

Rule Modeling and Markup Part I

Gerd Wagner

Brandenburg University of Technology
at Cottbus, Germany

Contents

- ❑ What are rules good for? Rules are everywhere!
- ❑ What is model-driven development ("MDA")?
- ❑ Why should you model rules?
- ❑ Can you use rules right away? No: you should first settle the underlying vocabulary.
- ❑ How to model rules with UML+OCL
- ❑ Which rules cannot be modelled with OCL?

Rules play an important role in ...

- Organizations or business systems:
 - for defining derived concepts
 - for constraining concepts
 - for describing, constraining, and prescribing the behavior of business actors
- Computational formalisms:
 - for specifying logical derivations, transformations ("productions") and transitions
- Software systems:
 - for specifying system behavior, e.g. with the help of production rules (OPS5, JESS, ILOG, ...) or ECA rules (SQL triggers, Gensym, ...)

"business rules"

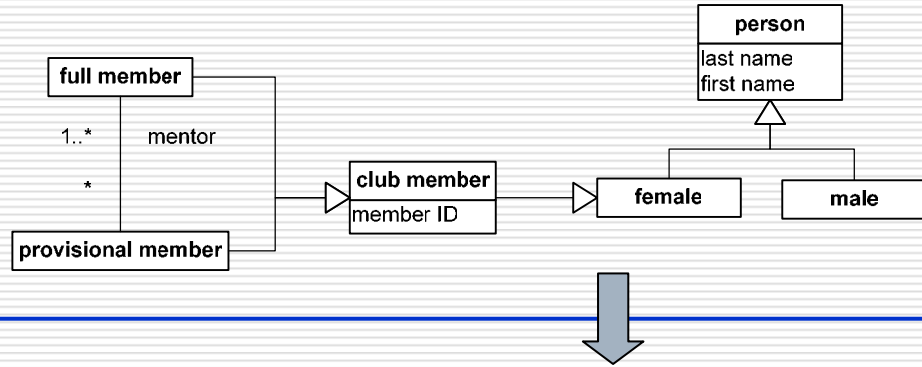
These are the 3 levels of the MDA*:

- ❑ Organizations or business systems: the level of **domain analysis** with "computation-independent modeling" (CIM)
- ❑ Computational formalisms: the level of **operational design** with "platform-independent modeling" (PIM)
- ❑ Software systems: the level of **implementation** with "platform-specific modeling" (PSM)

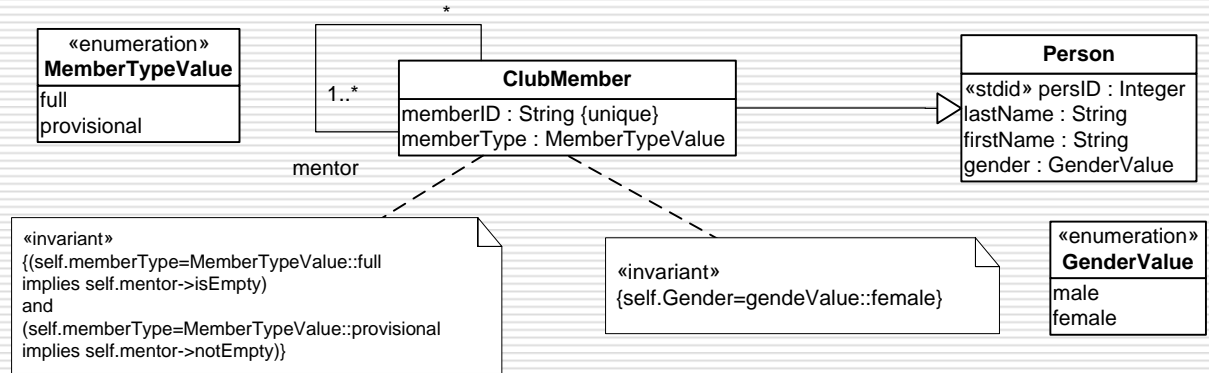
*MDA = The *Model-Driven Architecture* approach of the Object Management Group (OMG)

