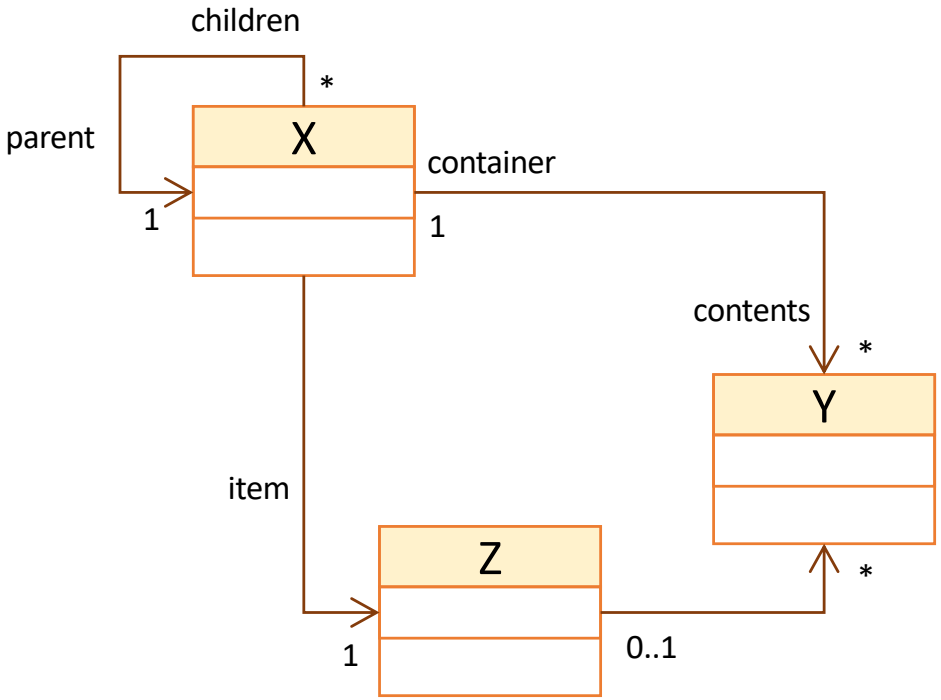
Monitoring Association Constraints in Model-Oriented Programming

Sylvain Guérin, Joël Champeau, Antoine Beugnard, Salvador Martinez

5th International Workshop on Modeling Language Engineering, MODELS 2023

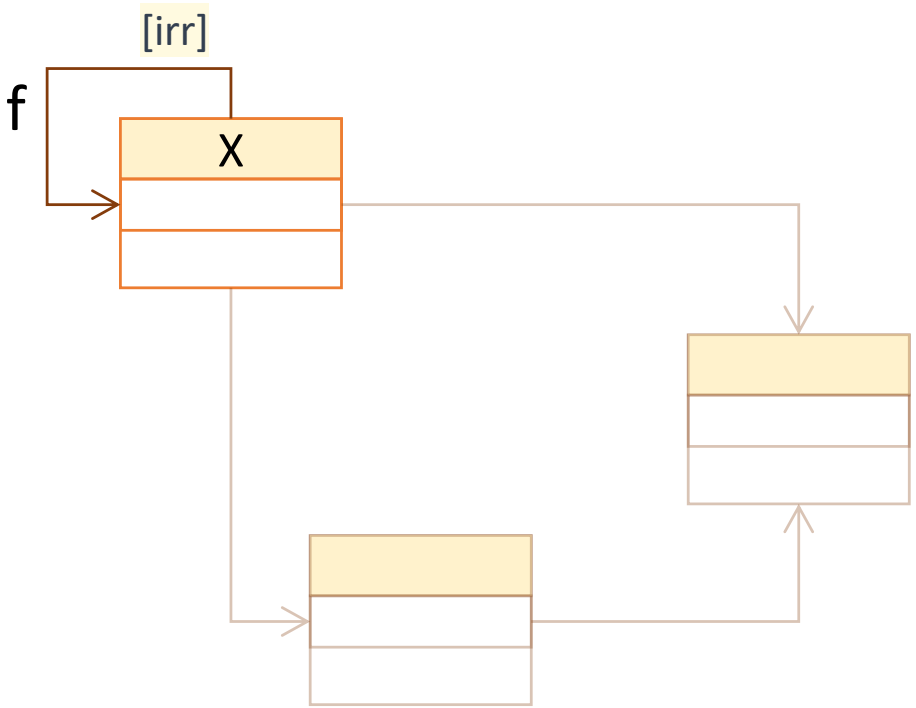
Agenda

1. Context and challenges
2. Model-Oriented Programming
3. Our approach
4. Examples
5. Conclusions and perspectives



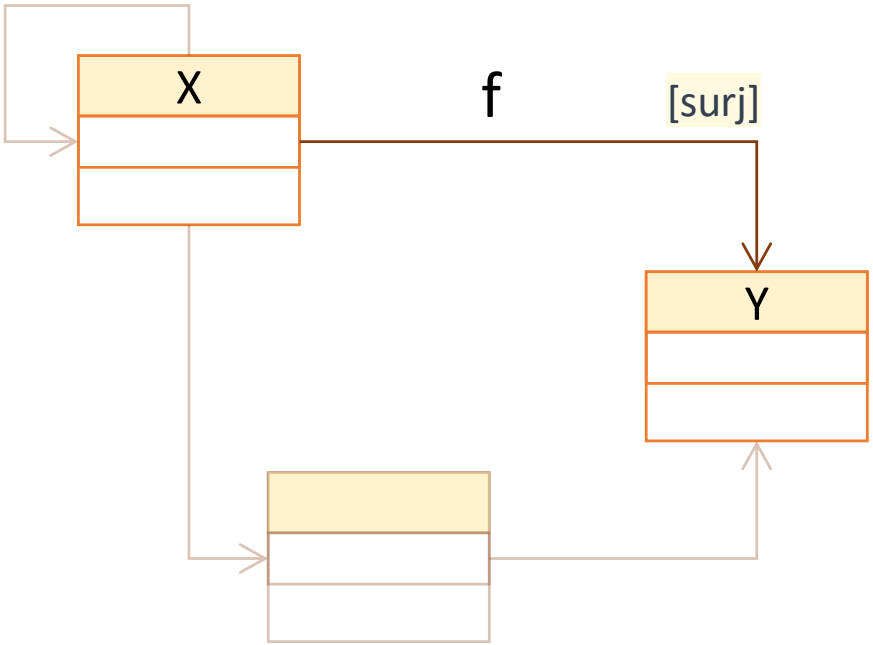
Semantics of relations in modeling languages ?

