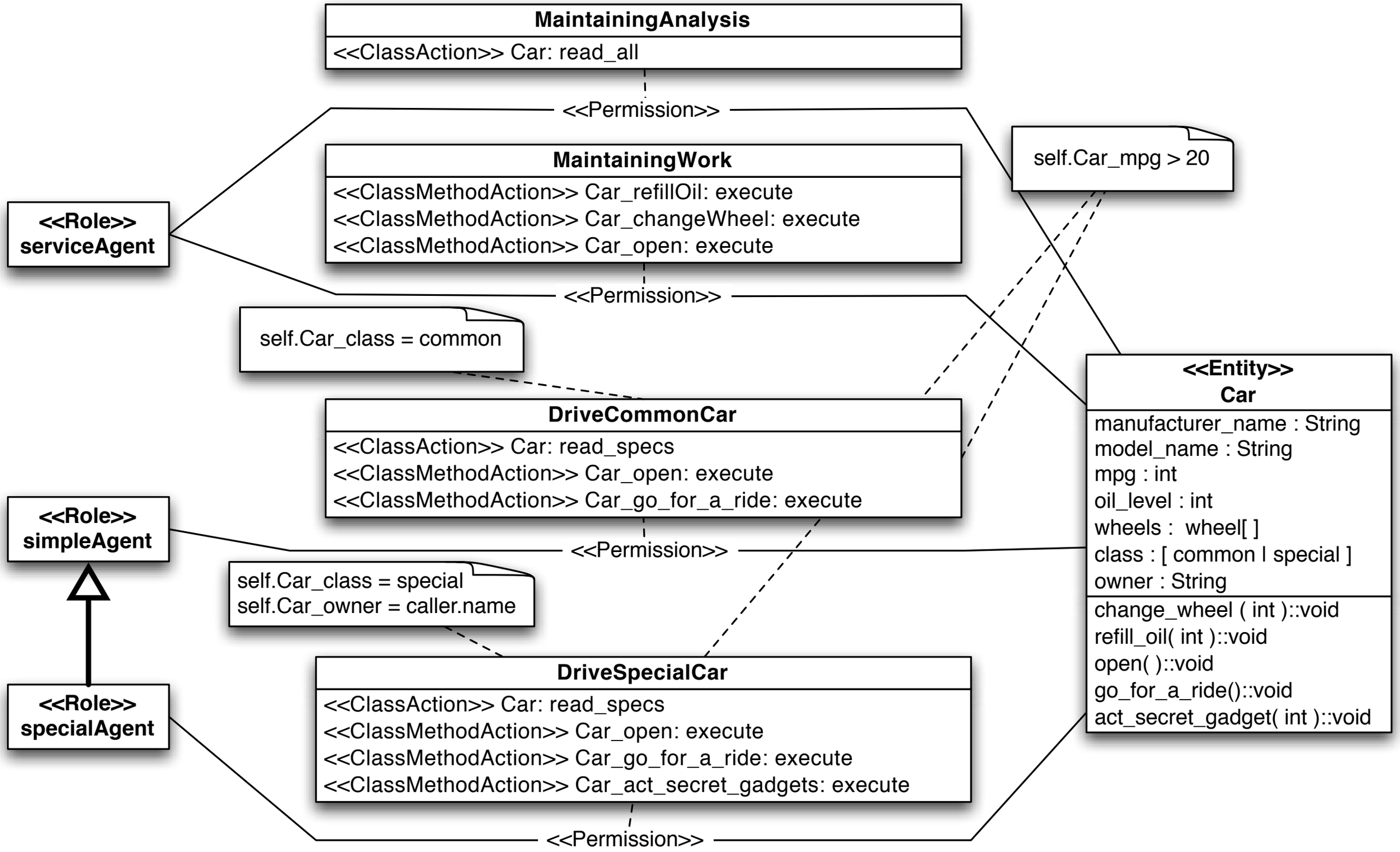
specialAgents may drive any car



copy / paste simpleAgent permissions

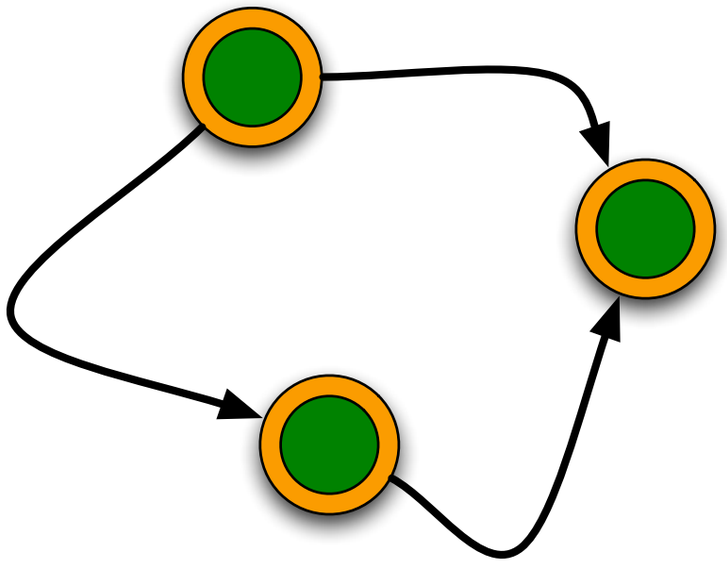


use role hierarchy

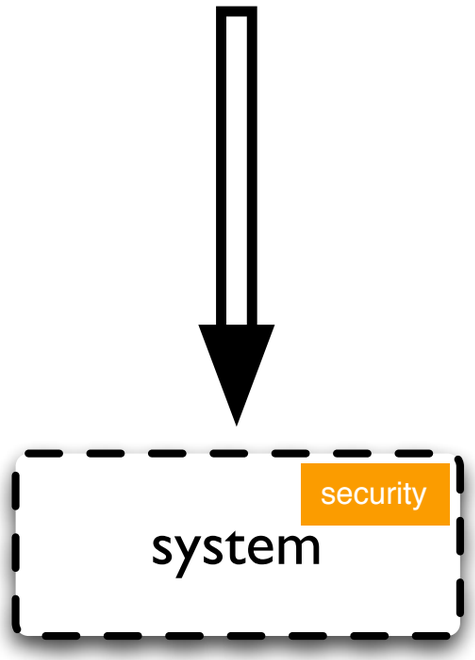


MDS: Model Driven Security

- mi6 as model
- cars as protected resources



- RBAC based security policies
- empty EJB stubs + code implementing security mechansims



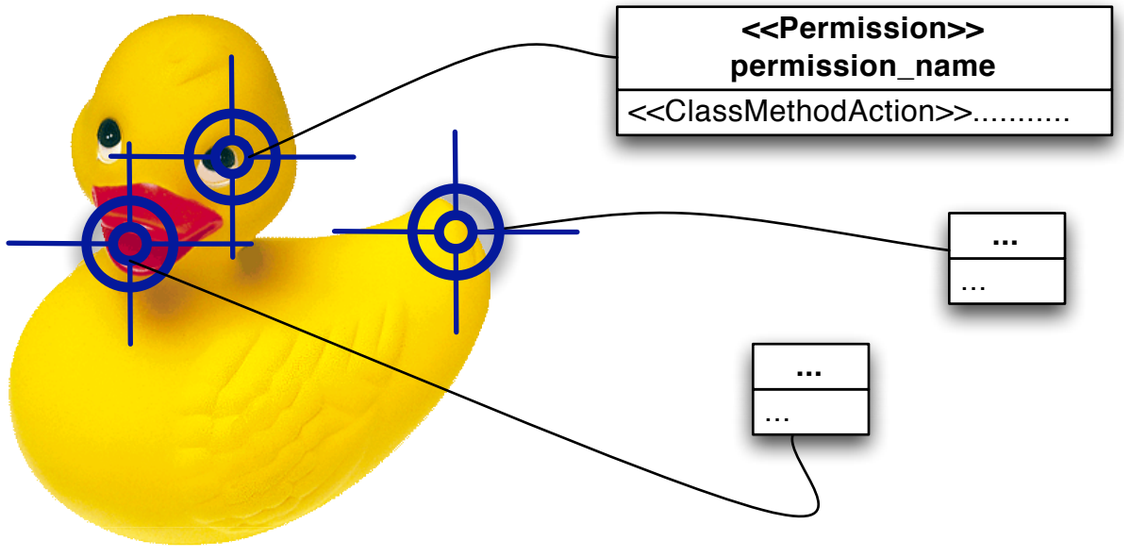
summary:

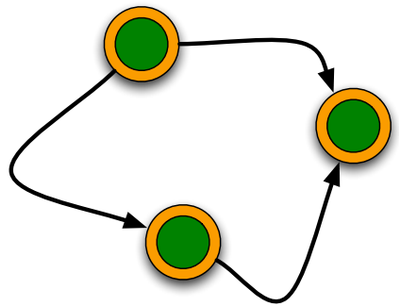
- roles | permissions | entities