MDA Model Transformations

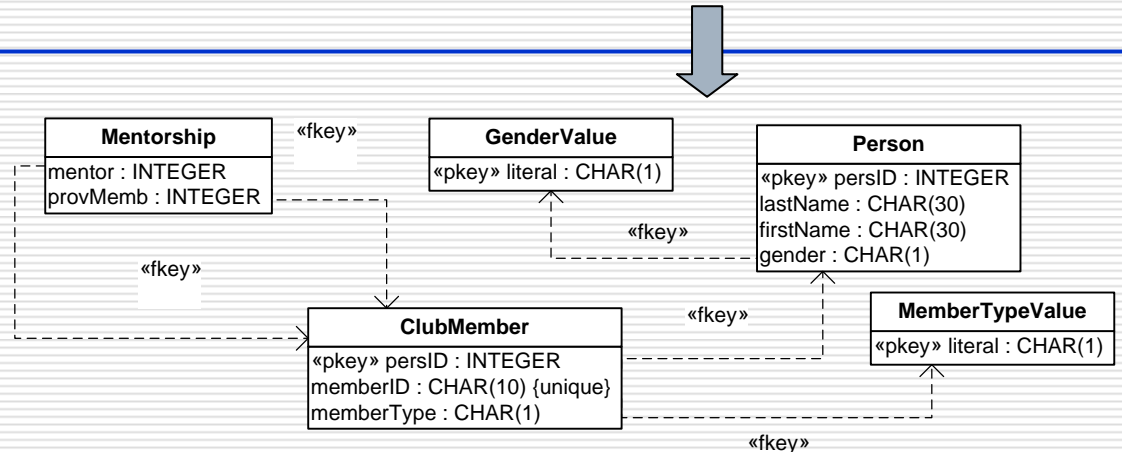
CIM



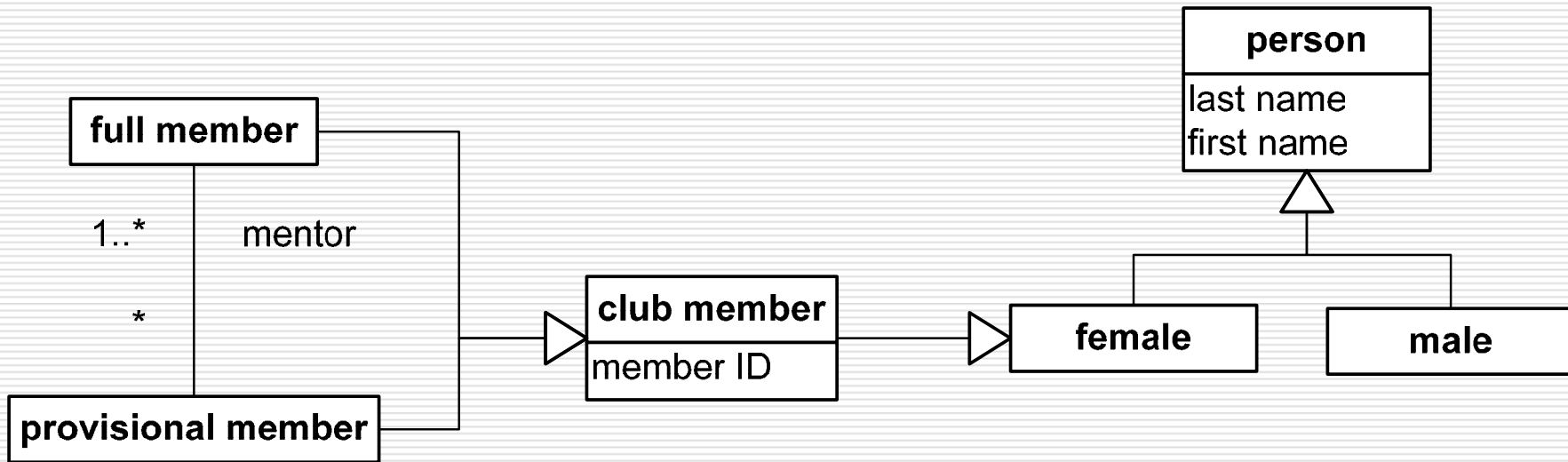
PIM



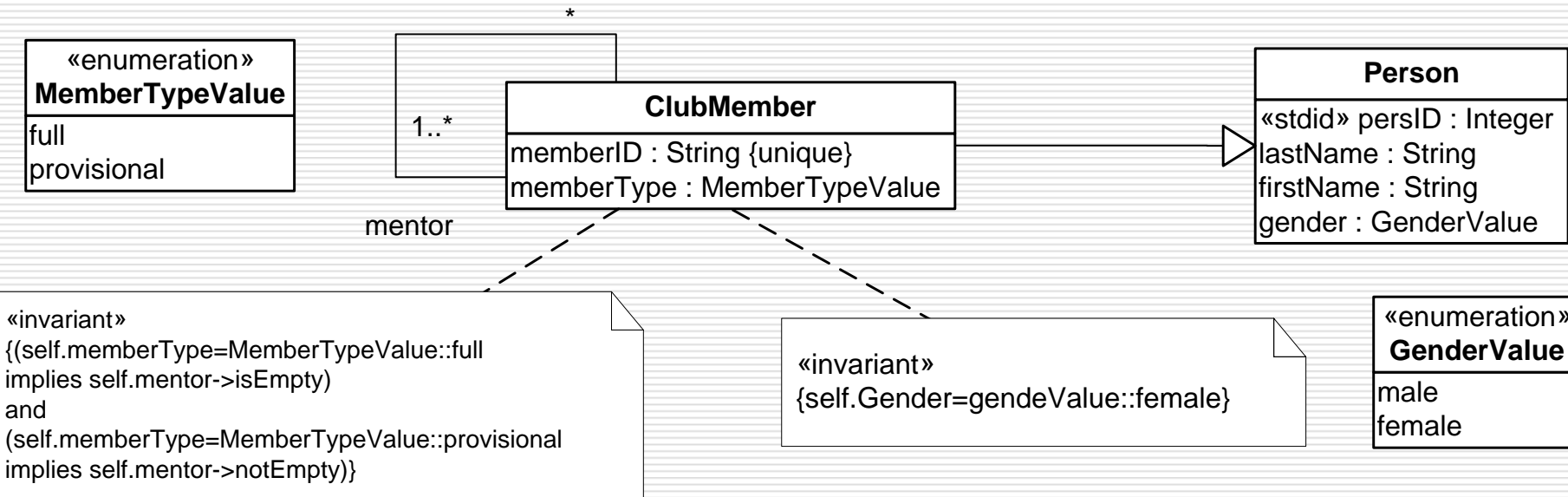
SQL-
PSM



CIM: Domain Model

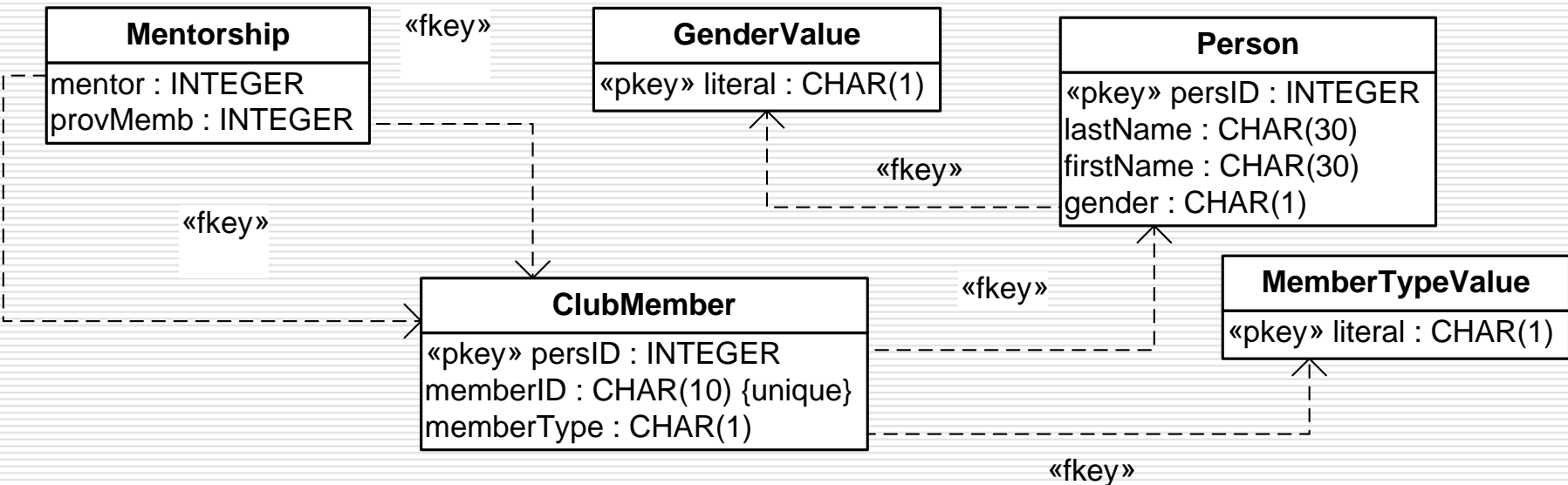


PIM: Platform-Indep. Design Model



PSM: Platf.-Spec. Implem. Model

SQL-PSM

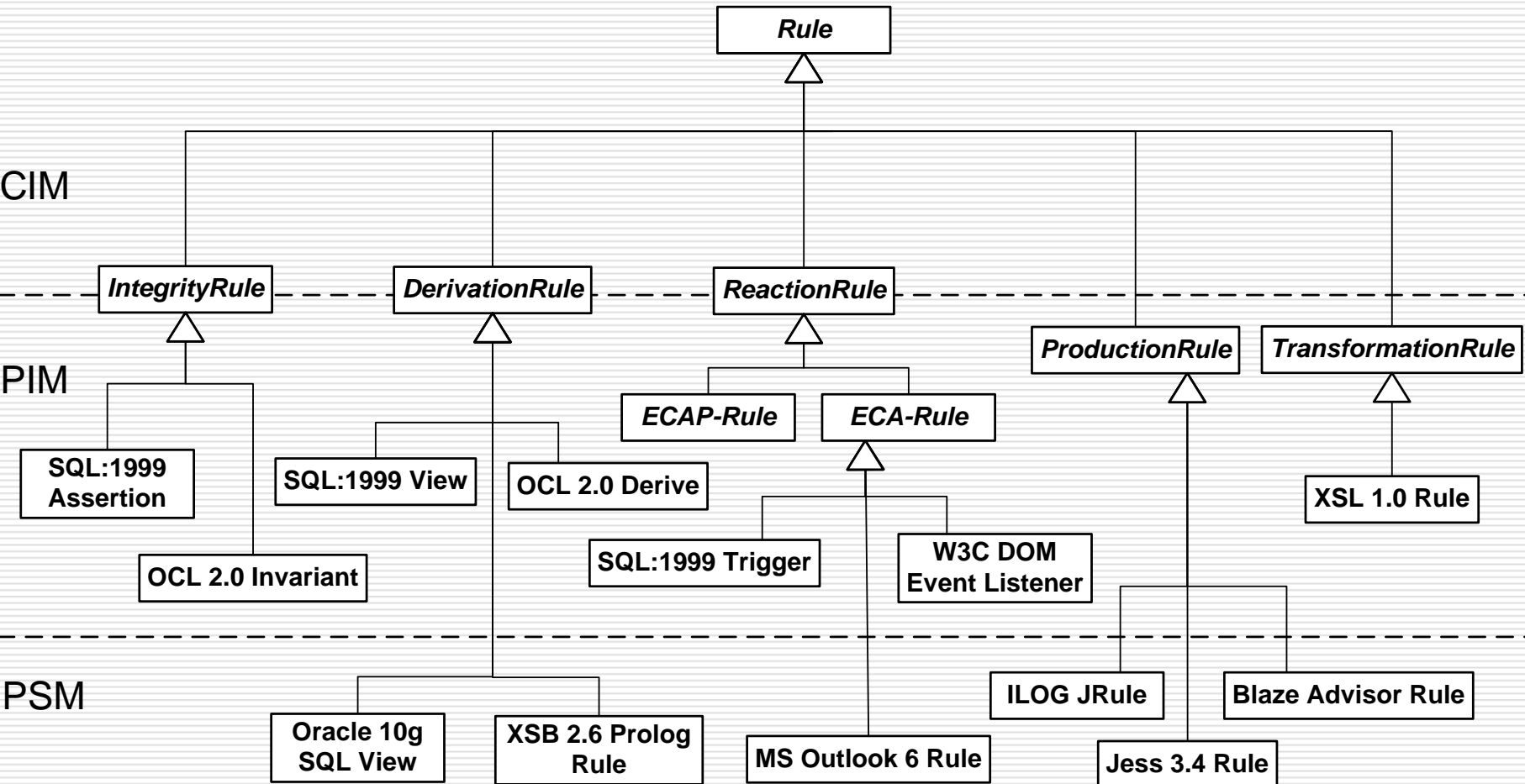


Rules in the MDA

Rules are a general concept used at all three abstraction levels of the MDA:

1. CIM: business rules, policies
2. PIM: formalized rule expressions
3. PSM: executable rule expressions

Rule Concepts at all 3 MDA-Levels



Rules at the Business Domain (CIM) Level

1. The driver of a rental car must be at least 25 years old
2. A gold customer is a customer with more than \$1Million on deposit
3. An investment is exempt from tax on profit if the stocks have been bought more than a year ago
4. When a share price drops by more than 5% and the investment is exempt from tax on profit, then sell it

Why are the *UML* and *MDA* important?