- ✓ Cardinalities
- ✓ Navigability
- ? More complex constraints ?



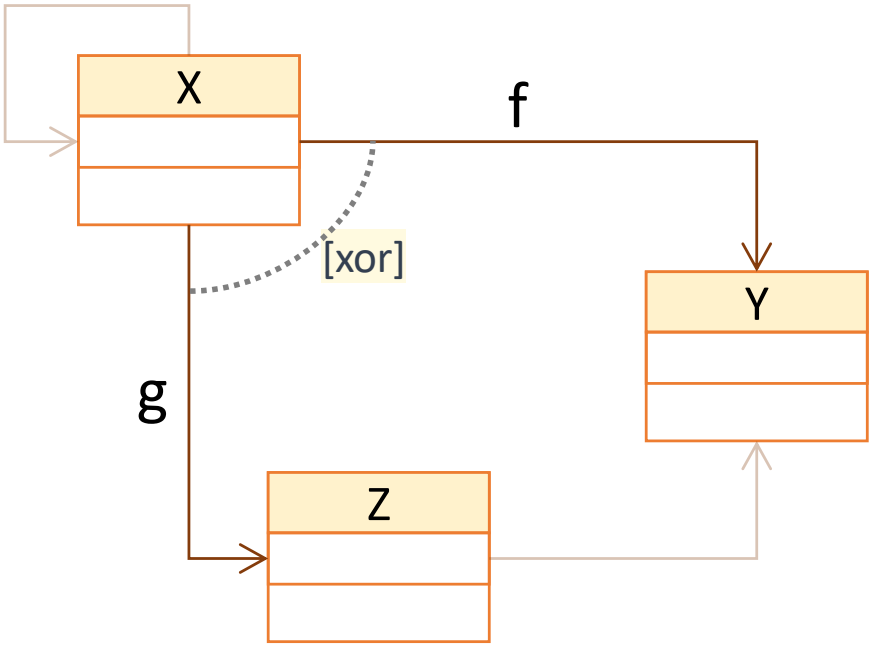
Irreflexive relation

$\forall x \in X, x \notin f(x)$



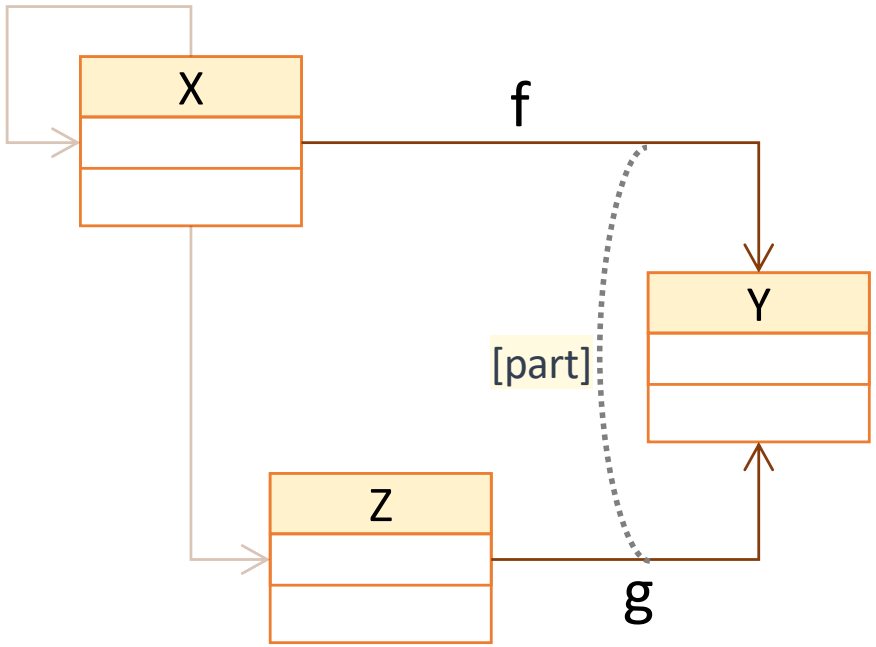
Surjective relation

$f(X) = Y$



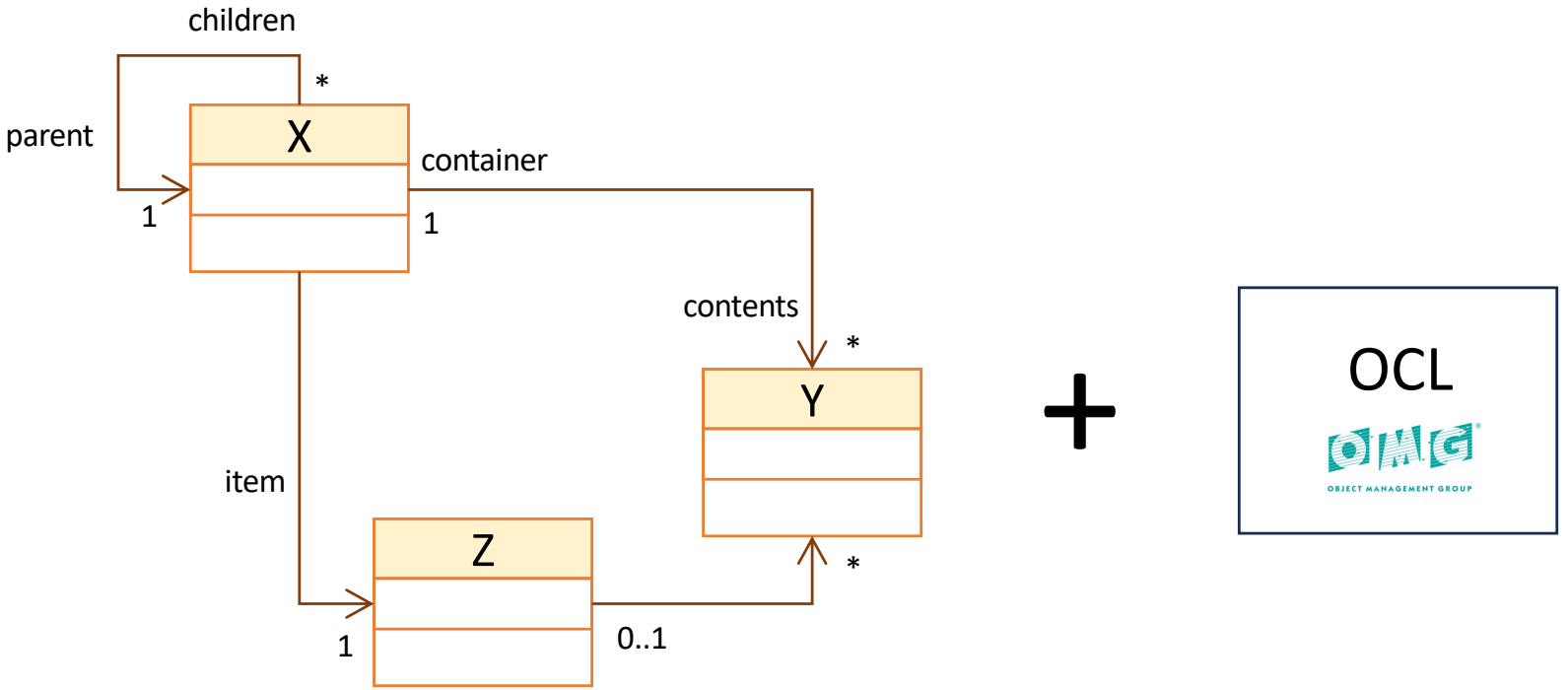
XOR relation

