Service Integration: Introduction to Code Generation

Gefördert durch:

Bundesministerium
für Wirtschaft
und Klimaschutz

IIP-Ecosphere Platform

Table of Contents

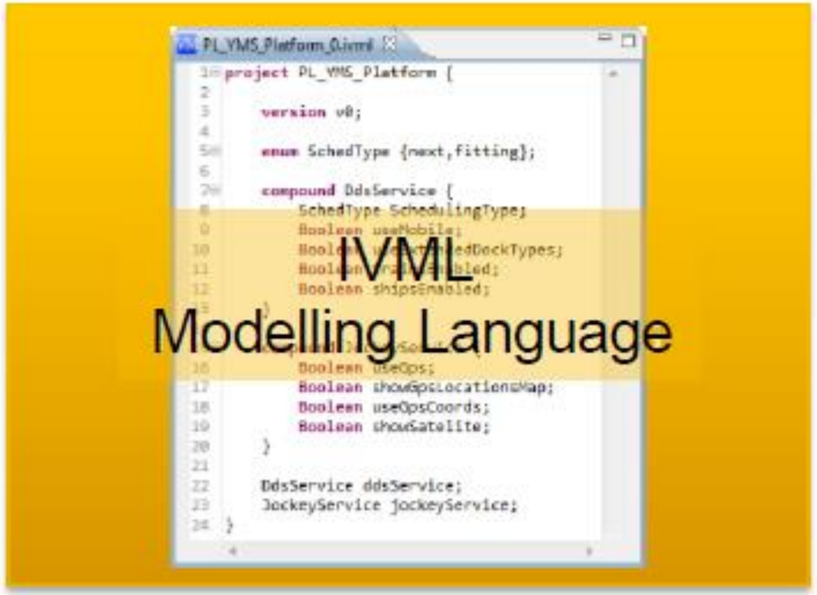
Prerequisites

- Required:
  - None
- Optional:
  - None

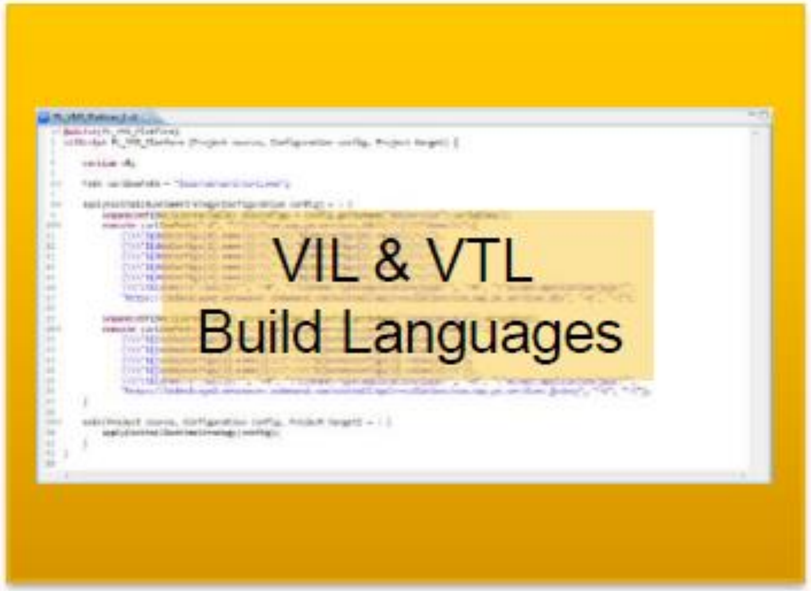
Table of Contents

Code generation (1)

- Multiple levels
  - Describing the artifact to be build (IVML)
  - Transforming the description into an executable artifact (VTL/VIL)
- As user you will work mostly with IVML (in the future also reduced due to UI)

