Chapter 8

IncQuery

8.1 Setup

8.1.1 Import and Install

- Download and import the initial projects.
- By now, the meta-model should be stable. Install it to the host eclipse to increase the productivity of the tool developement. Select the `socialnetwork`, `edit` and the `editor` | Export | Plug-in Developement | Deployable plug-ins and fragments | Finish.
- Click Ok to the Security Warning.
- Click Ok and restart the Eclipse when it asks to do it.
- The exported plug-ins should be closed to prevent confusions.

8.1.2 Modeling

- Switch to the Modelling perspective.
- In the `instance` project open the `my.socialnetwork` | `Social Network` | `Social Network Diagram`. A Sirius diagram opens which presents an example social network: Figure 8.1.
- Open the tree editor too.

8.2 Query Explorer

- Open the `queries` project and the src | hu.bme.mit.socialnetwork.queries | queries.eiq file. There is four category for the patterns.
- Write the following pattern to the util section:

```
private pattern person(Person,Name,Age,Gender) {
        Person.name(Person,Name);
        Person.age(Person,Age);
        Person.sex(Person,Gender);
}
```

- Open the query explorer (Window | Show View | Other. . . | EMF-IncQuery | Query Exporer).
- Open the query editor then the instance model editor, and push the play button on the Query exporter. It loads them, and match the queries on the instance models. The results are visible in the main area of the view. There should be 4 matches.

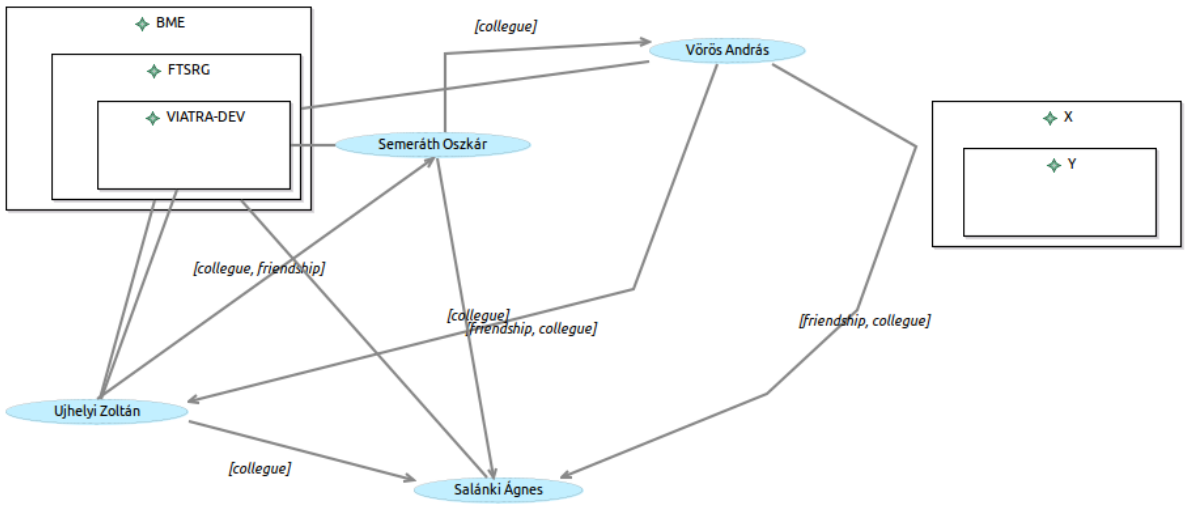

Figure 8.1: Example model in Sirius diagram

- You can filter the matches in the right side. Write 29 to the Age field in order to create a query which selects the persons with the age of 29.
- Change the model and the result instantly respond.