$$\forall x \in X, (f(x) = \emptyset \vee g(x) = \emptyset) \wedge (f(x) \neq \emptyset \vee g(x) \neq \emptyset)$$



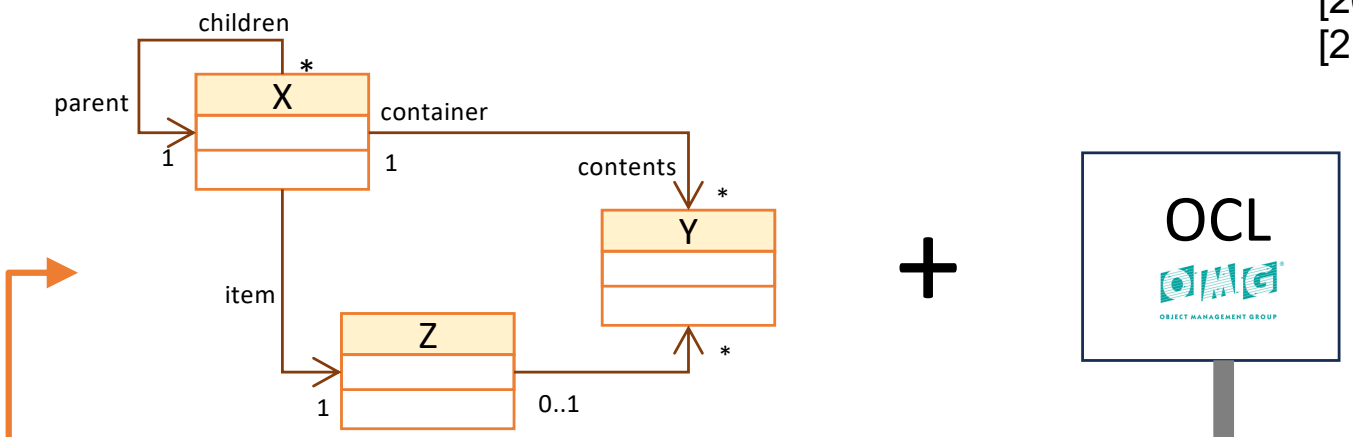
Partition relation

$$f(X) \cap g(Z) = \emptyset$$
$$\wedge\ f(X) \cup g(Z) = Y$$



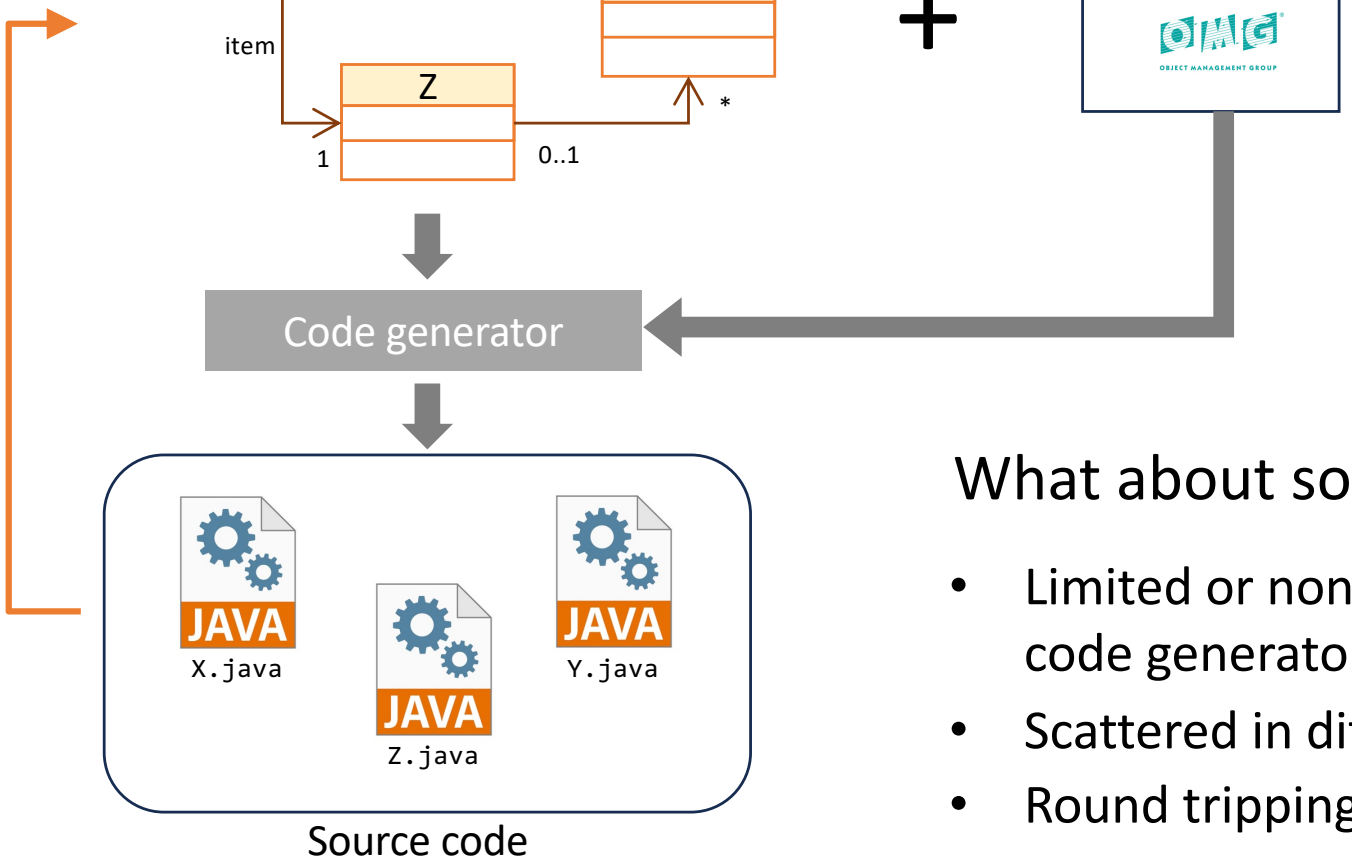
A solution : external constraint language

