Budapest University of Technology and Economics
Department of Measurement and Information Systems
Fault Tolerant Systems Research Group

Model Driven Software Development course

IncQuery

Oszkár Semeráth Gábor Szárnyas

August 11, 2014

Contents

Chapter 1

IncQuery

Figure 1.1: The logo of EMF-IncQuery

1.1 Setup

1. Import the project from `start.zip`.
2. Generate the model, the edit and the editor from the `genmodel` file in the `model` folder of the `hu.bme.mit.mdsd.erdiagram` project.
3. Run as **Eclipse Application**.
4. Import the `ERDiagramExample` to the runtime Eclipse and check the instance model.
5. Create a new IncQuery project and name it to `hu.bme.mit.mdsd.erdiagram.queries`.
6. Create a new query definition in a package named `hu.bme.mit.mdsd.erdiagram` and a file named `queries.eiq`. In the wizard create an empty query. Fill the first query:

```
package hu.bme.mit.mdsd.erdiagram

import "hu.bme.mit.mdsd.erdiagram"

pattern entityWithName(entity, name) {
    Entity.Name(entity,name);
}
```

7. Load the query and the instance model to the **Query Explorer**.

1.2 Simple Query Language Tutorial

1. Structure your source code to 4 blocks like this:

```
//------------------------------
// Support
//------------------------------

//------------------------------
// Visualize
//------------------------------

//------------------------------
// Validate
//------------------------------

//------------------------------
// Derived
//------------------------------
```

   Every pattern goes to one of those categories. The `entityWithName` goes to **Support**.

2. Create a query to the **Validate** that checks if the name of a `NamedElement` is only an empty string:

```
pattern emptyNamedElement(element: NamedElement) {
    NamedElement.Name(element, "");
}
```

3. Create a query to the **Validate** that checks if two entity has the same name:

```
pattern sameNamedEntities(entity1, entity2, commonName) {
    Entity.Name(entity1, commonName);
    Entity.Name(entity2, commonName);
    entity1!=entity2;
}
```

4. Create a query to the **Validate** that checks if the name starts with a noncapital letter:

```
pattern entityStartsWithSmallCase(entity) {
    Entity.Name(entity,name);
    check (
        !name.matches("^[A-Z].+")
    );
}
```

5. Create a query to the **Derived** that gets the other endign of a relation ending:

```
pattern other(ending:RelationEnding, other) {
    Relation.leftEnding(relation, ending);
    Relation.rightEnding(relation, other);
} or {
    Relation.rightEnding(relation, ending);
    Relation.leftEnding(relation, other);
}
```

6. Create a query to the **Visualize** that summarizes this three validation condition:

```
pattern badEntity(entity, name) {
    find sameNamedEntities(entity, _other, name);
} or {
    Entity.Name(entity, name);
    find emptyNamedElement(entity);
} or {
    Entity.Name(entity, name);
    find entityStartsWithSmallCase(entity);
}
```

7. Create a query to the **Visualize** that matches to the well-named entities:

```
pattern goodEntity(entity, name) {
    Entity.Name(entity, name);
    neg find badEntity(entity,_);
}
```

8. Create a query to the **Visualize** that gets the attributes:

```
pattern attribute(entity, attribute) {
    Entity.attributes(entity,attribute);
}
```

9. Create a query to the **Visualize** that gets the attributes:

```
pattern attribute(entity, attribute) {
    Entity.attributes(entity,attribute);
}
```

10. Create a query to the **Visualize** that gets relations:

```
pattern relation(entity1, entity2) {
    Relation.leftEnding.target(relation, entity1);
    Relation.rightEnding.target(relation,entity2);
}
```

11. Create a query to the **Visualize** that matches on the attributes and check the properties:

```
pattern attributeWithName(attribute, name, type, key){
    Attribute.Name(attribute,name);
    Attribute.type.Name(attribute,type);
    Attribute.isKey(attribute,true);
    key=="[k]";
} or {
    Attribute.Name(attribute,name);
```

```
    Attribute.type.Name(attribute,type);
    Attribute.isKey(attribute,false);
    key=="";
}
```

