DepIn-O: Code Smell Application

[1]Appendix to "DepIn-O: an Ontology on Dependency Injection Software Frameworks"

This appendix presents a basic practical application of DepIn-O, capturing *Code Smells*. In order to do that, we extended the operational version of OOC-O written in OWL by adding DepIn-O concepts, employing the ontology editor *Protégé*.

We focused on two particular DI Code Smells:

- *Concrete Class Injection* — a direct request to a Concrete Dependency by a Consumer Class, as it causes the loss of the flexibility brought by using abstractions.
- *Over-Injection* — any combination of Constructor Injectors with Property Injectors resulting in over four dependencies requests at initialization, a sign of a possible violation of the Single Responsibility Principle.

To exemplify how we captured these code smells with this application, we:

- Created the following object properties:
  - **hasInjector**
  - **injects**
- Created the following distinct individuals:
  - Instances of OOC-O Concrete Classes: **ClassA**, **ClassB**, **ClassC**.
  - Instances of OOC-O Abstract Classes: **ClassD**, **ClassE**, **ClassF**, **ClassG**, **ClassH** and **ClassI**.
  - Instances of OOC-O Constructor Method: **ClassAConstructor**, **ClassBConstructor** and **ClassCConstructor**.
  - Instances of OOC-O Instance Variable: **varClassG**, **varClassH** and **varClassI**.
  - Instances of OOC-O Instance Method: **injectorClassH**.
- Asserted the following properties to the individuals:
  - **ClassA** *hasInjector* **ClassAConstructor**.
  - **ClassA** *hasInjector* **varClassG**.
  - **ClassA** *hasInjector* **varClassH**.
  - **ClassAConstructor** *injects* **ClassD**.
  - **ClassAConstructor** *injects* **ClassE**.
  - **ClassAConstructor** *injects* **ClassF**.
  - **varClassG** *injects* **ClassG**.
  - **varClassH** *injects* **ClassH**.
  - **ClassB** *hasInjector* **ClassBConstructor**.
  - **ClassB** *hasInjector* **varClassI**.
  - **ClassBConstructor** *injects* **ClassC**.
  - **varClassI** *injects* **ClassI**.
  - **ClassC** *hasInjector* **ClassCConstructor**.
  - **ClassC** *hasInjector* **injectorClassH**.
  - **ClassCConstructor** *injects* **ClassD**.
  - **ClassCConstructor** *injects* **ClassE**.

– **ClassCConstructor** *injects* **ClassF**.
– **ClassCConstructor** *injects* **ClassG**.
– **injectorClassH** *injects* **ClassH**.

From that, we can conclude:

- **ClassA** is a consumer of more than four dependencies through either Constructor Injector or Property Injector, thus an example of *Over-Injection*.
- **ClassB** possesses a Constructor Injector that *injects* an OOC-O Concrete Class, thus an example of *Concrete Class Injection*.
- **ClassC** is a consumer of five dependencies, though it does **not** fit any of our code smells criteria, since the fifth dependency is injected by a Method Injector (non-Constructor Instance Method), and none of its requested dependencies are OOC-O Concrete Classes.

Using *FaCT++* as our reasoner, we present the DL queries employed to capture our code smells in Listing 1 and 2. Figures 1 2 show that the results matched our conclusions.

Listing 1. Query to capture *Concrete Class Injection*

```
(hasInjector some (Instance_Method and injects some Concrete_Class))
or
(hasInjector some (Instance_Variable and injects some Concrete_Class))
```

Even though DepIn-O establishes a distinction between a Constructor Injector (an Injector that is an OOC-O Constructor Method) and a Method Injector (an Injector that is a non-Constructor OOC-O Instance Method), Constructor Methods are still subclasses of Instance Methods, hence the query used successfully captures **ClassB** as its Constructor Method causes a *Concrete Class Injection*.

Listing 2. Query to capture *Over-Injection*

```
(hasInjector some (Constructor_Method and injects min 5 Class))
or
(hasInjector some (Constructor_Method and injects min 4 Class) and
hasInjector min 1 (Instance_Variable and injects some Class))
or
(hasInjector some (Constructor_Method and injects min 3 Class) and
hasInjector min 2 (Instance_Variable and injects some Class))
or
(hasInjector some (Constructor_Method and injects min 2 Class) and
hasInjector min 3 (Instance_Variable and injects some Class))
or
(hasInjector some (Constructor_Method and injects min 1 Class) and
hasInjector min 4 (Instance_Variable and injects some Class))
or
(hasInjector min 5 (Instance_Variable and injects some Class))
```