[4] D. Gessenharter (UML2)
[8] T. C. Lethbridge et al (UMPLE)
[20] D. Costal et al (UML)
[21] Y. Lamo et al. (DPF)



+

OCL
OMG
OBJECT MANAGEMENT GROUP



What about source code level ?

- Limited or non-existing support in code generators
- Scattered in different parts of the code
- Round tripping ?

Summary of challenges

- Modeling issues
  - Limited support in modeling languages
  - Limited support in code generators, even if an external constraints language is provided

Improve expression power

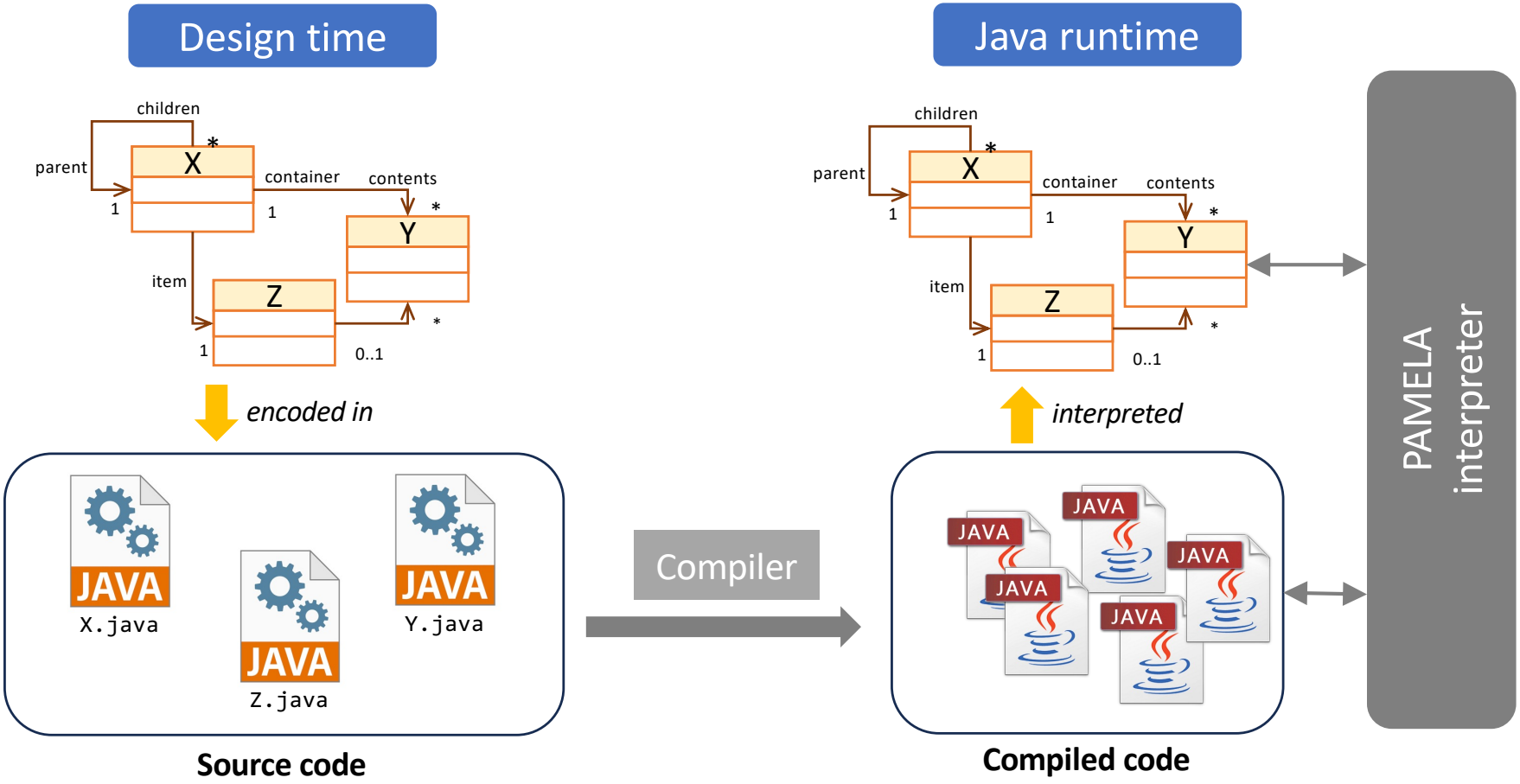
- Code generation issues
  - Scattered in different parts of the code
  - What if the developer modifies the code? No guarantee of constraint non-violation
  - Maintenance and round-trip

