Generating Integrity Preserving Associations,

The First step to Biome

Reinier van Oosten
VOOC

What is Gipa

- Gipa is a set of parameterized implementation level pattern classes.
- These can be used to generate methods and variables in domain model classes.
- The patterns describe a type of association and their composing roles.
- The current focus of Gipa is on structure associations like 1 to n relationship.
- The focus of Gipa is on integrity preservation.
- Gipa looks like MDA, but with a complete different focus.

Overview

- Integrity in domain models
- Homeostatic or self regulating approach
- Patterns of integrity preserving associations
- Specifying associations
- Code generation based on patterns and specification
- Biology inspired object-oriented modeling environment (Biome)

The problem:

- Objects may change their state and this change may influence the state of other objects.
- The state of objects may be in conflict with the state of other objects.
- This is the integrity problem
- This may be solved using with self regulating or homeostatic algorithms.
- Development of homeostatic algorithms is time consuming and requires extensive testing.

Integrity of object structures

- Classical ER structure
- Classical problem but not solved
- Dangling pointers-dangling objects

- Gipa solution: a set of rules that can be guarded by homeostatic methods.

Integrity in domain models

- 3 types of domain objects:
  - Data objects (Integer, Timestamp, Class etc)
  - Structure objects (Collections)
  - Model objects.

3 Rules

- Data objects should never be changed.
- Structure objects should be kept private.
- Navigation between model objects should always be bidirectional.

Advantage and disadvantage

- Advantages
  - The domain model is always stable.
  - There are a number of patterns supporting the rules.
- Disadvantages
  - Coding is a more complicated, and a very repetitive (boring) task.

Solution

- A framework for generating pattern based code.
  - A (domain) model specified in a package and a namespace
  - A collection of domain classes.
  - A collection of associations.
  - A collection of roles for each association, to be implemented by a set of classes.

Association patterns

- The pattern describes the association in a parameterized way.
- The names of methods and variables are stored in roles.
- Given the specification of the association in roles, methods and variables are generated.

Various patterns

- ToOne (= attribute access in the VW browser)
- ToMany
- OneToOne
- OneToMany
- ManyToMany
- DoubleLink
- Tree

- Any pattern anybody wishes to implement

Very simple example

```
patternToOneGet
   | stream |
   stream := CodingStream arguments: self  args.
   stream nextPutAll: ‘{one,simple:s}
        ^{one,var:s}’.
   ^stream selector ->(stream code}
```

Specification of an association

1 The association pattern

2 For each role specified by the association pattern, the variable- and method name bases.

3 XML format: GXD (Gipa XML Definition)

4 GXD comparable to class diagram definition in UML.

Example

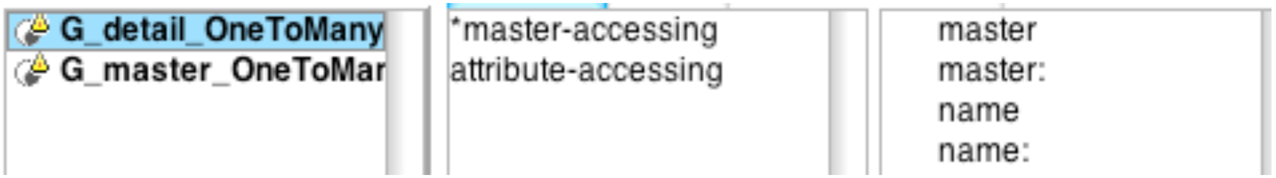
```
<GipaModel package="Gipa-generated oneToMany" namespace="GIPAExample”>

    <gipaClass className="G_master_OneToMany">
        <field name="name"/>
    </gipaClass>

    <gipaClass className="G_detail_OneToMany">
        <field name="name"/>
    </gipaClass>

    <gipaAssociation definition="OneToMany">
        <role role="one" variable="maten" single="maat" multiple="maten">
            <class className="G_master_OneToMany"/>
        </role>
        <role role="many" variable="master" keySelector="name" create="true">
            <class className="G_detail_OneToMany"/>
        </role>
    </gipaAssociation>
</GipaModel>
```