- ❑ Models are *knowledge assets* both in management and in engineering
- ❑ The Unified Modeling Language (UML) and the Model-Driven Architecture (MDA) support *knowledge management* in software engineering processes

Why should we model rules?

- ❑ A software engineering syllogism:
 1. For constructing an information system, it is **good practice** to model the business system and the information system
 2. Rules are part of these systems

 3. Consequently, it is good practice to model rules
- ❑ Rule modeling is part of the general model-driven approach to software and information systems engineering

What about "rule-based systems"?

- ❑ Of course, we may also want to model the rules of a rule-based software system
- ❑ But rule-based software systems are just a particular implementation platform class, ...
- ❑ ... which is not very widely used

Rules come on top of a vocabulary

- Rules are expressed with the help of sentential **atoms**, whose constituents are **elements of a vocabulary**
- A business rule is based on a business vocabulary

Vocabularies can be expressed...

- **Textually** as terminologies/taxonomies
 - in English, as proposed by **ISO 704**
 - in 'Structured' English, as proposed by the **SBVR** submission to **OMG**'s BSBR RfQ
- **Visually** as conceptual class diagrams in UML, as proposed by the **OMG**
- **Formally** as "signatures" in predicate logic or as "ontologies" in RDF and OWL, as proposed by the **W3C**

Vocabularies and the MDA

<i>Vocabulary Language</i>	<i>CIM</i>	<i>PIM</i>	<i>PSM</i>
Controlled English	√		
UML Class Diagrams	√	√	√
RDFS / OWL		√	√

Example 1

Rule expression:

Each additional driver of a car rental must be a qualified driver.

Vocabulary:

1. Class terms:

car rental

qualified driver (is a **person**)

2. Fact type expressions (predicates):

person is driver of **car rental**

person is additional driver of **car rental**

Example 1: Vocabulary as Class Diagram

Class terms:

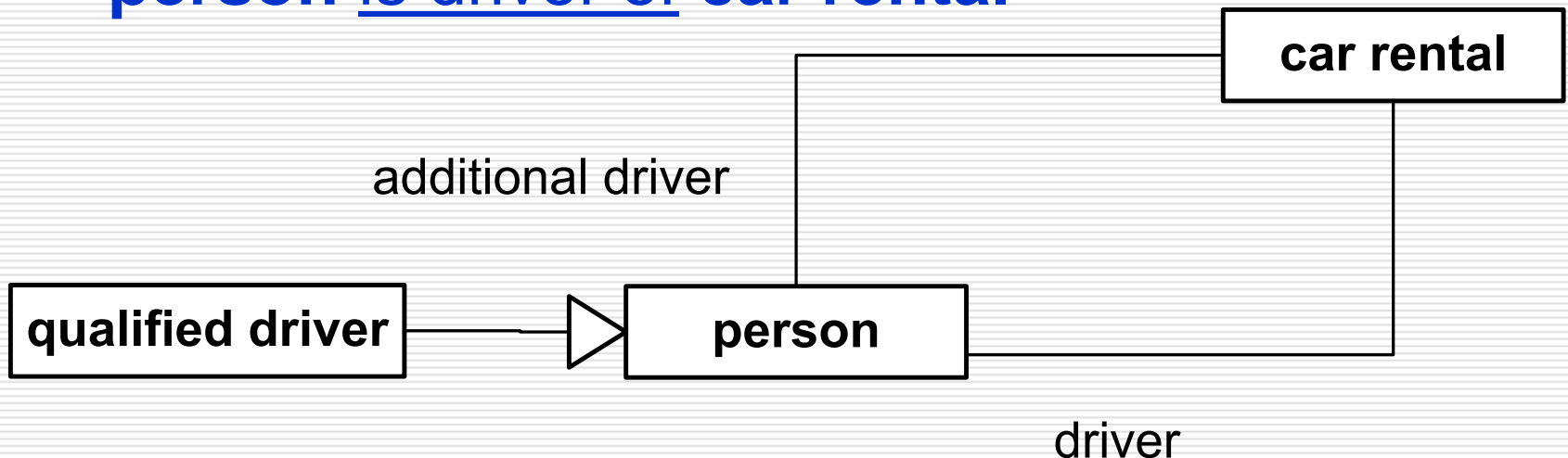
car rental

qualified driver (is a person)

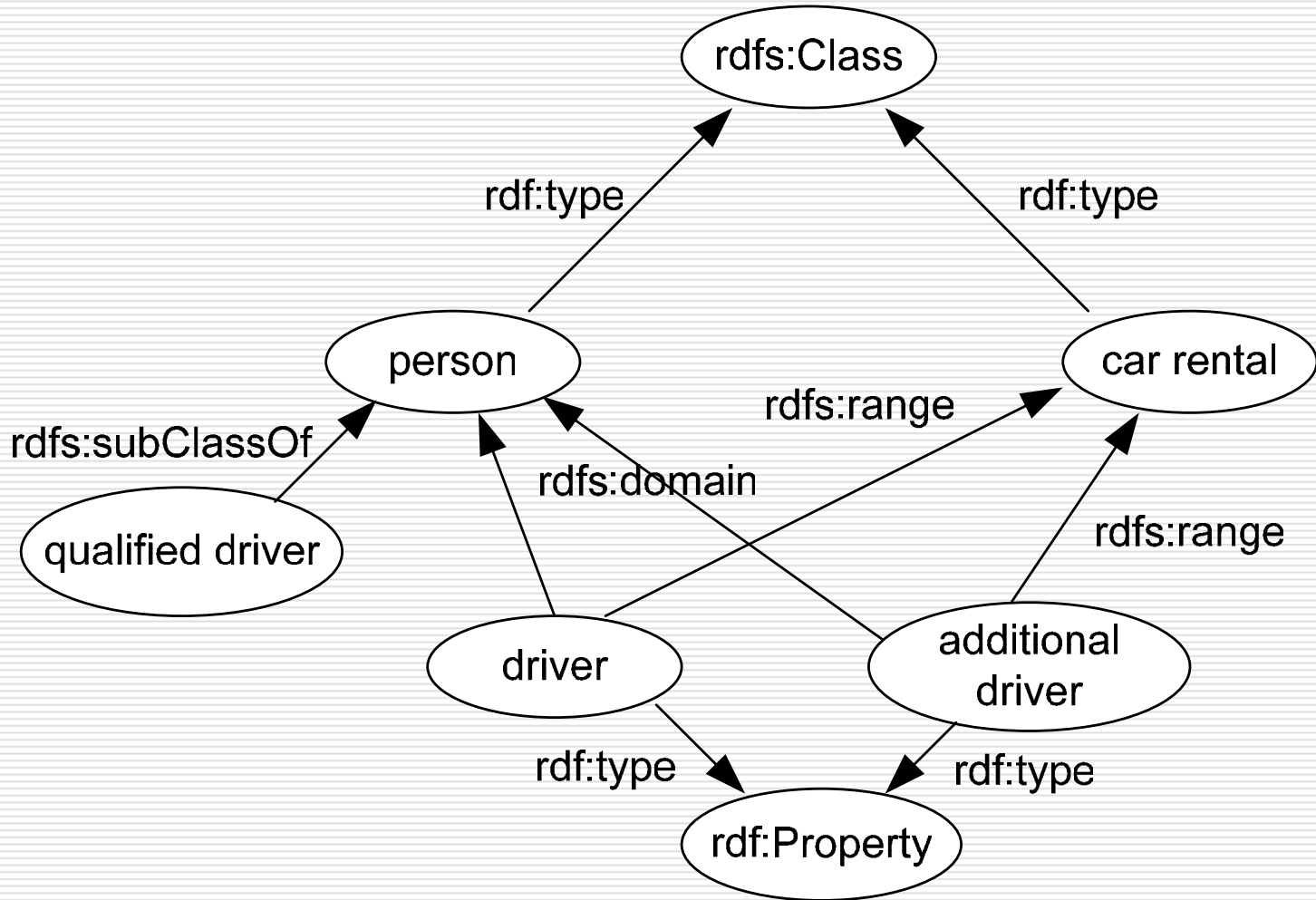
Fact type expressions (predicates):