Model-oriented programming

Agenda

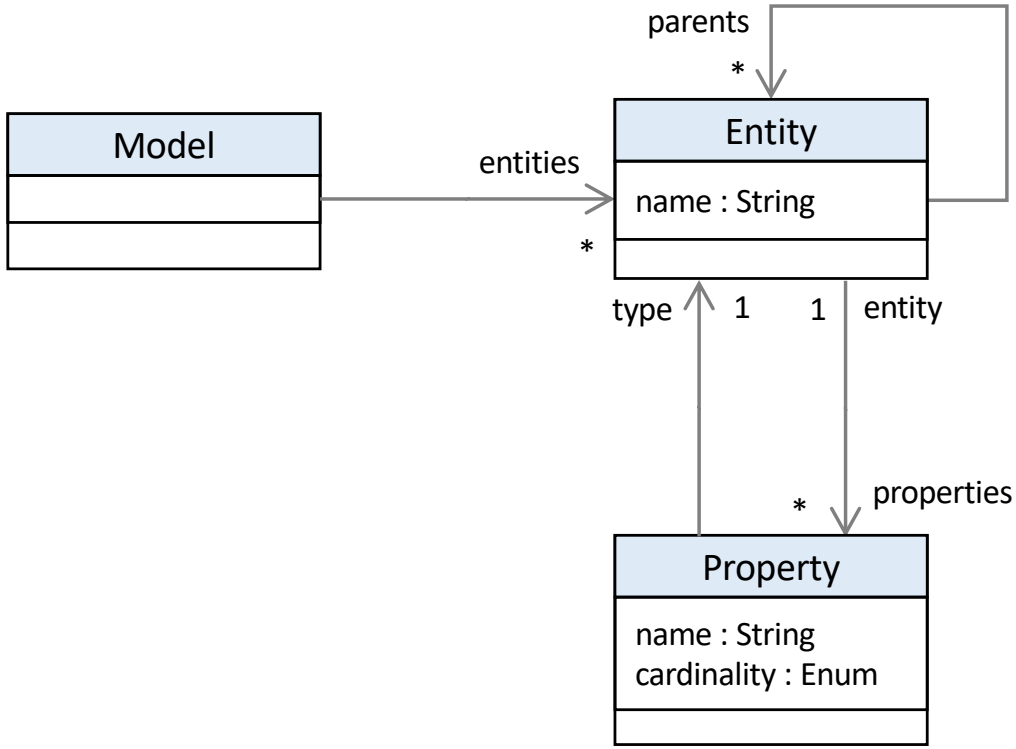
1. Context and challenges
2. Model-Oriented Programming
3. Our approach
4. Examples
5. Conclusions and perspectives

Model-Oriented Programming with PAMELA



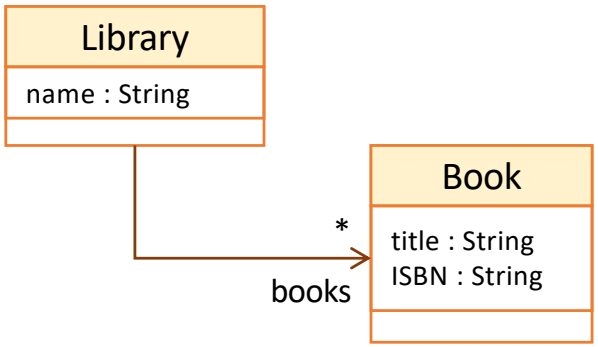
- “model” is encoded as source code meta-data (java annotations)
- “model” is interpreted at run-time

(simplified) PAMELA metamodel



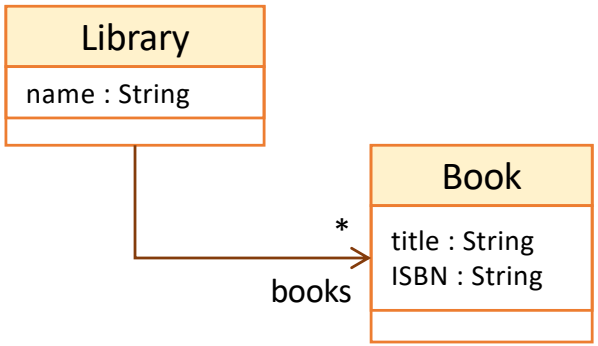
Modeling with PAMELA

```
@ModelEntity
public interface Book extends AccessibleProxyObject {

    @Initializer
    public Book init(@Parameter("title") String aTitle);

    @Getter("title")
    public String getTitle();

    @Setter("title")
    public void setTitle(String aTitle);

    @Getter("ISBN")
    public String getISBN();

    @Setter("ISBN")
    public void setISBN(String value);

}
```




```
@ModelEntity
public interface Library extends AccessibleProxyObject {

    @Getter("name")
    public String getName();

    @Setter("name")
    public void setName(String aName);

    @Getter(value = "books", cardinality = Cardinality.LIST)
    @Embedded
    public List<Book> getBooks();

    @Adder("books")
    public void addToBooks(Book aBook);

    @Remover("books")
    public void removeFromBooks(Book aBook);

    @Reindexer("books")
    public void moveBookToIndex(Book aBook, int index);

    @Finder(collection = "books", attribute = "title")
    public Book getBook(String title);

}
```

Executing PAMELA models




```
// Instantiate the meta-model
// by computing the closure of concepts graph
PamelaMetaModel pamelaMetaModel
    = PamelaMetaModelLibrary.retrieveMetaModel(Library.class);
// Instantiate the factory
PamelaModelFactory factory = new PamelaModelFactory(pamelaMetaModel);
// Instantiate a Library
Library myLibrary = factory.newInstance(Library.class);
myLibrary.setName("My library");
// Instantiate some Books
Book myFirstBook
    = factory.newInstance(Book.class, "Lord of the ring");
Book anOtherBook
    = factory.newInstance(Book.class, "Holy bible");
myLibrary.addToBooks(myFirstBook);
myLibrary.addToBooks(anOtherBook);
```

PAMELA highlights

- Composition of model interpretation + custom code
  - Modeling without code generation (less code, fewer bugs)
  - Strong coupling between model and code with smooth integration
- Meta programming
  - Code instrumentation, operation-based
  - Contract-based programming (JML)
  - Aspect-Oriented Programming
- Features
  - Multiple inheritance and traits programming, content management, undo/redo stack, clipboard operations, notifications, validation, graph-based computations, …

[1] Sylvain Guérin, Guillaume Polet, Caine Silva, Joel Champeau, Jean-Christophe Bach, Salvador Martínez, Fabien Dagnat, Antoine Beugnard, *PAMELA : an annotation based Java Modeling Framework*, in Science of Computer Programming, volume 210, 102668, 2021