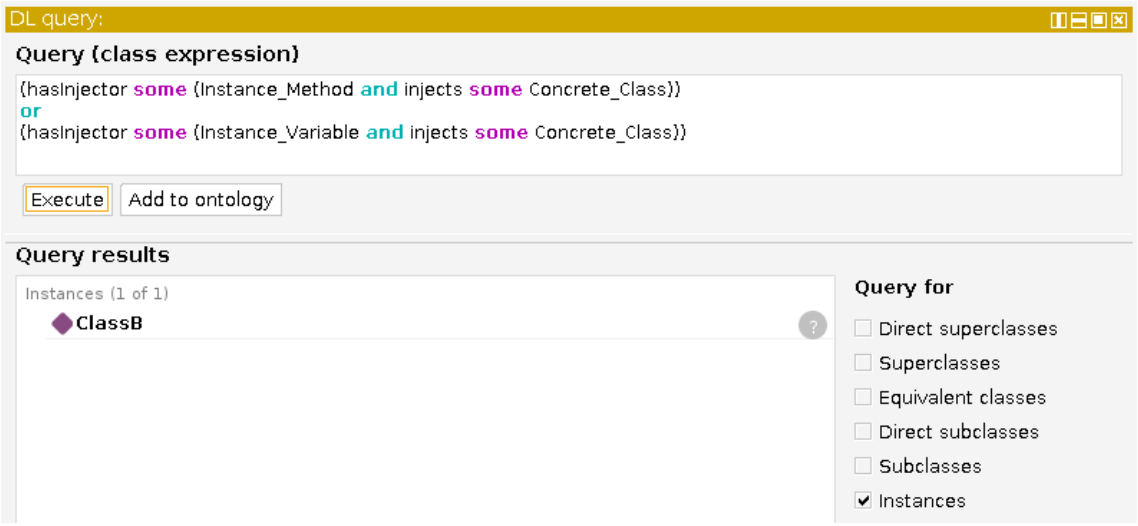

Figure 1. ***Concrete Class Injection*** **capture**

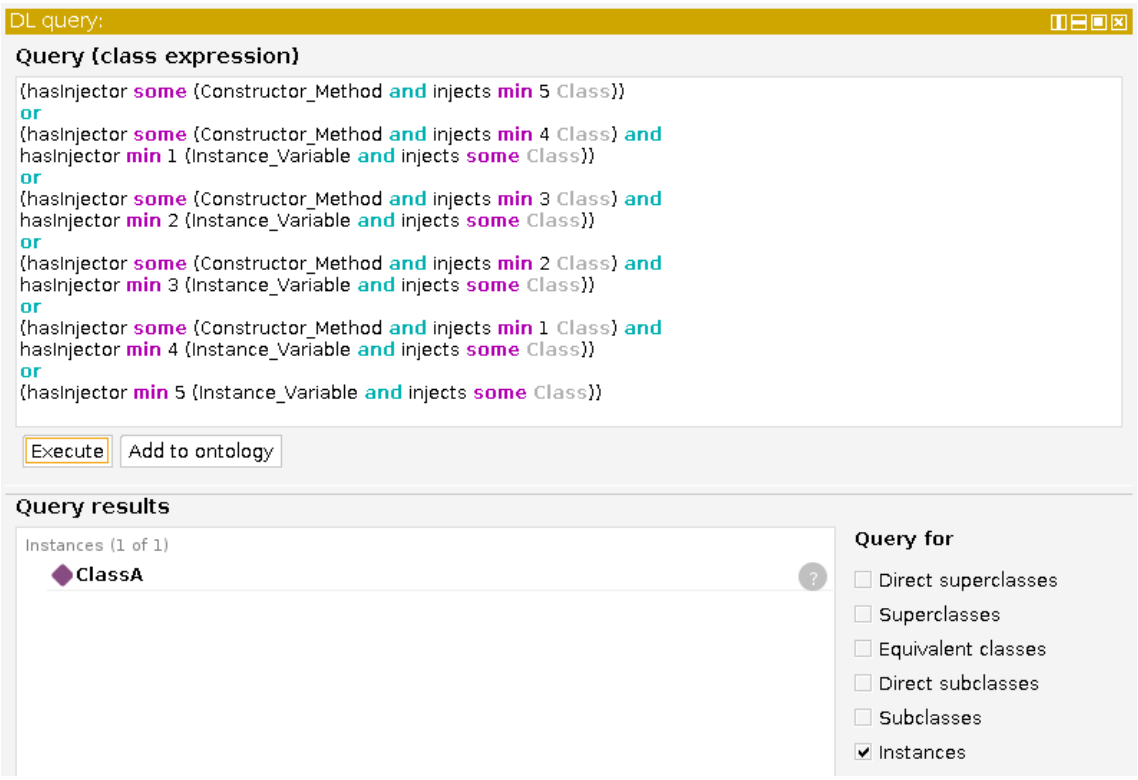

Figure 2. ***Over-Injection*** **capture**