person is additional driver of **car rental**

person is driver of **car rental**



Example 1: Vocabulary as RDF Graph



Example 1: Vocabulary as RDF/XML

```
<rdf:RDF xmlns:rdf="..." xmlns:rdfs="..." xmlns="..." >
<rdfs:Class rdf:ID="Person"/>
<rdfs:Class rdf:ID="QualifiedDriver">
  <rdfs:subClassOf rdf:resource="#Person"/>
</rdfs:Class>
<rdfs:Class rdf:ID="CarRental"/>
<rdf:Property rdf:ID="driver">
  <rdfs:domain rdf:resource="#CarRental"/>
  <rdfs:range rdf:resource="#Person"/>
</rdf:Property>
<rdf:Property rdf:ID="additionalDriver">
  <rdfs:domain rdf:resource="#CarRental"/>
  <rdfs:range rdf:resource="#Person"/>
</rdf:Property>
```

What did you like more?

The vocabulary represented as:

- UML class diagram?
- RDF graph?
- RDF/XML?

I claim that UML **class diagrams** are more **concise** and more **readable** (provided that you are familiar with all 3 languages).

Example 1 (cont.)

Original rule expression:

Each additional driver of a car rental must be a qualified driver.

Reformulations:

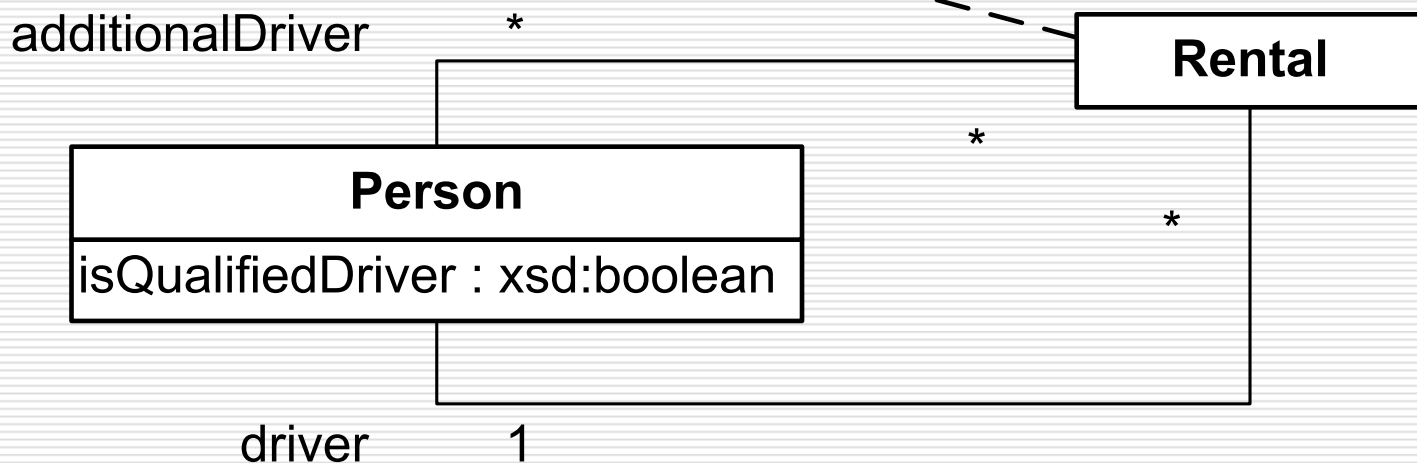
1. IF **person** is additional driver of car rental
THEN **person** is a **qualified driver**
2. IF **person** is a **additional driver**
THEN **person** is a **qualified driver**

Example 1: Express the rule with OCL

IF **person** is additional driver of car rental
THEN **person** is a qualified driver

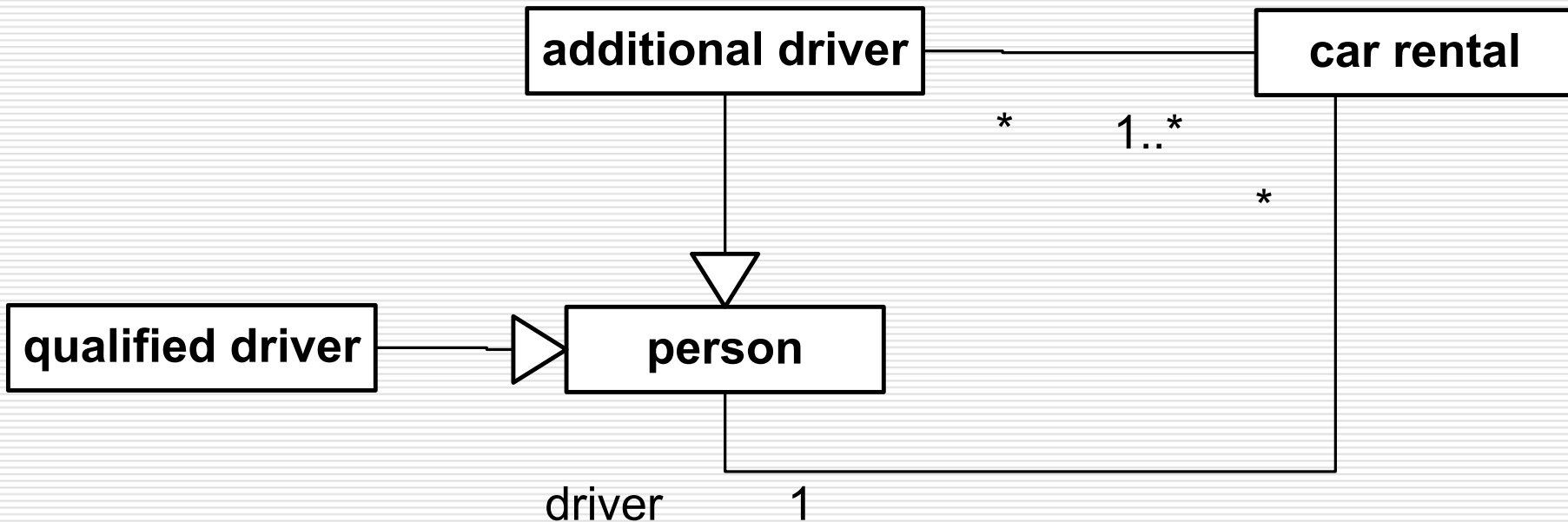
```
«invariant»  
{additionalDriver->forAll( p |  
p.isQualifiedDriver)}
```