[2] Caine Silva, Sylvain Guérin, Raùl Mazo, Joël Champeau, *Contract-based design patterns : a design by contract approach to specify security patterns*, Proceedings of the 15th International Conference on Availability, Reliability and Security, ARES '20, 2020

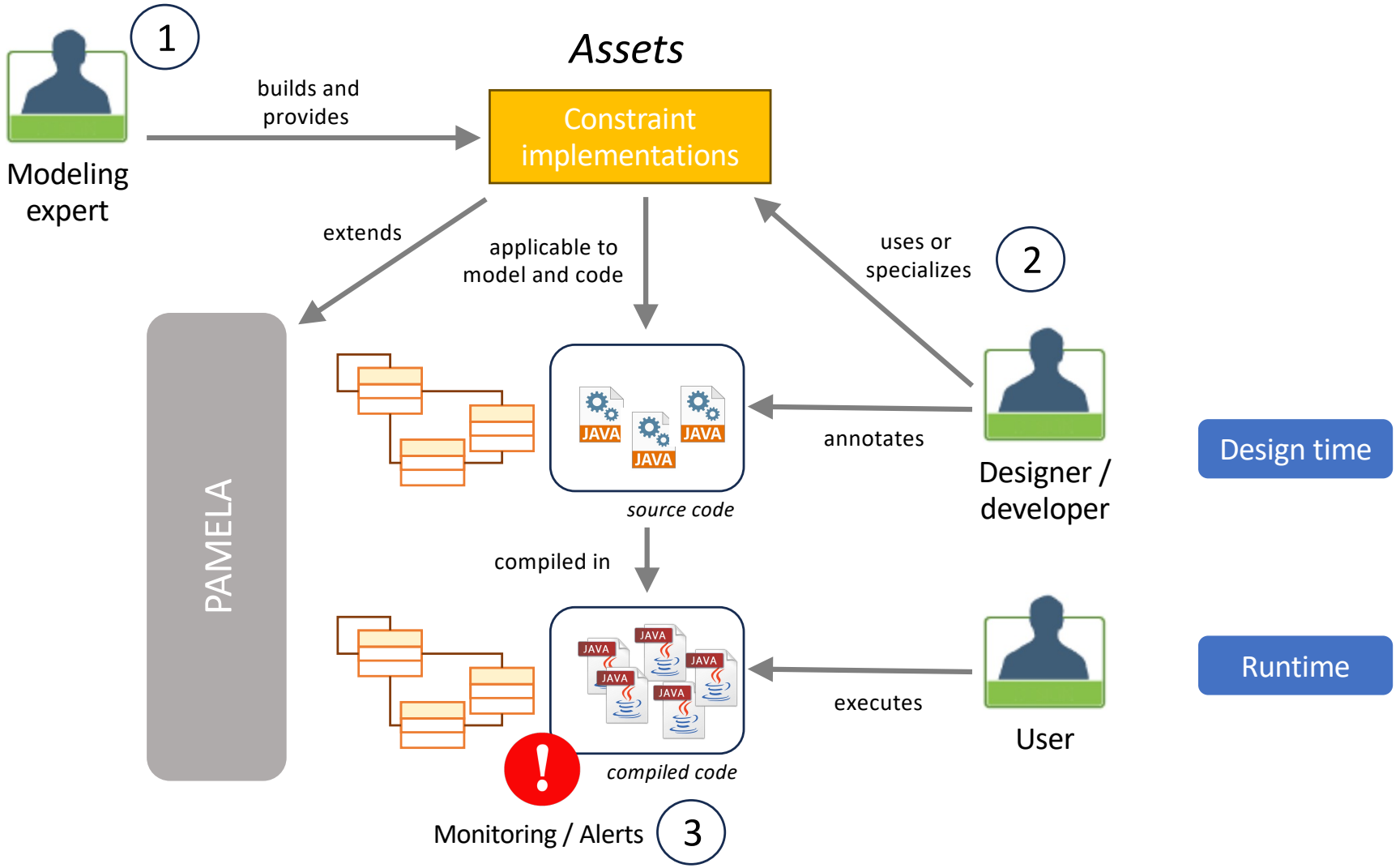
Agenda

1. Context and challenges
2. Model-Oriented Programming
3. Our approach
4. Examples
5. Conclusions and perspectives

- ✓ Need for advanced association constraints
- ✓ Interest of model-oriented programming

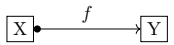
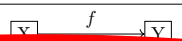
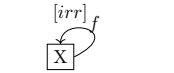
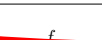
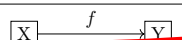
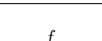
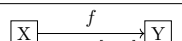
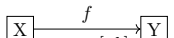
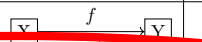
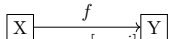
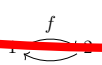
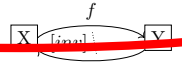
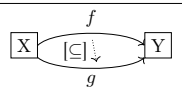
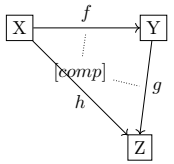
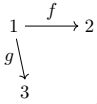
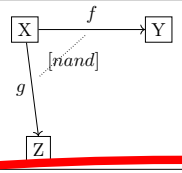
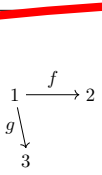
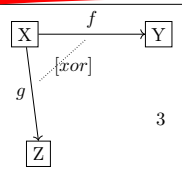
Our approach : reify constraints

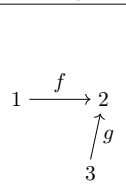
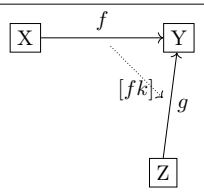
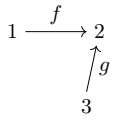
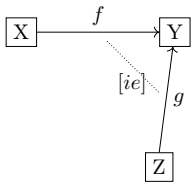
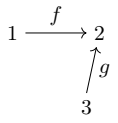
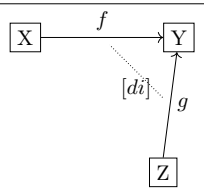
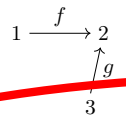
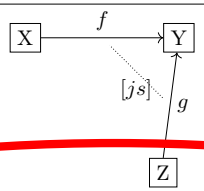
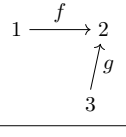
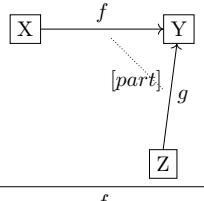
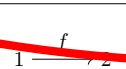
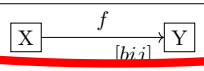
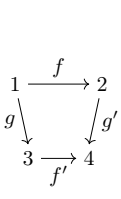
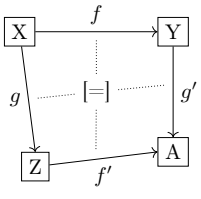
1. Implement constraints in PAMELA (*modeling experts)*
2. Place constraints in models and source code (*designers/developers)*
3. Monitor constraints at runtime