- composite actions
- action hierarchy
- authorization constraints
- role hierarchy

conventional approach vs. MDS

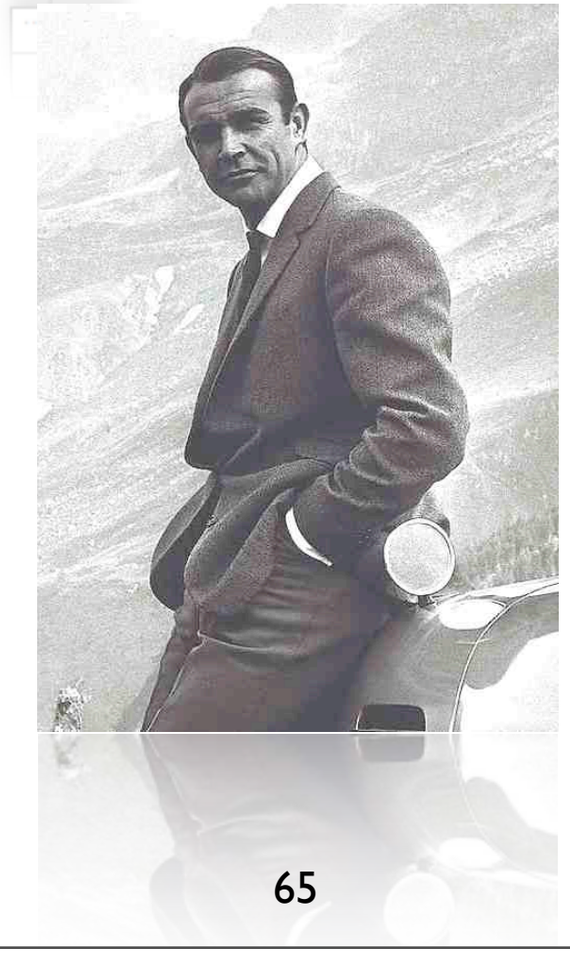
conventional approach	MDS
low level	arbitrary level of abstraction
policy format: XML	model elements (UML)
copy - paste / wildcards	hierarchy / composite container
running code from day 1	time intensive modeling, business logic comes later

remember:


<<Permission>>
permission_name
<<ClassMethodAction>>.....
...
...
...




security
system



bottom line:

- model driven security offers:
  - common representation for system and security
  - general language composition schema
  - arbitrary levels of abstraction
  - unambiguous target code generation
  - semantics as basis for model checking

bottom line:

- model driven security drawbacks:
  - modeling needs time and skills

    (reduce needed skills: tool development process, system development process)
  - new composite actions / action hierarchies
    ⇒ change the dialect

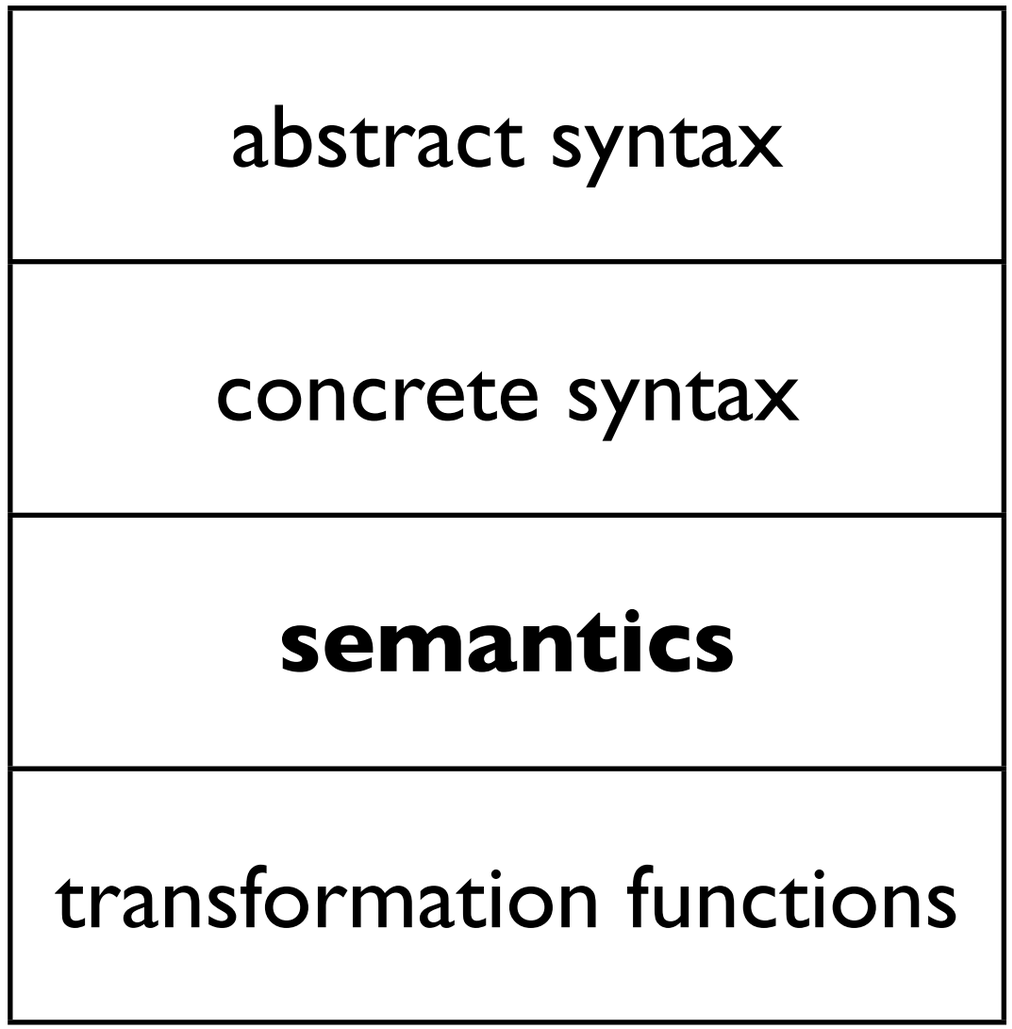
    ⇒ recomposition of language

    (can be solved with macros)
  - modifying the model ⇒ apply transformation functions again

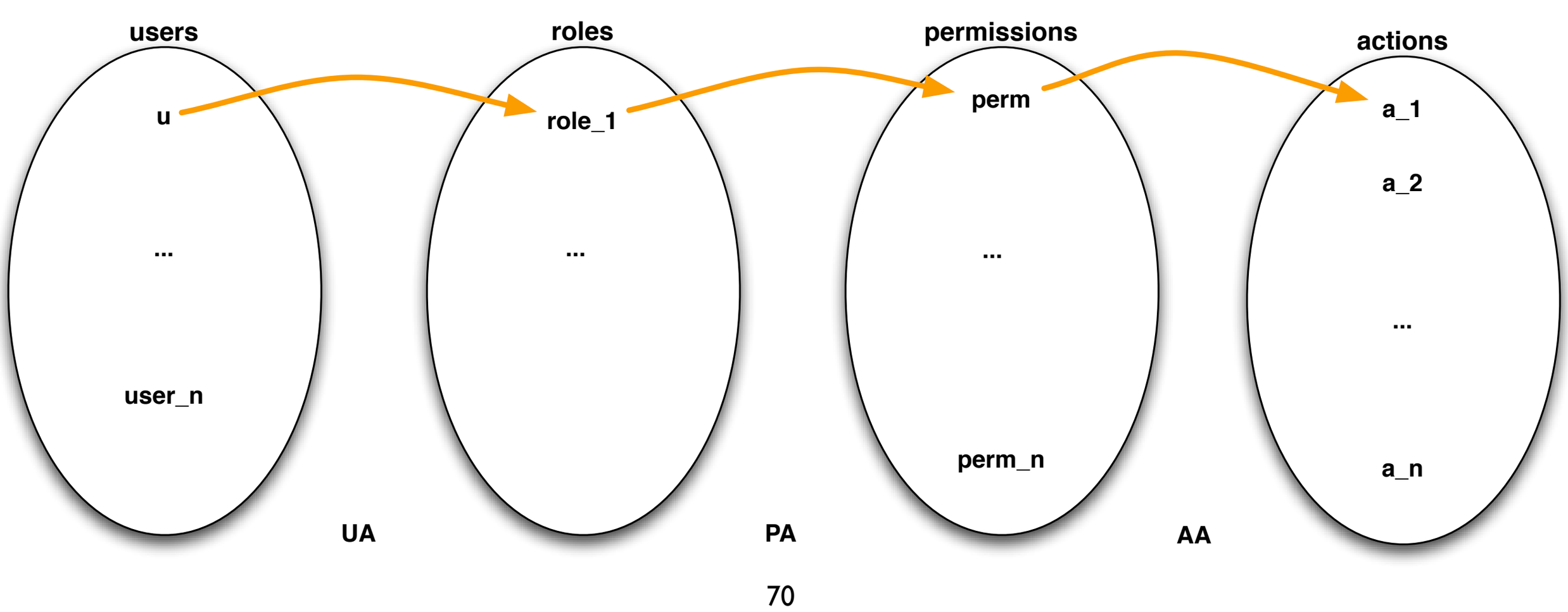
    (can be solved with dedicated IDE or business logic stored outside of bean)
  - (“code generator” needed)