1.3 Visualization tutorial

1. Use the visualize block to create a view. Annotate the patterns:

```
@Item(item = entity, label = "$name$")
pattern goodEntity(entity, name)

@Item(item = entity, label = "$name$")
@Format(color = "#ff0000")
pattern badEntity(entity, name)

@Item(item = attribute, label = "$key$$name$: $type$")
@Format(color = "#00ffff")
pattern attributeWithName(attribute, name, type, key)

@Edge(source = entity, target = attribute)
pattern attribute(entity, attribute)

@Edge(source = entity1, target = entity2)
pattern relation(entity1, entity2)
```

2. Watch the result in the **Viewers** sandbox.

1.4 Advanced Query language tutorial

1. For the sake of simplicity switch off the **Query Explorer** for the previous patterns with the following annotation:

```
@QueryExplorer(display = false)
```

2. Create **Support** patterns for the inheritance:

```
//@QueryExplorer(display = false)
pattern superEnitities(entity, superEntity) {
    Entity.isA(entity, superEntity);
}

//@QueryExplorer(display = false)
pattern allSuperEntities(entity, superEntity) {
    find superEnitities+(entity, superEntity);
}
```

3. Create a pattern that detects a circle in the type hierarchy:

```
pattern circleInTypeHierarchy(entity) {
    find allSuperEntities(entity, entity);
}
```

4. Create a pattern that detects a (transitive) diamond in the type type hierarchy:

```
pattern diamondInTypeHierarchy(entity1, entity2, entity3, entity4) {
    find allSuperEntities(entity1,entity2);
    find allSuperEntities(entity1,entity3);
    entity2 != entity3;
    find allSuperEntities(entity2,entity4);
    find allSuperEntities(entity3,entity4);
}
```

5. Every diamond has matched at least two times. This should be prevented if we make the pattern assimetric by defining somehow that `entity2 < entity3`. Let us define an ordering relation between the entities:

```
pattern order(a, b) {
    Entity.Name(a, name1);
    Entity.Name(b, name2);
    check(
        name1.compareTo(name2) < 0
    );
}
```

   And change the diamond code:

```
pattern diamondInTypeHierarchy(entity1, entity2, entity3, entity4) {
    find allSuperEntities(entity1,entity2);
    find allSuperEntities(entity1,entity3);
    //entity2 != entity3;
    find order(entity2, entity3);
    find allSuperEntities(entity2,entity4);
    find allSuperEntities(entity3,entity4);
}
```

6. By the way, calculate the infimum of the order:

```
pattern FirstInOrder(first: Entity) {
    neg find order(_, first);
}
```

7. Extend the patterns to get the inherited relations and attributes too:

```
pattern attribute(entity, attribute) {
    Entity.attributes(entity,attribute);
} or {
    find allSuperEntities(entity, superEntity);
    find attribute(superEntity, attribute);
}
```

   and

```
pattern relation(entity1, entity2) {
    Relation.leftEnding.target(relation, entity1);
    Relation.rightEnding.target(relation, entity2);
} or {
    find allSuperEntities(entity1, superEntity);
    find relation(superEntity, entity2);
}
```

8. Print out how many attributes a well-formed entity has:

```
@Item(item = entity, label = "$name$ ($attributes$)")
pattern goodEntity(entity, name, attributes) {
    Entity.Name(entity, name);
    neg find badEntity(entity,_);
    attributes == count find attribute(entity,_);
}
```

1.5 Validation

1. At first we need to import the query project to the host Eclipse. To do this copy the path of the project from the properties menu.
2. Close the project in the runtime Eclipse to avoid conflicts.
3. Go to the host Eclipse and import the query project.
4. Annotate some pattern with `@Constraint`, like:

```
@Constraint(message = "The name is not unique!", location=entity1, severity = "error")
pattern sameNamedEntities(entity1, entity2, commonName)

@Constraint(message = "The name is empty!", location=element, severity = "error")
pattern emptyNamedElement(element: NamedElement)
```

5. Start the runtime Eclipse, open the instance model and **right click** on the resource and choose **EMF-IncQuery validation | Initialize EMF-IncQuery validators on Editor**.
6. If you make a mistake an error will rise.

1.6 Derived feature

1. Create a new EReference named `otherEnding` in the `RelationEnding` to itself. Set the following properties:
   - Changeable = false
   - Derived = true
   - Transient = true
   - Volatile = false
2. Annote the pattern pattern `other`:

```
@QueryBasedFeature
pattern other(ending:RelationEnding, other)
```

3. Start the runtime Eclipse and try the feature in the instance model.