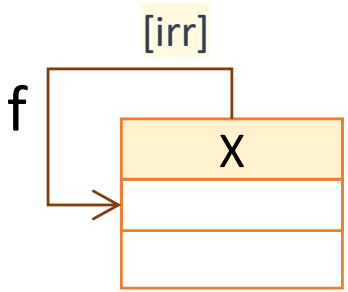

1
Assets
builds and provides
Modeling expert
Constraint implementations
extends
applicable to model and code
uses or specializes
2
PAMELA
JAVA
source code
annotates
Designer / developer
Design time
compiled in
compiled code
executes
User
Runtime
Monitoring / Alerts
3

Agenda

1. Context and challenges
2. Model-Oriented Programming
3. Our approach
4. Examples
5. Conclusions and perspectives

Name	Arity	Visualization	Semantic
total			$\forall x \in X, f(x)$ is defined (or $\lvert f(x)\rvert \geq 1$)
mult(m,n)			$\forall x \in X, m \leq \lvert f(x)\rvert \leq n$, with $0 \leq m \leq n$ and $n \geq 1$
irreflexive			$\forall x \in X, x \notin f(x)$
injective			$\forall x, x' \in X, f(x) = f(x') \Rightarrow x = x'$
key			$\forall x, x' \in X, x \neq x' \Rightarrow f(x) \neq f(x')$
primary-key			f is [total] and [injective]
non-overlapping			$\forall x, x' \in X, f(x) \cap f(x') \neq \emptyset \Rightarrow x = x'$
surjective			$f(X) = Y$
inverse			$\forall x \in X, \forall y \in Y : y = f(x) \Leftrightarrow x = g(y)$
image-inclusion			$\forall x \in X, f(x) \subseteq g(x)$
composition			$\forall x \in X, h(x) = \bigcup\{g(y) \mid y \in f(x)\}$
nand			$\forall x \in X, f(x) = \emptyset \vee g(x) = \emptyset$
xor			$\forall x \in X, (f(x) = \emptyset \vee g(x) = \emptyset) \wedge (f(x) \neq \emptyset \vee g(x) \neq \emptyset)$

Name	Arity	Visualization	Semantic
foreign-key			g is [primary-key] and $f(X) \subseteq g(Z)$
image-equal			$f(X) = g(Z)$
disjoint-image			$f(X) \cap g(Z) = \emptyset$
joinly-surjective			$f(X) \cup g(Z) = Y$
partition			$f(X) \cap g(Z) = \emptyset \wedge f(X) \cup g(Z) = Y$
bijective			f is [mult(1,1)], [injective] et [surjective]
commutative			$\forall x \in X, g'(f(x)) = f'(g(x))$



Irreflexive relation

$\forall x \in X, x \notin f(x)$

2

Design time

```
@ModelEntity
public interface X extends MonitorableProxyObject {

    @Getter("singleX")
    @Irreflexive
    X getSingleX();

    @Setter("singleX")
    public void setSingleX(X value);

}
```

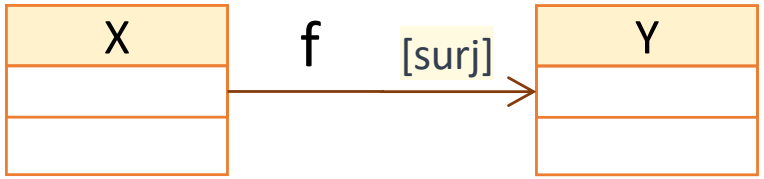
Use of @Irreflexive predicate

Alternative method using JML

```
@ModelEntity
@Invariant("singleX != object")
public interface X extends MonitorableProxyObject {

  // ...

}
```

2

Design time



Surjective relation

$$f(X) = Y$$

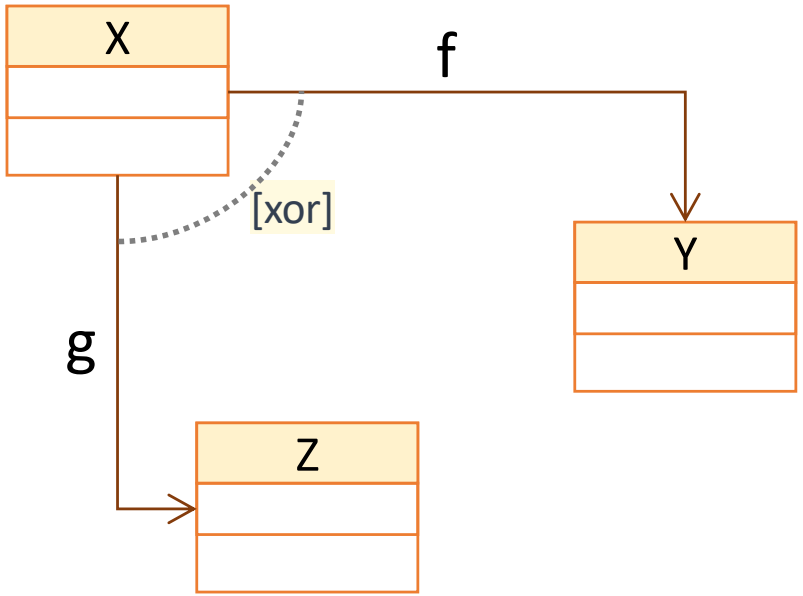
```
@ModelEntity
public interface X extends MonitorableProxyObject {

    @Getter("Y")
    @Surjective
    Y getY();

    @Setter("Y")
    public void setY(Y value);

}
```

2

Design time

XOR relation

$$\forall x \in X, (f(x) = \emptyset \vee g(x) = \emptyset) \wedge (f(x) \neq \emptyset \vee g(x) \neq \emptyset)$$