SecureUML

- modeling language

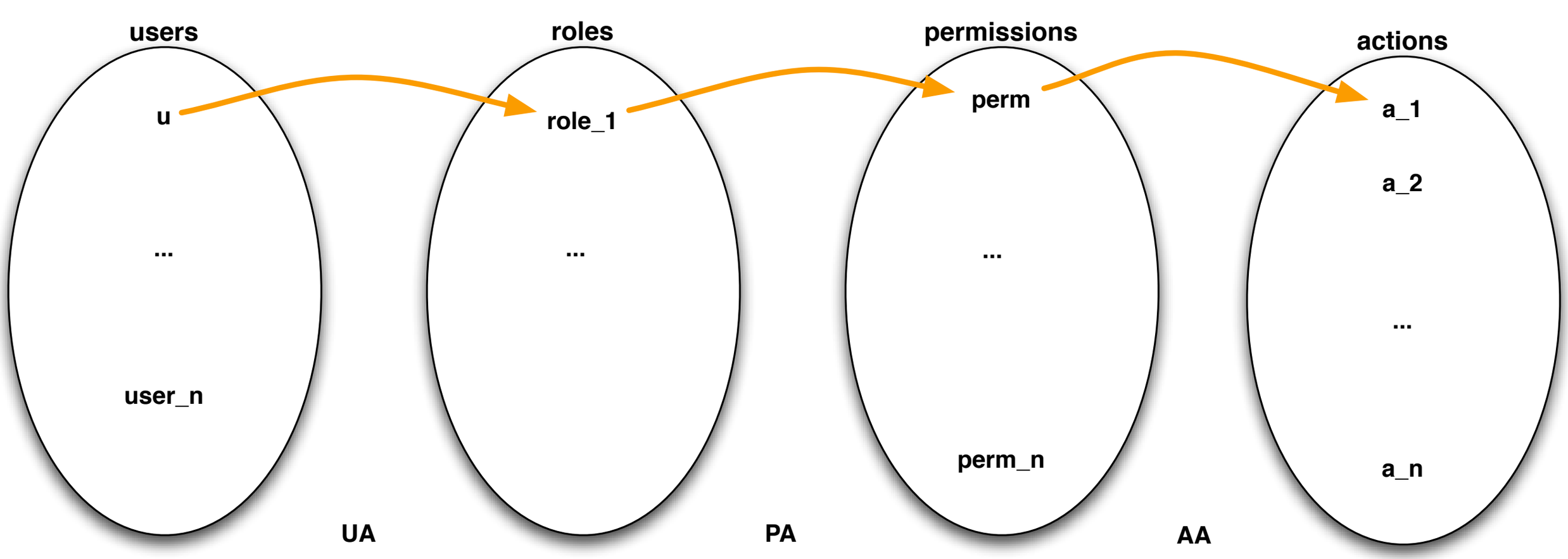


role based access control

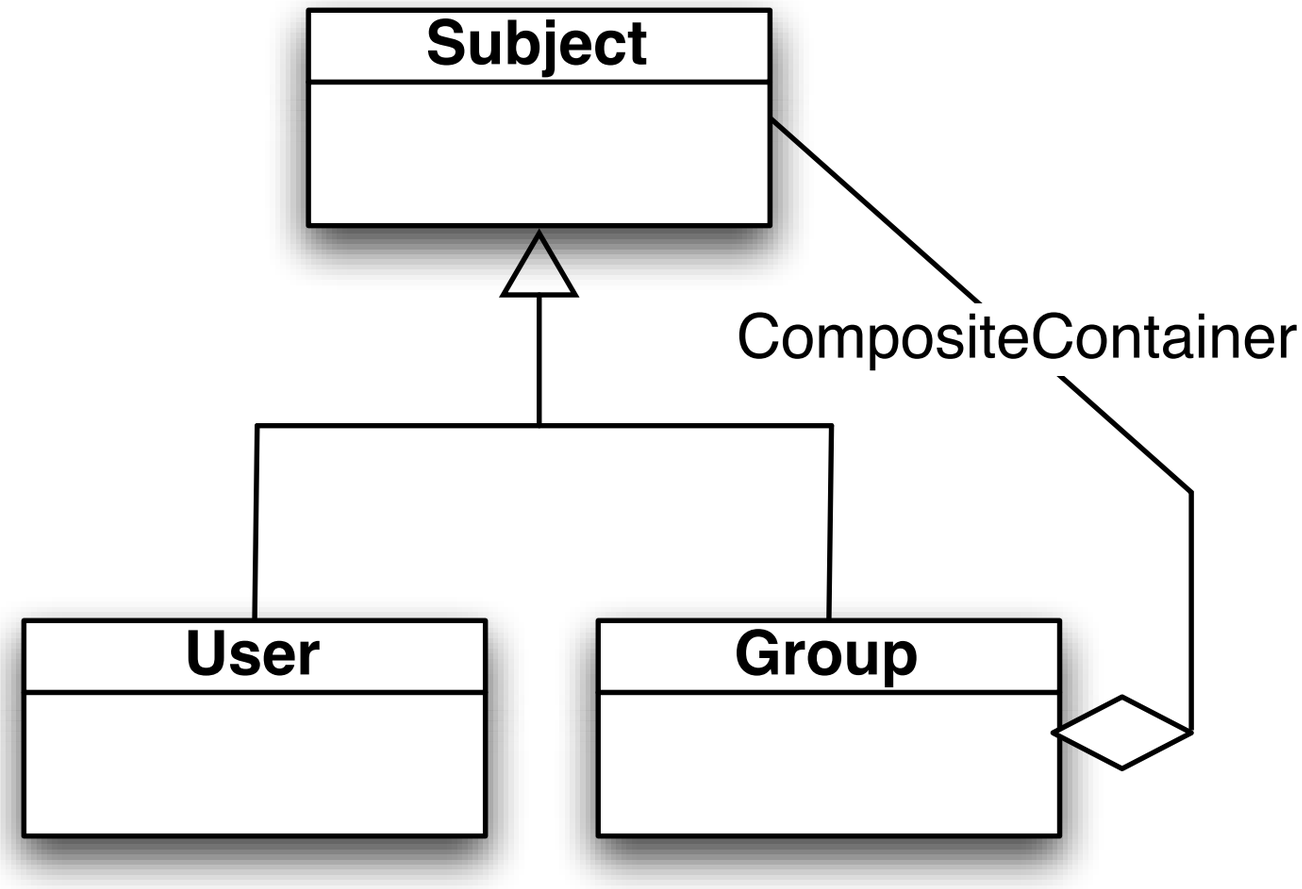


role based access control

$$\text{RBAC}_{simple} = \{$$
$$(\text{u}, \text{a}_1) \in Users \times Actions |$$
$$\exists role_1 \in Roles,\ perm \in Permissions\ .$$
$$(\text{u}, \text{role}_1) \in UA\ \wedge$$
$$(\text{role}_1,\ perm) \in PA\ \wedge$$
$$(\text{perm}, \text{a}_1) \in AA$$
$$\}$$