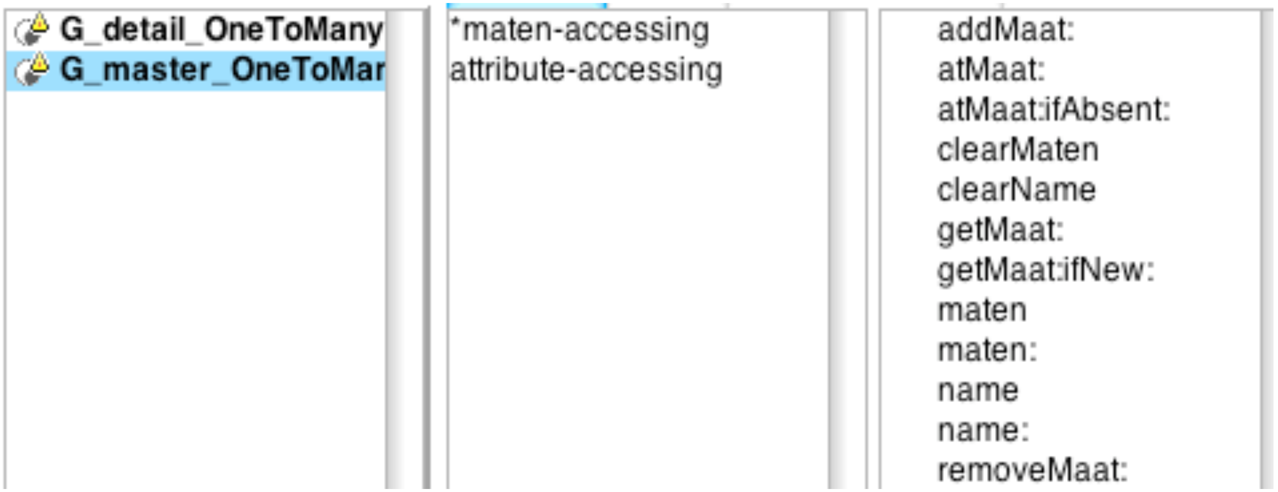
Generated classes and methods



G_detail_OneToMany
G_master_OneToMar

*master-accessing
attribute-accessing

master
master:
name
name:



G_detail_OneToMany
G_master_OneToMar

*maten-accessing
attribute-accessing

addMaat:
atMaat:
atMaat:ifAbsent:
clearMaten
clearName
getMaat:
getMaat:ifNew:
maten
maten:
name
name:
removeMaat:

Crucial method for “many”

```
master: newMaster
        newMaster = master ifTrue:[^newMaster].
        master ifNotNil: [:old |
            master := nil.
            old removeMaat: self].
        newMaster ifNotNil: [
            master := newMaster.
            newMaster addMaat: self]
```

Crucial method for “one”

```
addMaat: newMaat
    | result |
    maten ifNil: [maten := OrderedCollection new].
    result := maten detect: [:item | item name = newMaat name]
    ifNone:[
        maten add: newMaat.
        newMaat master: self.
        ^newMaat].
    result == newMaat ifTrue:[^newMaat]
        ifFalse:[
            ^GipaDuplicateKeyException raiseRequestWith:
                (Array with: result with: newMaat) ]
```

Result

- A class diagram can be specified in gxd using an xml editor like oxygen
- This generates Smalltalk code 10 times larger.
- This generated code is already tested.

Work to do

- Gipa is part of a larger project: Biome
- Biome: Biology Inspired Object-oriented Modelling and Engineering
- Structure of Biologic science can be used as a metaphore for agile object-oriented development and its environment

Mapping Biologic science to OO development

Fundaments	
Cell theory	Object theory
Gene theory	Class and Role Persistence

Research areas	Design areas
Physiology	Processing, Algorithms, Unit testing
Structure	Object structure, class diagram
Taxonomy	Inheritance
Ecology	Interaction, Associations

Mapping II

Force and Design	
Evolution	Versions

Forces	
Energy	Efficiency
Homeostasis	Not named, Often used in stable parts, Self regulating associations. Homeostasis

Focus of Biome

- Modelling (Associations and their patterns)
- Versioning (Evolution of models)
- Persistancy
- IDE

Focus of Biome (modelling)

- Modelling of associations and association patterns
- Implementing roles by classes
- Stimulating homeostasis to make associations and models self regulating

Focus of Biome (versioning)

- Current versioning VW = packages.
- A Gipa model is stored in package.
- Needed granularity: associations
- Needed version association:
  - pattern version-> association version
- Implementing an association version is a transaction

Focus of Biome (Persistancy)

- Associations and roles can be mapped to Gipa descriptions in GXD
- Supporting Glorp using GXD
- Supporting XML marshalling using GXD

Focus of Biome (IDE)

- Supporting Association-Role modelling
- Supporting Association-Role pattern modelling
- Supporting version implementation as transaction
- Supporting updating associations as a result of an association pattern upgrade or downgrade.