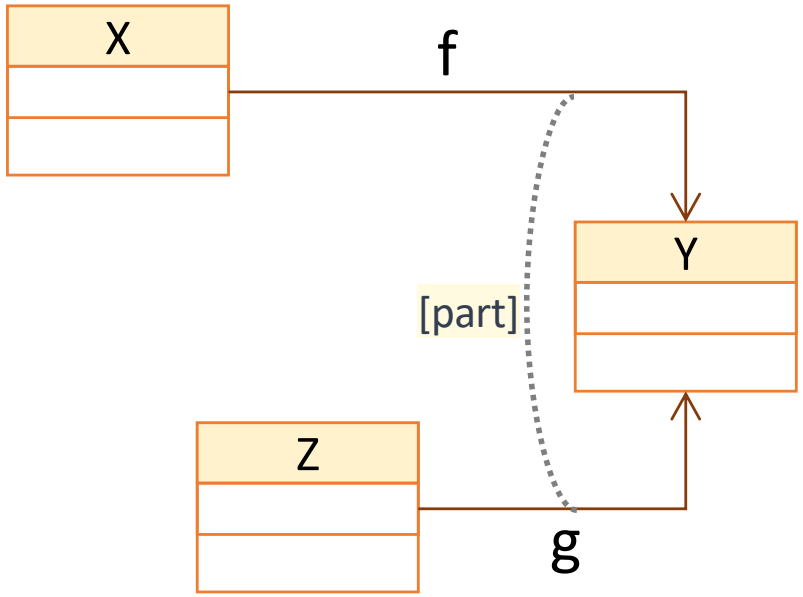

```java
@ModelEntity
public interface X extends MonitorableProxyObject {

    static final String XOR_ASSOCIATION = "YandZareExclusive";

    static final String Y_KEY = "y";
    static final String Z_KEY = "z";

    @Getter(Y_KEY)
    @XOrAssociation(patternID = XOR_ASSOCIATION)
    public Y getY();

    @Setter(Y_KEY)
    public void setY(Y y);

    @Getter(Z_KEY)
    @XOrAssociation(patternID = XOR_ASSOCIATION)
    public Z getZ();

    @Setter(Z_KEY)
    public void setZ(Z z);

}
```

Partition relation

2

Design time

$$f(X) \cap g(Z) = \emptyset \wedge f(X) \cup g(Z) = Y$$




```java
@ModelEntity
public interface Y extends MonitorableProxyObject {

    static final String PARTITION_X_Z = "XAndZArePartitionned";

    static final String Y_KEY = "y";
    static final String Z_KEY = "z";

    @Getter(Y_KEY)
    @Partition(patternID = PARTITION_X_Z)
    public X getY();

    @Setter(Y_KEY)
    public void setY(X y);

    @Getter(Z_KEY)
    @Partition(patternID = PARTITION_X_Z)
    public Z getZ();

    @Setter(Z_KEY)
    public void setZ(Z z);

}
```

```java
public class XOrAssociationDefinition<I> extends PatternDefinition {

    public static final String SOURCE_ROLE = "Source";

    public ModelEntity<I> sourceEntity;
    public ModelProperty<? super I> property1;
    public ModelProperty<? super I> property2;

    public XOrAssociationDefinition(String identifier, PamelaMetaModel pamelaMetaModel) {

    public Class<? extends XOrAssociationInstance> getInstanceClass() {

    public void finalizeDefinition() throws ModelDefinitionException {

    @Override
    public <I2> void notifiedNewInstance(I2 newInstance, ModelEntity<I2> modelEntity, PamelaModel model)
            throws IllegalAccessException, IllegalArgumentException, InvocationTargetException {
        if (modelEntity == sourceEntity) {
            // We create a new PatternInstance for each new instance of subjectModelEntity
            XOrAssociationInstance<I> newPatternInstance
                = new XOrAssociationInstance(this, model, newInstance);
        }
    }

    public boolean isMethodInvolvedInPattern(Method method) {

    public String toString() {

    public boolean isValid() {

}
```

XOrAssociationDefinition.java

```java
public class XOrAssociationFactory extends AbstractPatternFactory<XOrAssociationDefinition> {

    public XOrAssociationFactory(PamelaMetaModel pamelaMetaModel) {
        super(pamelaMetaModel);
    }

    protected void discoverMethod(Method m) throws ModelDefinitionException {

}
```

XOrAssociationFactory.java

```java
public class XOrAssociationInstance<I> extends PatternInstance<XOrAssociationDefinition<I>>
    implements PropertyChangeListener {

    private final I entity;
    private boolean isChecking = false;

    public XOrAssociationInstance(XOrAssociationDefinition<I> definition, PamelaModel model, I entity)

    public boolean isValid() {

    public I getEntity() {

    /** Called when a notification has been received */
    public void propertyChange(PropertyChangeEvent evt) {

    /**
     * Method called before every method of interest is about to be invoked. Performs the execution, if relevant
     */
    public ReturnWrapper processMethodBeforeInvoke(Object instance, Method method, Object[] args)

    public void processMethodAfterInvoke(Object instance, Method method, Object returnValue, Object[] args)

    /**
     * Method called before after all method invoke. It performs the invariant and postcondition checks.
     */
    void checkAfterInvoke(I instance, Method method, Object returnValue, Object[] args)

    private boolean checkXor(I instance) {

}
```

XOrAssociationInstance.java

1 Modeling expert's point of view

Configurable monitoring

3 Runtime

- Monitoring is configurable (and can be switched off)
- Follows a visible state semantics (constraints are/may be verified before and/or after relevant methods)
- Different strategies available
  - Explicitly monitored methods only
  - Interpreted and monitored methods
  - All public methods exclude some not monitored
  - All public methods
  - No monitoring
- No constraints proving, but a useful assistance in early development stages, and for debugging

Agenda

1. Context and challenges
2. Model-Oriented Programming
3. Our approach
4. Examples
5. Conclusions and perspectives

Conclusions

- Model-oriented programming leverages high-level association constraints
- Improve model expressivity with extension capabilities
- An existing « catalogue » of constraints, extensible
- Usages : early debugging, monitoring, « protected » runtime

Future work

- Implements more constraints
- Configurable constraints
- Constraints composition
- Integrated Development Environment
- Evaluate performance impact on complex constraints

Questions ?