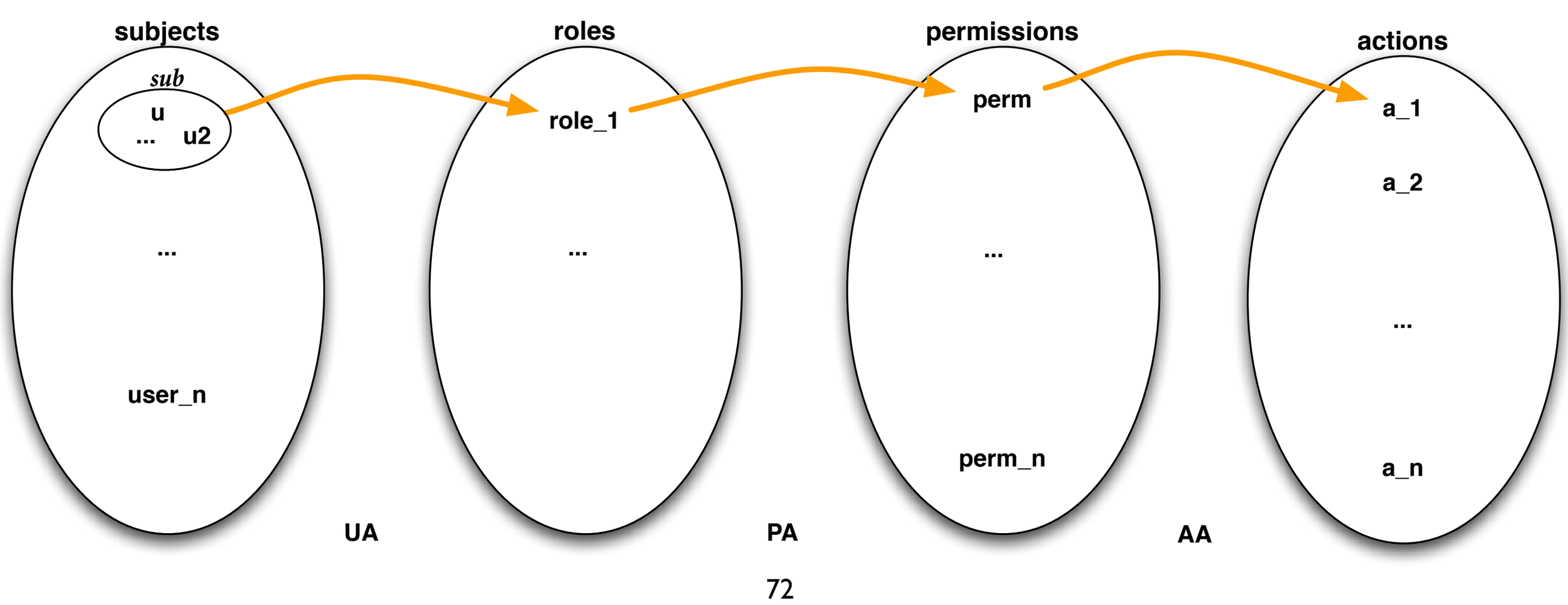


adding subjects



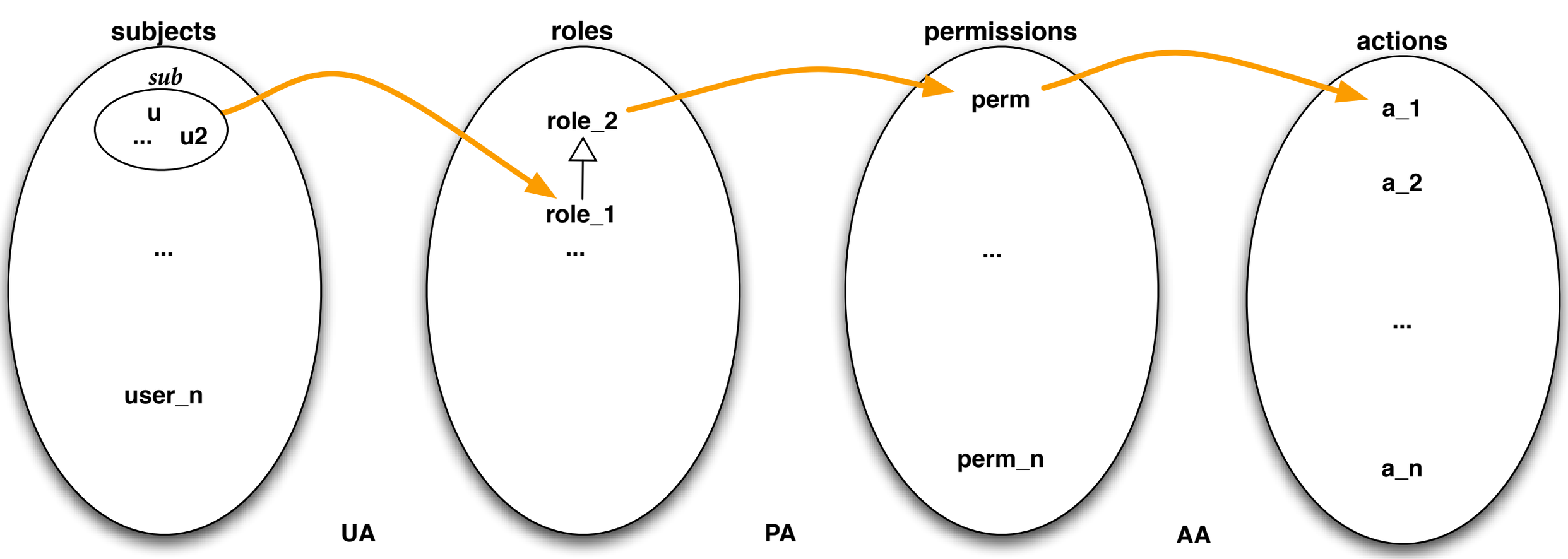
adding subjects

$$
\begin{aligned}
&\mathrm{RBAC}_{w.subjects} = \{ \\
&(\mathrm{u}, \mathrm{a}_1) \in Users \times Actions | \\
&\exists sub \in Subjects,\ role_1 \in Roles,\ perm \in Permissions, a_1 \in Actions\ . \\
&(\mathrm{sub}, \mathrm{role}_1) \in UA\ \wedge \\
&\mathrm{sub} \geq_{Subjects} u \wedge \\
&(\mathrm{role}_1,\ perm) \in PA\ \wedge \\
&(\mathrm{perm},\ \ \mathrm{a}_1) \in AA \\
&\}
\end{aligned}
$$