This is the OCL expression
of the rule



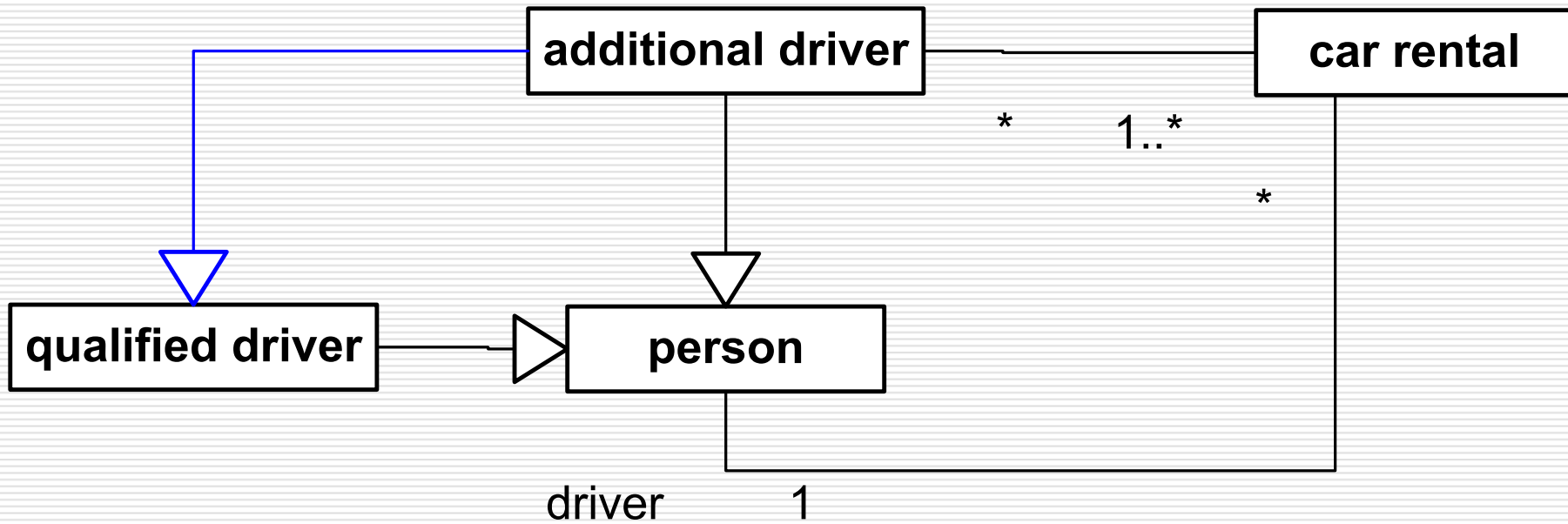
Example 1: Express the rule graphically

Use the role type **additional driver** instead of the corresponding role name:



Example 1: Express the rule graphically

IF **person** is a **additional driver**
THEN **person** is a **qualified driver**



Example 2

Rule expression:

A qualified driver is a person with a driver's license who is more than 24 years old.

Vocabulary:

1. Class terms:

qualified driver (is a **person**)

person with a driver's license (is a **person**)

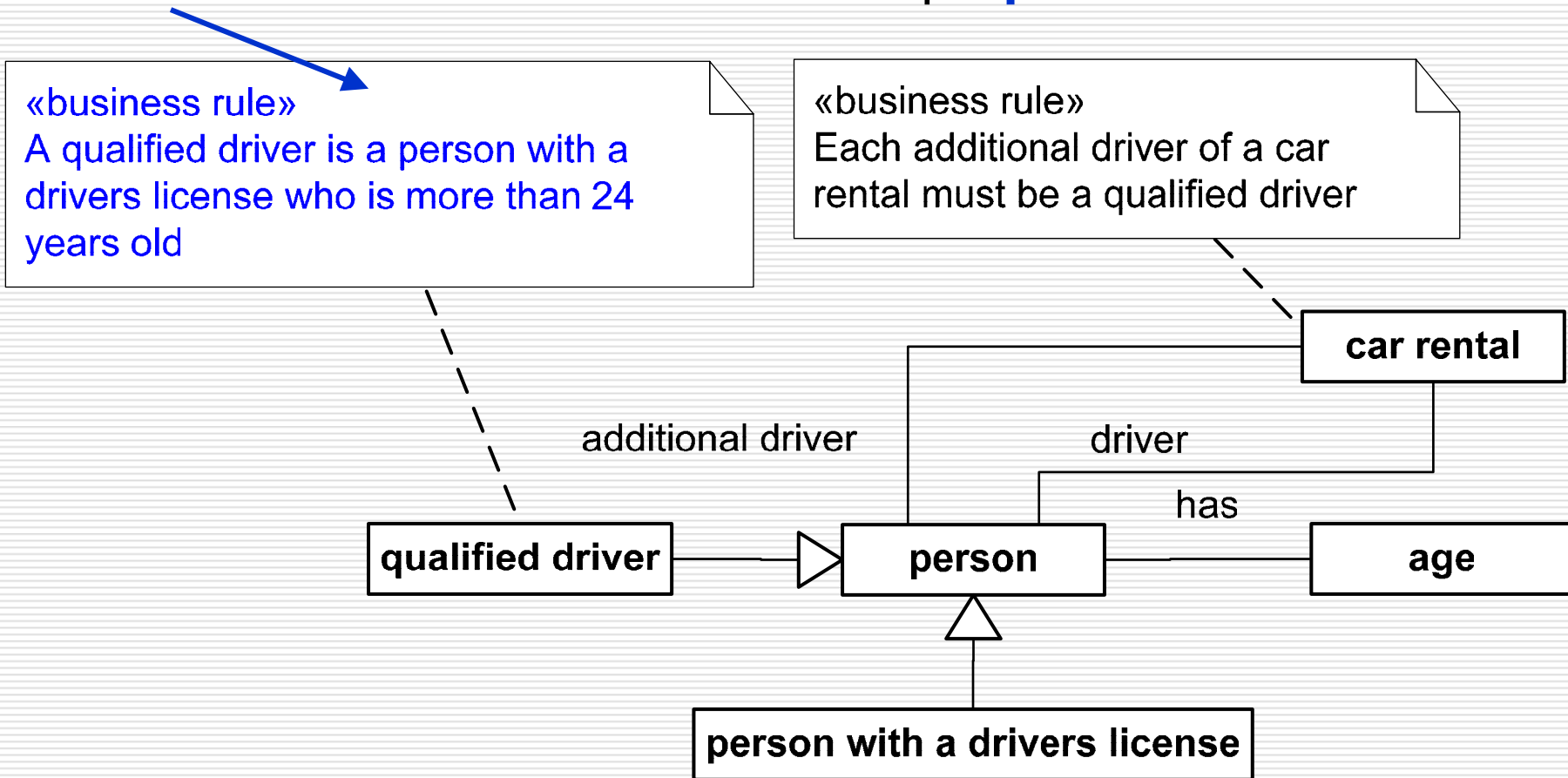
2. Fact type expressions (predicates):

person has **age**

(or **integer** is age of **person**)

Example 2: A CIM with two rules

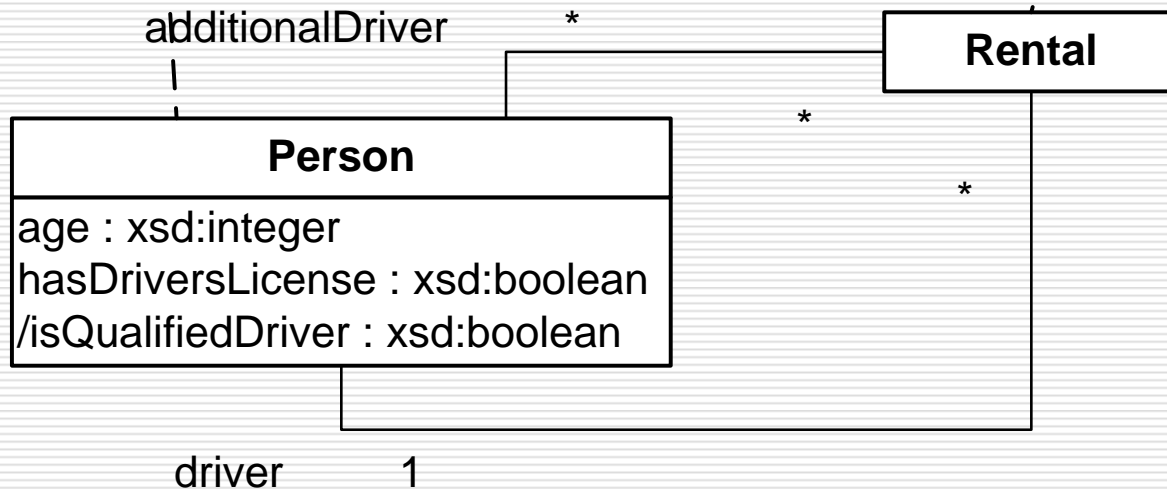
This rule defines the derived concept **qualified driver**.



Example 2: The corresponding PIM

«derivation rule»
context isQualifiedDriver : xsd:boolean derive:
self.hasDriversLicense and age>24

«invariant»
{additionalDriver->forall(p |
p.isQualifiedDriver)}



Exercise 1: Case Financial Trading

1. Read the handout and try to understand the vocabulary used.
2. Try to understand the rule statements.
3. Can you classify the rules?
 - a) Which of them are integrity rules?
 - b) Which of them are derivation rules?
 - c) Are there any other rules?