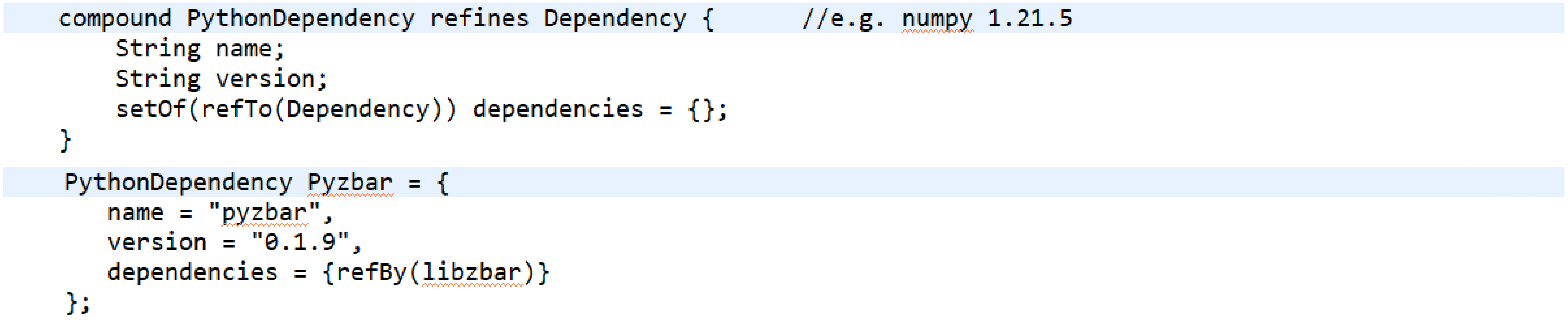


```
project PL_YMS_Platform {

    version v0;

    enum SchedType {next,fitting};

    compound DdsService {
        SchedType SchedulingType;
        Boolean useMobile;
        Boolean [illegible]edDockTypes;
        Boolean [illegible]bled;
        Boolean shipsEnabled;
    }
[illegible]
        Boolean useOps;
        Boolean showGpsLocationsMap;
        Boolean useOpsCoords;
        Boolean showSatellite;
    }

    DdsService ddsService;
    JockeyService jockeyService;
}
```



Code generation (2)

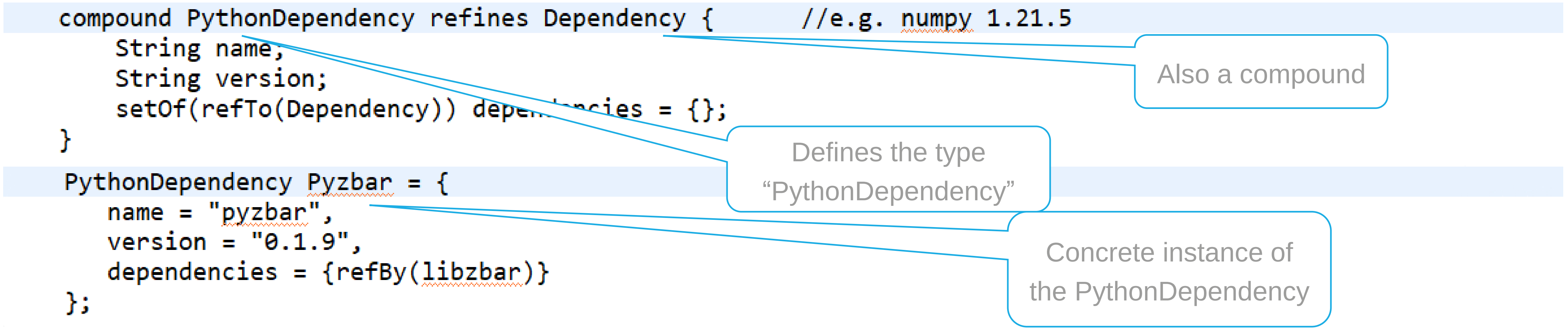
- IVML (Integrated Variability Modelling Language)
    - Language to describe variability and attributes of code artifacts
    - Top level of a IVML file is the project (in our case a project describes the application)
    - Supports basic types like Boolean, Integer, Real, String and the composition of new types
    - Supports composition of types though compounds



```
compound PythonDependency refines Dependency {      //e.g. numpy 1.21.5
    String name;
    String version;
    setOf(refTo(Dependency)) dependencies = {};
}

PythonDependency Pyzbar = {
   name = "pyzbar",
   version = "0.1.9",
   dependencies = {refBy(libzbar)}
};
```

Code generation (2)

- IVML (Integrated Variability Modelling Language)
  - Language to describe variability and attributes of code artifacts
  - Top level of a IVML file is the project (in our case a project describes the application)
  - Supports basic types like Boolean, Integer, Real, String and the composition of new types
  - Supports composition of types though compounds



```
compound PythonDependency refines Dependency {      //e.g. numpy 1.21.5
    String name;
    String version;
    setOf(refTo(Dependency)) dependencies = {};
}
PythonDependency Pyzbar = {
   name = "pyzbar",
   version = "0.1.9",
   dependencies = {refBy(libzbar)}
};
```

Also a compound
Defines the type "PythonDependency"
Concrete instance of the PythonDependency

Code generation (3)

- VIL (Variability Instantiation Language/ .vil)
  - Defines the instantiation process of a given product
  - Defines how the process of instantiation would work
- VTL (VIL Template Language/ .vtl)
  - Focuses on specific instantiation of text artifacts
  - Executes the instantiation process for a specific configuration
  - Produces the finished generated .py/.java files
- These languages use the .ivml files as configuration for the instantiation
- You should not need to work with these files

Summary

- What we learned
  - What each file type is responsible for
  - The rough structure of defining and instantiating types in IVML
- How to go on
  - How to edit datatypes