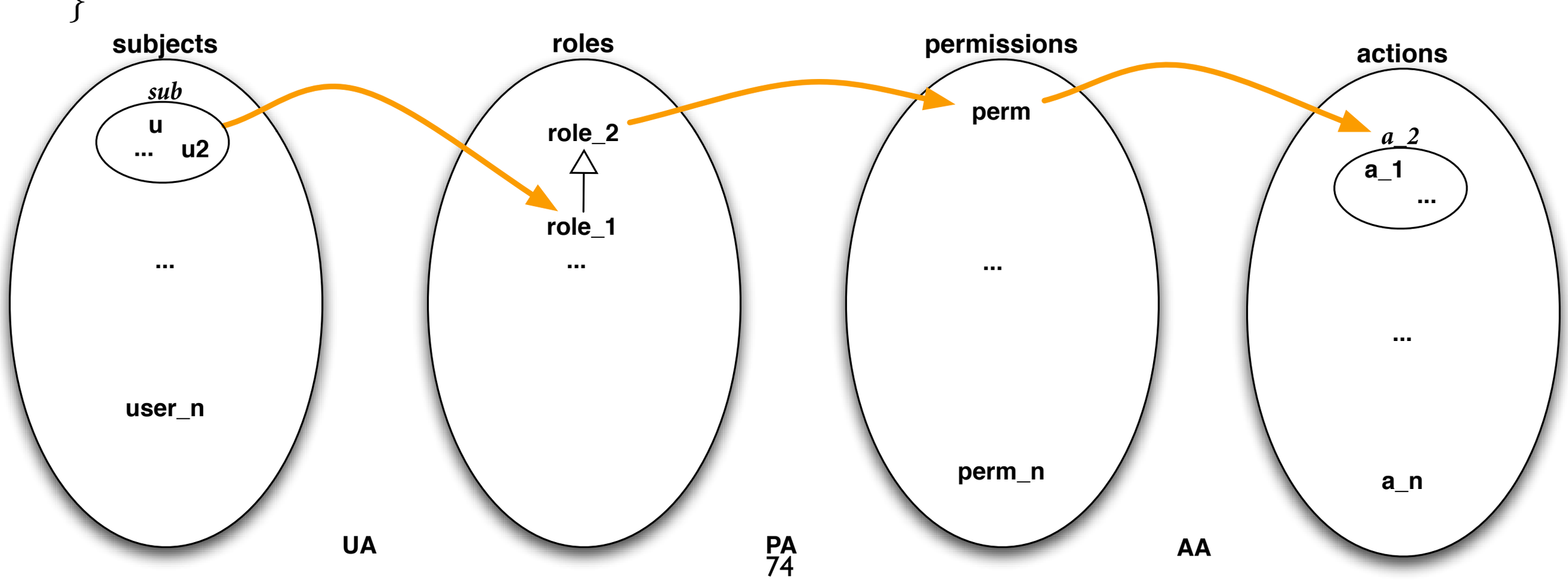
adding role hierarchy

$$
\begin{aligned}
&\mathrm{RBAC}_{w.roleH.=}\{\\
&(\mathrm{u}, \mathrm{a}_1) \in Users \times Actions |\\
&\exists sub \in Subjects,\ role_1,\ role_2 \in Roles,\ perm \in Permissions,\ a_1 \in Actions\ .\\
&(\mathrm{sub},\ \ \mathrm{role}_1) \in UA\ \wedge\\
&\mathrm{sub} \geq_{Subjects} u\ \wedge\\
&\mathrm{role}_1 \geq_{Roles} role_2\ \wedge\\
&(\mathrm{role}_2,\ perm) \in PA\ \wedge\\
&(\mathrm{perm},\ \ \mathrm{a}_1) \in AA\\
&\}
\end{aligned}
$$



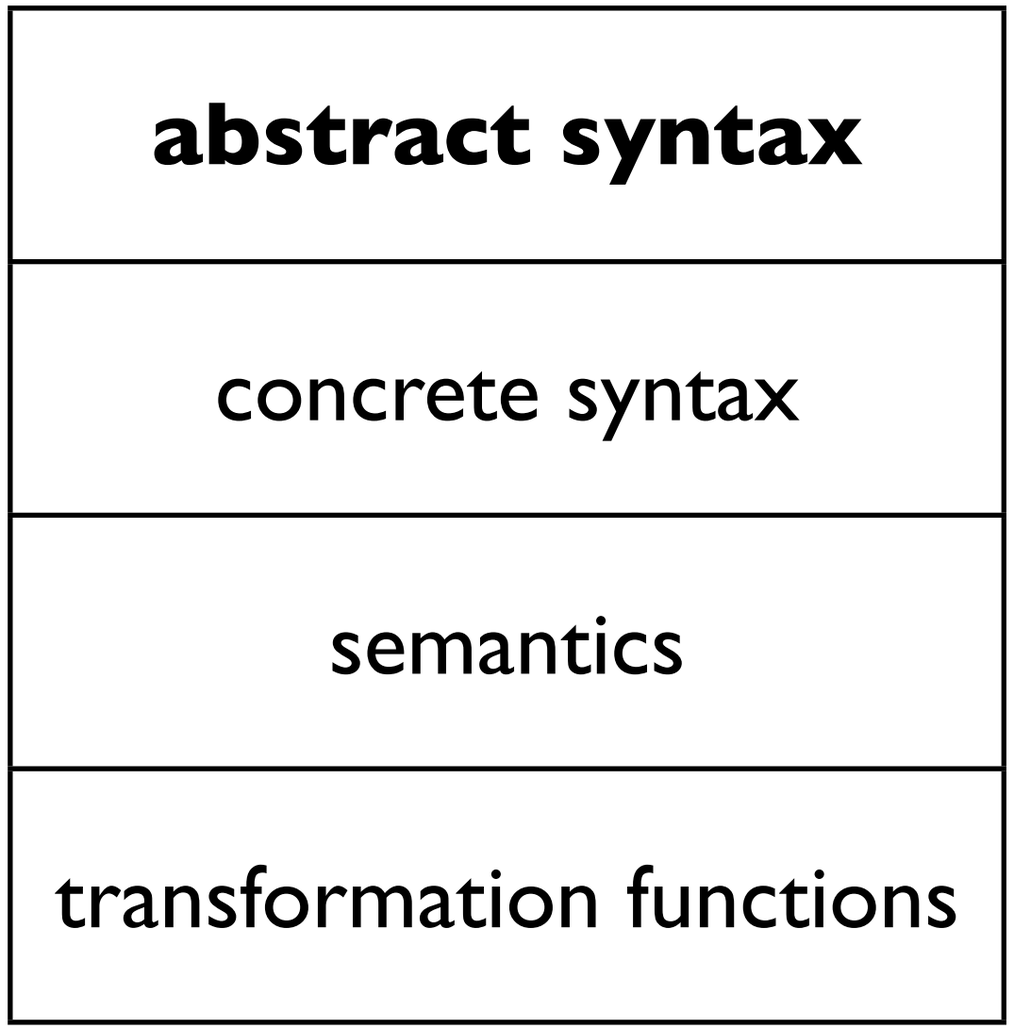
adding composite actions

$$
\begin{aligned}
&\mathrm{RBAC}_{w.compA.} = \{ \\
&(\mathrm{u},\ \mathrm{a}_1) \in Users \times Actions | \\
&\exists sub \in Subjects,\ role_1, role_2 \in Roles,\ perm \in Permissions,\ a_2 \in Actions\ . \\
&(\mathrm{sub},\ \mathrm{role}_1) \in UA\ \wedge \\
&\mathrm{sub} \geq_{Subjects} u \wedge \\
&\mathrm{role}_1 \geq_{Roles} role_2 \wedge \\
&\mathrm{a}_2 \geq_{Actions} a_1 \wedge \\
&(\mathrm{role}_2,\ perm) \in PA\ \wedge \\
&(\mathrm{perm},\ \mathrm{a}_2) \in AA \\
&\}
\end{aligned}
$$