A Short Introduction into OCL

- ❑ OCL = Object Constraint Language
- ❑ OCL can be used to express integrity rules (invariants), derivation rules and queries in the context of a UML model in a **precise** and **unambiguous** manner.
- ❑ Invarianten are implicitly universally quantified sentences ("for all instances of the context,...")
- ❑ All OCL expressions are related to a specific **context element** such as a specific class or a specific association. The keyword **self** refers to an instance of the context element, that is, to an object or to a link.

OCL Invariants

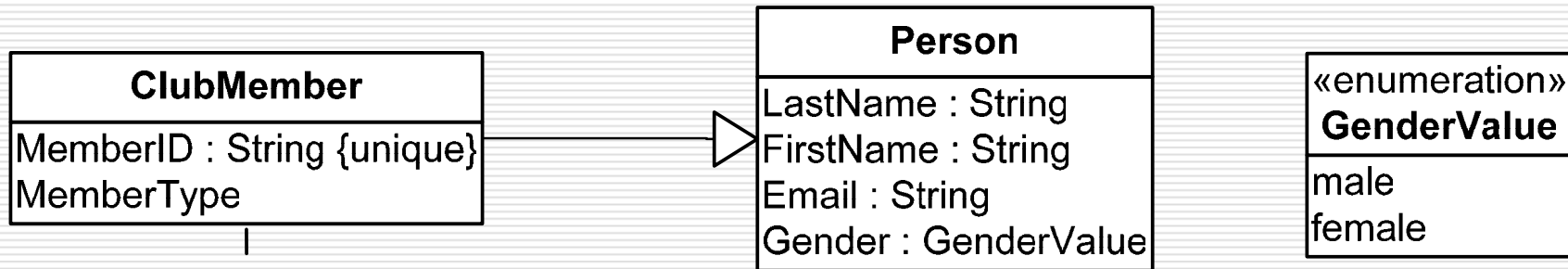
- ... are related to a specific **context** such as a specific class or an association
- ... can be attached to a modeling element in a UML diagram, or can be specified textually using the format

CONTEXT ClubMember INV:

self.Age > 18

"Club members must be more than 18 years old"

Using an enumeration literal in an OCL invariant

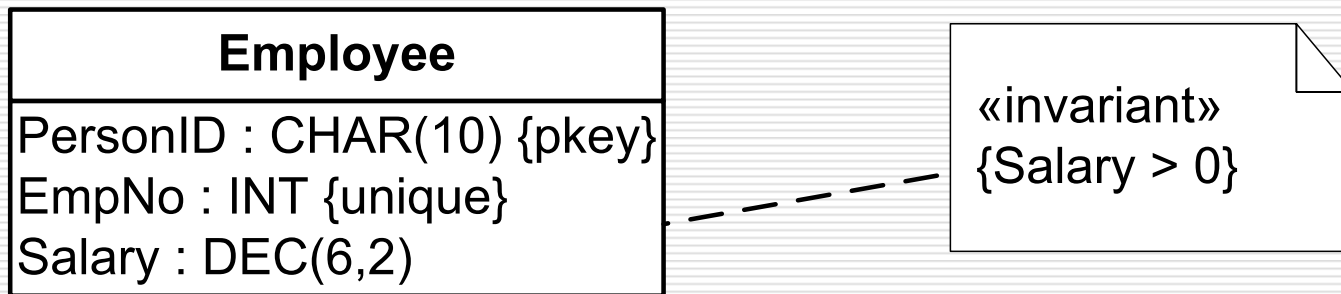


«invariant»
{self.Gender = GenderValue::female}

CONTEXT ClubMember INV:
self.Gender =
GenderValue::female

Instead of stating in ordinary language:
"All club members must be female"

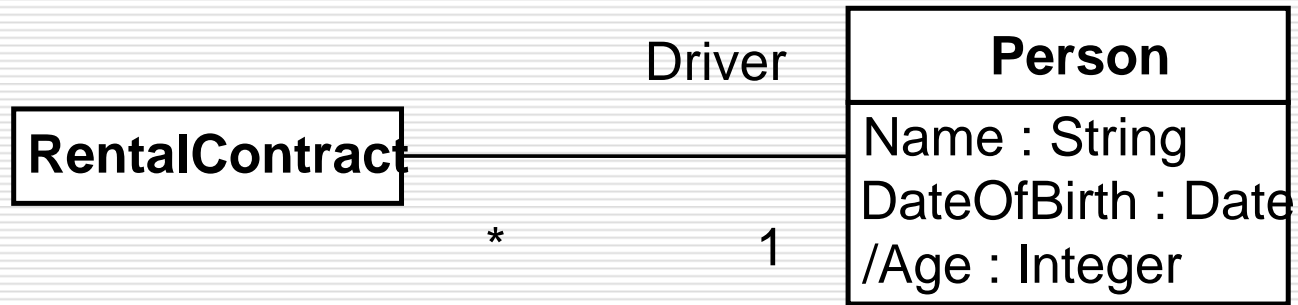
A simple invariant constraining an attribute value