8.3 IncQuery Patterns

- Write a pattern to the constraint section which checks if two person has the same names (features: `==`, `!=`):

```
pattern personsWithSameName(P1,P2) {
	Person.name(P1,N1);
	Person.name(P2,N2);
	P1 != P2;
	N1 == N2;
}
```

- Write two pattens to the visualisation which collects only the males and the females with the name, and in case of men the age (features: `find`, enums, anonym variables):

```
pattern Men(Person, Name, Age) {
	find person(Person,Name,Age,Sex::male);
}
pattern Woman(Person, Name) {
	find person(Person,Name,_,Sex::female);
}
```

- Write a pattern to the constraint section which selects the persons where the name is undefined (features: `neg find`, `or`):

```
pattern personWithoutName(P) {
	Person.name(P,"");
} or {
	Person(P);
	neg find person(P,_,_,_);
}
```

- Create a pattern to trafo which collects all members of a community (feature: +):

```
private pattern subcommunities(C1:Community,C2:Community) {
       Community.subCommunity(C1,C2);
}
pattern allMember(C, M) {
       Community.members(C,M);
} or {
       find subcommunities+(C,S);
       Community.members(S,M);
}
```

- Write a pattern in the constraint section to select the communities which does not have member:

```
pattern communityWithoutMember(C) {
       Community(C);
       neg find allMember(C,_);
}
```

- Create a pattern which checks whether there is a relation between two person or not. Only one direction should be presented. The direction should be based on the alphabetic order of the names.

```
pattern relations(P1, P2) {
       Acquaintance.source(A,P1);
       Acquaintance.target(A,P2);
       Person.name(P1,N1);
       Person.name(P2,N2);
       check(N1<N2);
} or {
       Acquaintance.target(A,P1);
       Acquaintance.source(A,P2);
       Person.name(P1,N1);
       Person.name(P2,N2);
       check(N1<N2);
}
```

8.4 Zest Visualisation

- Write the following annotations to the patterns in the visualisation section:

```
@Item(item = Person, label = "$Name$ ($Age$)")
pattern Men(Person, Name, Age)

@Format(color = "#FF55AA")
@Item(item = Person, label = "$Name$")
pattern Woman(Person, Name)

@Edge(source = P1,target = P2)
pattern relations(P1, P2)
```

- Add a visual definition to the following pattern:

```
@Format(color = "#FF55AA")
@Item(item = Person, label = "$Name$")
pattern Woman(Person, Name)
```

- In the Query Explorer Right click on the model | Reinitialise IncQuery Viewers
- Inspect the different views. The Zest visualisation provides a graph-based diagram visible in Figure 8.2.

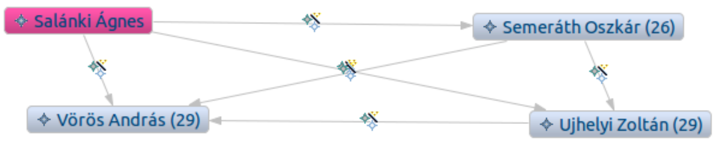

Figure 8.2: Zest Visualisation

8.5 Validation Constraints

- Put the following constraints to the patterns in the constraint section:

```
@Constraint(location = C, severity = "warning",
  message="Community $C.name$ has no member.")
pattern communityWithoutMember(C)

@Constraint(location = P, severity = "error", message="Unnamed person.")
pattern personWithoutName(P)

@Constraint(location = P1, severity = "error",
  message="Two people has the same name: $P1.name$")
```

- If you open an instance model in the runtime eclipse you can press Righ click | Initialise IncQuery Validators on Editor... to create error messages when a constraint pattern satisfied.
- If you create some error (like erase or duplicate a name) an error will appear in the Problem view.

8.6 Viatra Transformation

- Check the generated code.
- Open the AddTransitiveCommunities.xtend file, and add the folowing rule:

```
// Rules
val addTransitiveCommunities = createRule.
  precondition(AllMemberMatcher::querySpecification).
  action[c.members += m].
  build
```

- Write the following body to the `activate()` method:

```
def public activate() {
  addTransitiveCommunities.fireAllCurrent
}
```

- A command handler of a drop-down menu will call this method.
- Start a runtime eclipse, and a import the instance project.
- Right click on the instance model and select the Fill Transitive option. If you open the model, each transitive community membership will be added.