Service Integration: How to Configure Services
Configure Services

Configure Datatypes
- Determine needed types, describe them
- Configure basic service attributes
- Configure logging

Configure Services
- Define the application
- Define the service mesh
- Create the templates

Configure Application
- Implement the service functionality
- Install the services
- Build the application

Build the Application
- Create (unit) tests
- Use generated tests

Test the Services
- Add application
- Start up a platform
- Run the application

Deploy the Application
Table of Contents

• Prerequisites
• Configure the Services
Prerequisites

• Required:
  • Installed the platform and its dependencies or the development container
  • Installed the IDE for IIP-Ecosphere Platform (provided Eclipse Version)
  • How to configure datatypes

• Optional:
  • Introduction to code generation
Table of Contents

- Prerequisites
- Configuring the Services
What are Services

- Example of an application made up from services
- A selection of services chained together

Simple application structure:
- Source Service: Receive data and pass it on
- Transformer (possible Python): Transform/evaluate data
- Sink Service: Final evaluation: trigger action based on result
Configuring Services (1)

- The .ivml Files to configure the services and the application can be found in “src/test/easy/….ivml”

- Name and attributes of the application
- Define the service mesh
- Define the separate services
- Define the needed datatypes
- Define the technical aspects
• Define the services in the `AllServicesPart....ivml` file:

```java
project AllServicesPartPython {
    import AllTypes;

    annotate BindingTime bindingTime = BindingTime::compile to .;

    Service source = JavaService {
        id = "Source",
        name = "Source",
        description = "",
        ver = "0.1.0",
        deployable = true,
        asynchronous = false,
        traceRcv = TraceKind::SYSOUT,
        traceSent = TraceKind::SYSOUT,
        class = "de.iip_ecosphere.platform.impl.shop.TestSource",
        artifact = "de.iip-ecosphere.platform.apps:TestTestAppServices:" + iipVer,
        kind = ServiceKind::SOURCE_SERVICE,
        output = {{type=refBy[InData]}}
    };
```

• **Servicetype**: Python or JavaService need to be distinguished as the setup is different
• **id**: The id through which a service can be accessed in the UI/CLI
• **class**: (ONLY JAVA) The class where the code for the service can be found. Will also be used to determine package on template generation
Configure Services (3)

- Define the services in the `AllServicesPart....ivml` file:

```java
project AllServicesPartPython {

import AllTypes;

annotate BindingTime bindingTime = BindingTime::compile to .;

Service source = JavaService {
    id = "Source",
    name = "Source",
    description = "",
    ver = "0.1.0",
    deployable = true,
    asynchronous = false,
    traceRcv = TraceKind::SYSOUT,
    traceSent = TraceKind::SYSOUT,
    class = "de.iip_ecosphere.platform.impl.shop.TestSource",

    artifact = "de.iip_ecosphere.platform.apps:TestTestAppServices:" + iipVer,
    kind = ServiceKind::SOURCE_SERVICE,
    output = {{type=refByInData}}
};
```

- `artifact`: (The same for all Services) The Maven artifact where the class can be found `<mavenGroupId>:<artifactId>`, the values for these can be read from the `.pom` of the templates after their creation
- `kind`: The type of service: SOURCE_SERVICE, SINK_SERVICE or TRANSFORMATION_SERVICE (there are more)
Configure Services (4)

• Define the services in the `AllServicesPart....ivml` file:
  • You might need to look at the name of the defined datatypes as they are needed to define in- and out-put of services

```java
import AllTypes;

annotate BindingTime bindingTime = BindingTime::compile to ;

Service source = JavaService {
    id = "Source",
    name = "Source",
    description = "",
    ver = "0.1.0",
    deployable = true,
    asynchronous = false,
    traceRcv = TraceKind::SYSOUT,
    traceSent = TraceKind::SYSOUT,
    class = "de.iip_ecosphere.platform.impl.shop.TestSource",
    artifact = "de.iip_ecosphere.platform.apps:TestTestAppServices:" + iipVer,
    kind = ServiceKind::SOURCE_SERVICE,
    output = {{type=refByInData}}
};
```

• `output`: The datatype we intend to pass on from this service. Usually self defined in the `AllTypesPart...ivml`
• `input`: The datatype we want this service to receive. Usually self defined in the `AllTypesPart...ivml`
• `Note`: Sources only support `output` and Sinks only support `input`
Configure Services (5)

- Define the services in the `AllServicesPart....ivml` file:

```java
project AllServicesPartPython {
import AllTypes;

annotate BindingTime bindingTime = BindingTime::compile to .;

Service source = JavaService {
    id = "Source",
    name = "Source",
    description = "",
    ver = "0.1.0",
    deployable = true,
    asynchronous = false,
    traceRcv = TraceKind::SYSOUT,
    traceSent = TraceKind::SYSOUT,
    class = "de.iip_ecosphere.platform.impl.shop.TestSource",
    artifact = "de.iip-ecosphere.platform.apps:TestTestAppServices:" + iilVer,
    kind = ServiceKind::SOURCE_SERVICE,
    output = {{type=refBy[InData]}}
};
```

- **asynchronous**: defines whether or not this service acts asynchronous or synchronous
- **traceRcv**: Traces data reception by service.
  - TraceKind::NONE: No tracing
  - TraceKind::LOG: Using the logging framework
  - TraceKind::TRACE: Using AAS tracing
  - TraceKind::SYSOUT: Writing to the console
- **traceSent**: Trace data upon sending by service.
Configure Services (6)

- Define the services in the AllServicesPart....ivml file:
  - Example of a service in this case a python service
  - For further information on python services view the video “How to Create a Python Application”

```java
Service pyth = PythonService {
    id = "PyService",
    name = "PyService",
    description = "",
    ver = "0.1.0",
    deployable = true,
    asynchronous = true,
    traceRcv = TraceKind::SYSOUT,
    traceSent = TraceKind::SYSOUT,
    input = {{type=refBy(InData)}},
    output = {{type=refBy(OutData)}},
    artifact = "de.iip-ecosphere.platform.apps:TestTestAppServices:" + iipVer,
    kind = ServiceKind::TRANSFORMATION_SERVICE,
    dependencies = {refBy(PYTHON39)}
};
```

- **kind:** Python services are always TRANSFORMATION_SERVICES as this is currently the only supported version
Summary

• What we learned
  • How to configure services
• How to go on
  • How to configure our application
  • How to build an application
  • How to test an application