CONTEXT Employee INV:
`self.Salary > 0`

`self` can be omitted

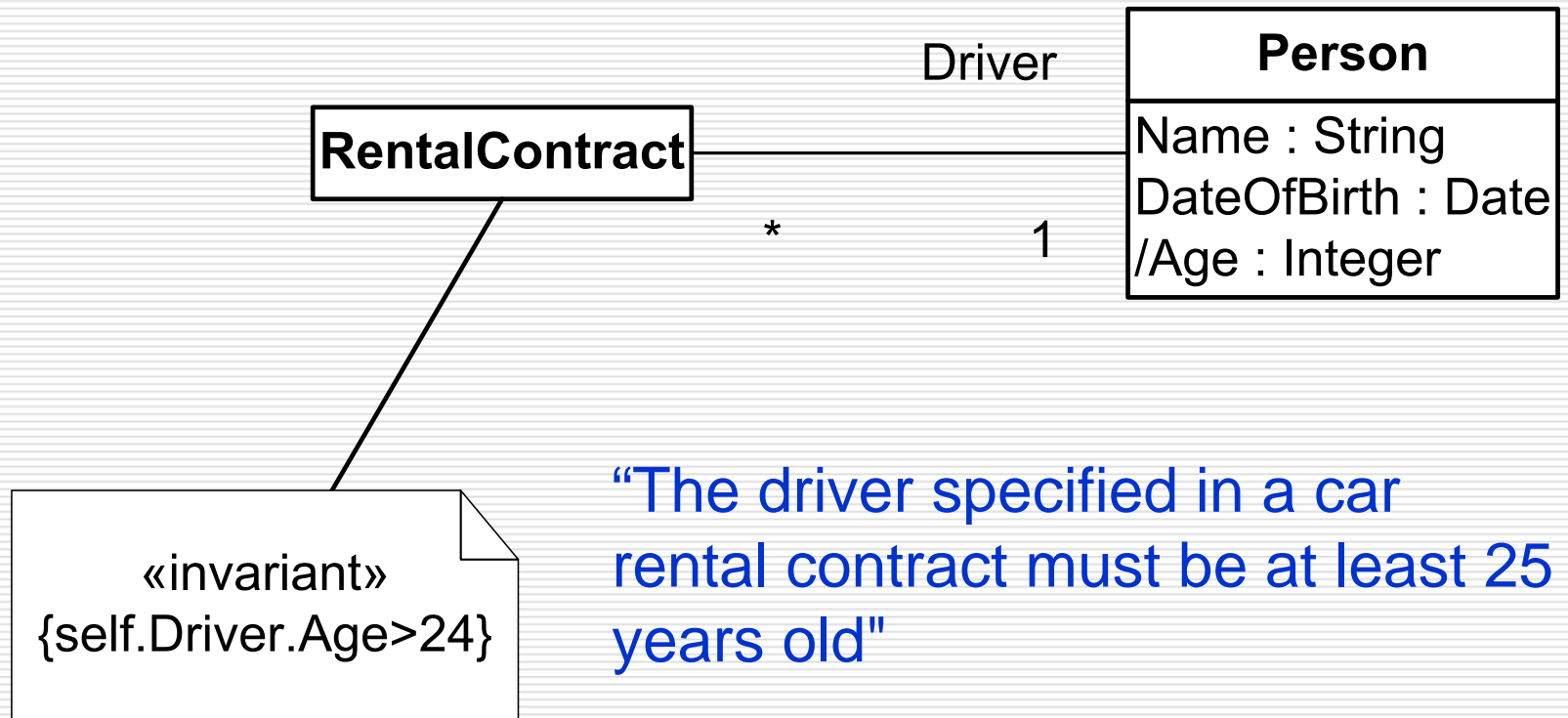
OCL Path Expressions



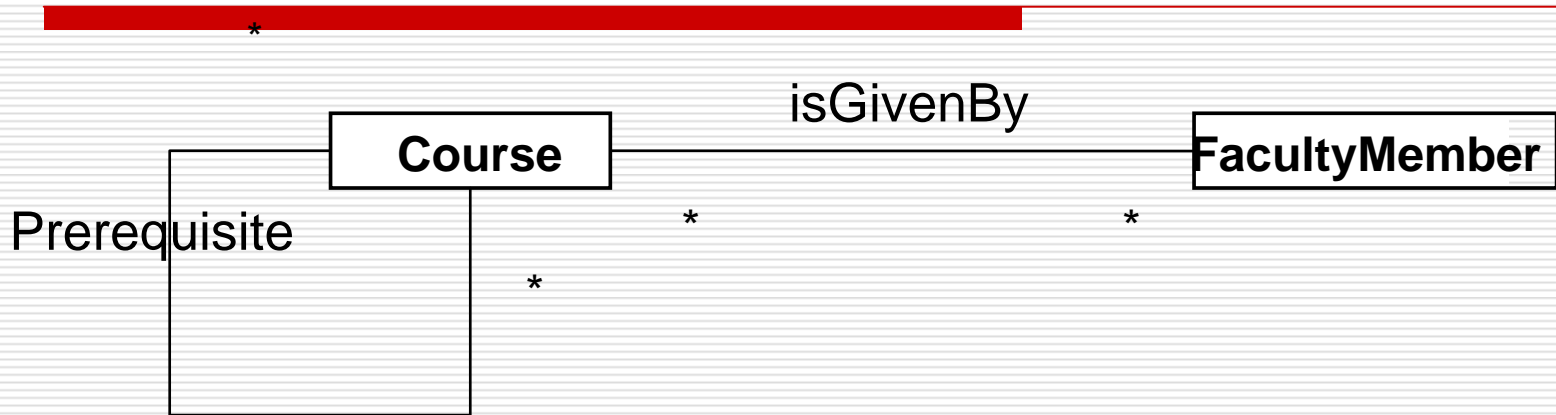
In the context of `RentalContract`:

- `self.Driver` refers to the set of `Person` instances having a `Driver` link with the context `RentalContract` object (those persons playing the role of a driver) – this is exactly one `Person` instance here
- `self.Driver.Name` refers to the `Name` attribute of the `Person` instance that is the `Driver` of the context `RentalContract` object

Using a functional role name for referring to a single associated object



In the context of the **Course** class:

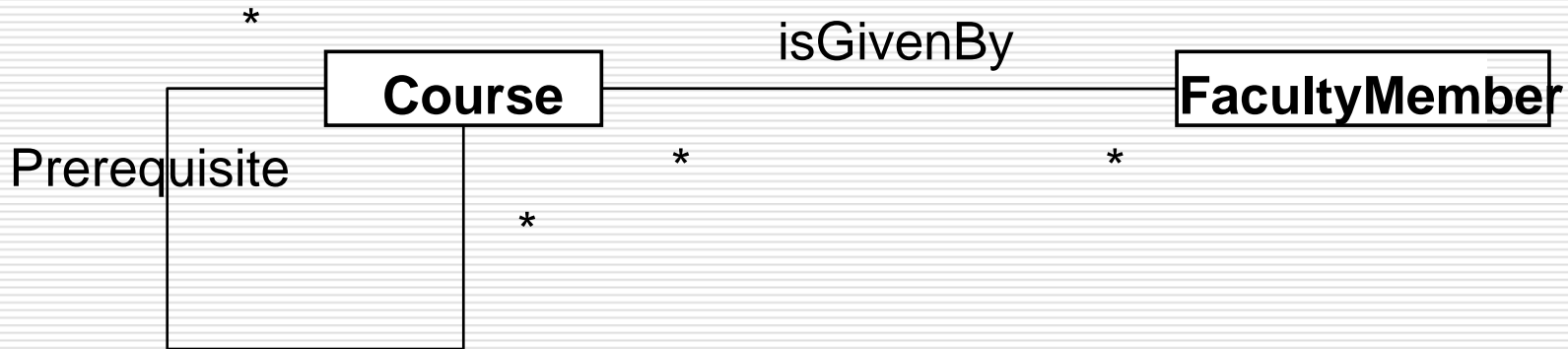


- ❑ **self.Prerequisite** refers to the set of all Course instances that have a Prerequisite link with the given Course instance
- ❑ **self.FacultyMember** refers to the set of all FacultyMember instances that have an isGivenBy link with the given Course instance

Operations on collections

- $c \rightarrow \text{op}(\dots)$ denotes the application of a collection operation $\text{op}(\dots)$ to a collection c
- $\text{isEmpty}()$ is a Boolean-valued OCL operation that tests if a collection is empty
- $\text{size}()$ is an OCL operation that provides the size of a collection
- $\text{sum}(x)$ is an OCL operation that computes the sum of a collection of numbers
- $\text{includes}(x)$ is a Boolean OCL operation that tests if a collection contains an instance x
- $\text{forAll}(o1, o2, \dots \mid \text{condition})$ is a Boolean OCL operation that tests if all members of a collection satisfy a condition

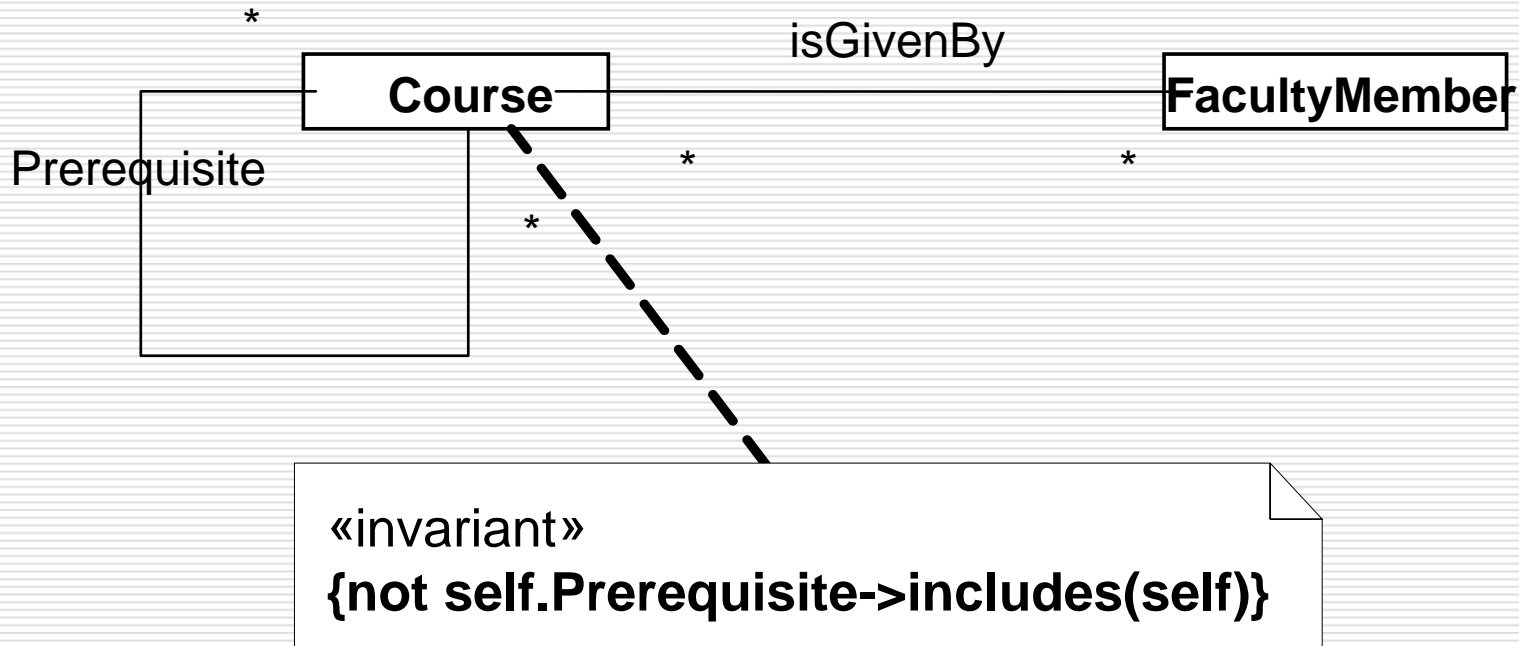
In the context of the **Course** class:



`self.FacultyMember->size()`

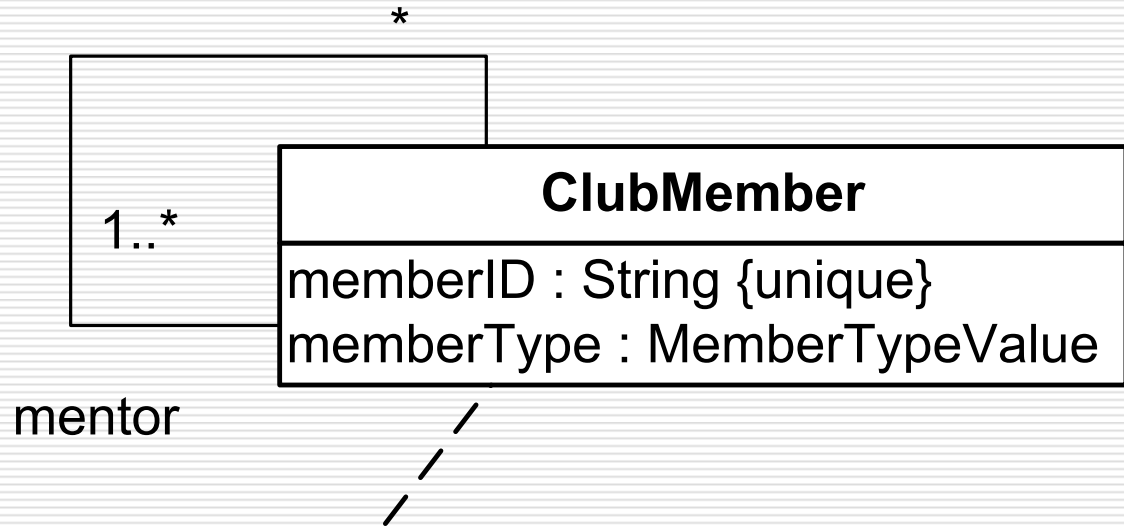
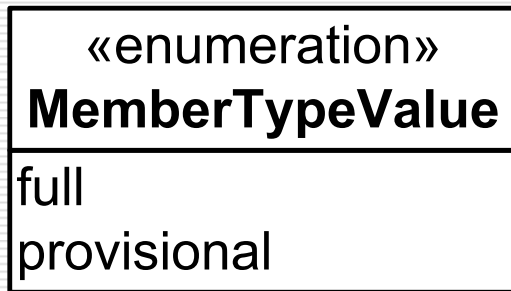
denotes the number of FacultyMember instances linked to the given Course instance

Using a non-functional role name referring to a set of instances



“A course must not be its own prerequisite”

Restricted Universal Quantification



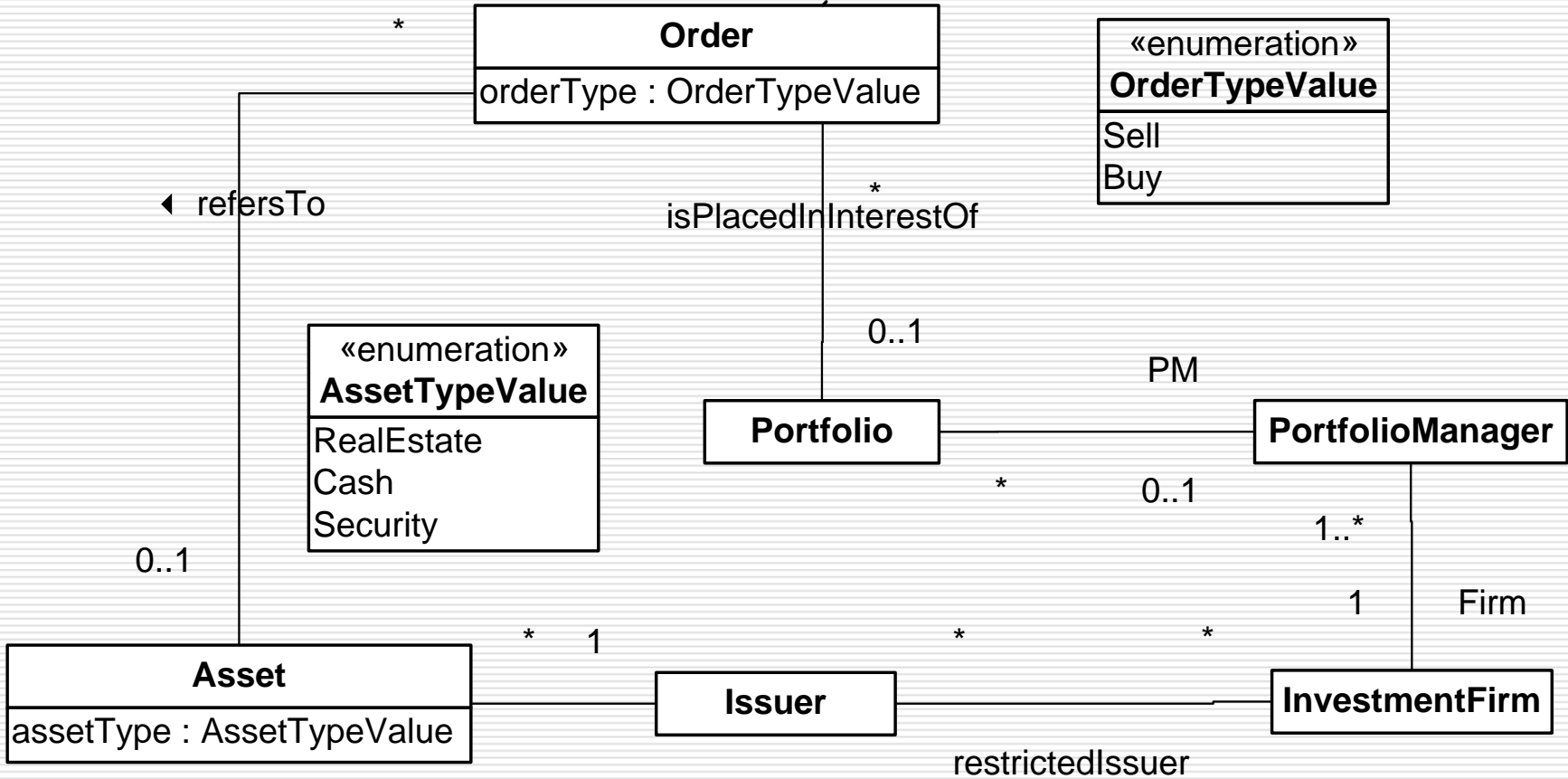
«invariant»
{self.mentor->forAll(m | m.memberType=MemberTypeValue::full)}

“Mentors must be full members”

"No securities issued by a restricted issuer must be bought"

«invariant»
 {orderType = OrderTypeValue::Buy
 and Asset.assetType = AssetTypeValue::Security
 implies
 Portfolio.PM.Firm.restrictedIssuer->excludes(Asset.Issuer)}

R1



A portfolio is rated *platinum*, if its value is greater than 1 Mio \$. It is rated *gold*, if its value is less than 1 Mio \$ and greater than 100.000 \$. It is rated *regular*, if its value is less than 100.000\$.

```
context Portfolio::status : PortfolioStatusValue
```

```
derive: if value >= 1000000
```

```
  then PortfolioStatusValue::Platinum
```

```
  else if value >= 100000
```

```
  then PortfolioStatusValue::Gold
```

```
  else PortfolioStatusValue::Regular
```

```
endif endif
```

R5

Portfolio

creationDate : Date

/value : Real

/status : PortfolioStatusValue

«enumeration»

PortfolioStatusValue

Platinum

Gold

Regular

What about other types of rules?

UML/OCL can only express

- Integrity rules
- Derivation rules

UML/OCL cannot express

- production rules
- ECA/reaction rules

R6 is an example of an ECA/reaction rule:
 The owner of a portfolio must be sent a dispose recommendation if there is a downgrade for a security held in it.