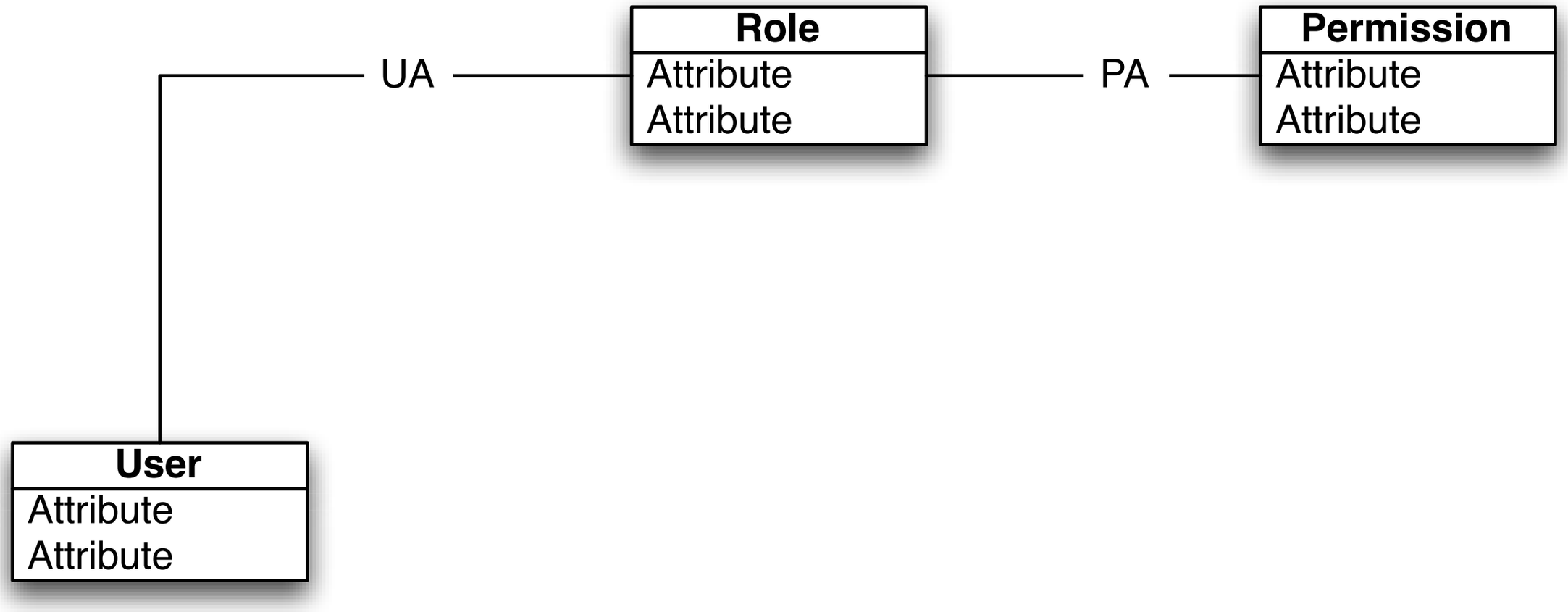


SecureUML

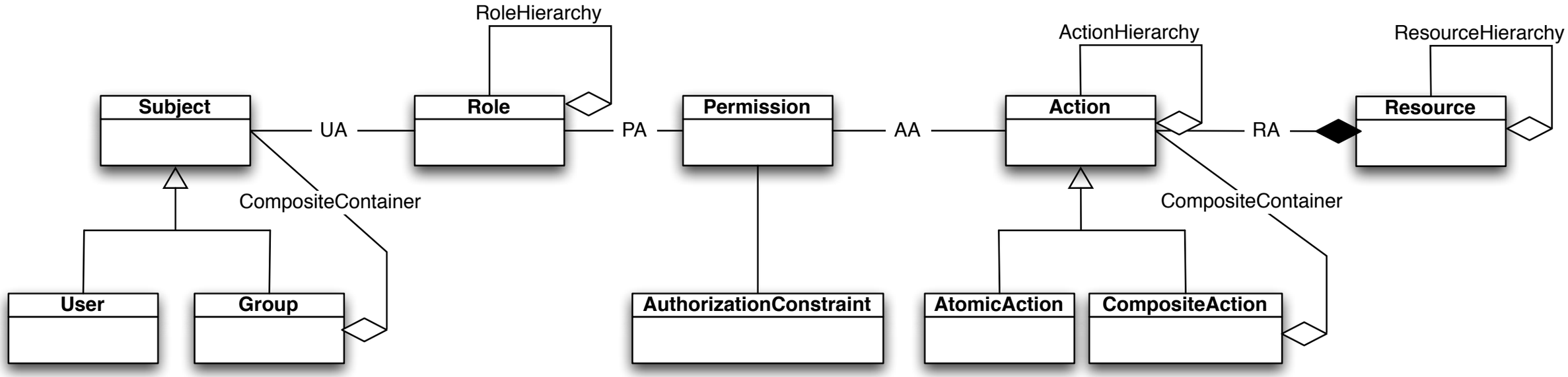
- modeling language



abstract syntax



abstract syntax SecureUML



source: Security Engineering, Prof. D. Basin