Service Integration: How to Configure an Application
Configure the Application

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

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

Configure Application
- Configure logging
- Implement the service functionality
- Install the services
- Build the templates

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

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

Deploy the Application
Table of Contents

- Prerequisites
- Configure the Application
- Configure the Service Mesh
- Generate Templates
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
  • How to configure services

• Optional:
  • Introduction to code generation
<table>
<thead>
<tr>
<th>Table of Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Prerequisites</td>
</tr>
<tr>
<td>• <strong>Configure the Application</strong></td>
</tr>
<tr>
<td>• Configure the Service Mesh</td>
</tr>
<tr>
<td>• Generate Templates</td>
</tr>
</tbody>
</table>
A service mesh defines the connections of services

- It is a directed data flow graph build from connectors and services
- The graph roots at one or multiple source services
- The graph terminates in sink services
- An application can consist of multiple service meshes
• Define the application in the `AllApplicationPart....ivml` file:

```java
project ApplicationPartPythonApp {

    import ServiceMeshPartPythonApp*

    @BindingTime bindingTime = BindingTime::compile to .;

    Application myApp = {
        id = "TestIntegrationApp",
        name = "TestTestApp",
        ver = "0.1.0",
        description = "",
        services = {refBy(myMesh)}
    };
```

• **Application**: Defines an application consisting of service meshes and services. Services shall be re-usable.
  - The **id** of the application for management through UI/CLI
  - **services**: defines the service mesh associated with this task/company
Prerequisites
Configure the application
Configure the service mesh
Generate templates
• Define the service mesh in the `ServiceMeshPart….ivml` file

```java
ServiceMesh myMesh = {
    description = "WorkshopApp",
    sources = {[refBy(mySource)} //defines all source services
};

MeshSource mySource = {
    pollInterval = 800, //if source service is asynchronous = false!
    impl = refBy(source), //name of a concrete service defined in AllServicesPart...
    next = {refBy(myConnMySourceMyTransformer)}
};

MeshConnector myConnMySourceMyTransformer = {
    name = "Source->Transformer",
    next = refBy(myTransformer)
};

MeshProcessor myTransformer = {
    impl = refBy(pyth), //defines the class containing his impl.
    next = {refBy(myConnMyTransformerMyReceiver)}
};

MeshConnector myConnMyTransformerMyReceiver = {
    name = "Transformer->Receiver",
    next = refBy(myReceiver)
};

MeshSink myReceiver = {
    impl = refBy(sink)
};
```

• Chains together the separately defined services
• Alternation between Service and Connector
• Each service kind has a corresponding mesh class like `MeshSource`, `MeshSink` and `MeshProcessor`
• Add `<pollInterval> to source in case of synchronous service`
• Define the service mesh in the `ServiceMeshPart....ivml` file

```
ServiceMesh myMesh = {
    description = "WorkshopApp",
    sources = {refBy(mySource)} //defines all source services
};

MeshSource mySource = {
    pollInterval = 1000, //if source service is asynchronous = false!
    impl = refBy(source), //name of a concrete service defined in AllServicesPart...
    next = {refBy(myConnMySourceMyTransformer)}
};

MeshConnector myConnMySourceMyTransformer = {
    name = "Source->Transformer",
    next = refBy(myTransformer)
};

MeshProcessor myTransformer = {
    impl = refBy(transformer), //defines the class containing his impl.
    next = {refBy(myConnMyTransformerMyReceiver)}
};

MeshConnector myConnMyTransformerMyReceiver = {
    name = "Transformer->Receiver",
    next = refBy(myReceiver)
};

MeshSink myReceiver = {
    impl = refBy(sink)
};
```

- **impl**: refers to the name of a defined service
- **next**: defines the next service along the data flow (through a connector)
- **name**: here as “<service>-><service>” which is only for documentation purposes and does not affect the generation.
Table of Contents

- Prerequisites
- Configure the Application
- Configure the Service Mesh
- Generate Templates
Generate Templates

- The templates for the `.ivml` files should enable you to now build an application.
- In `impl.model` run `mvn generate-sources`
  - This will get the platform model as well as generate the templates.
  - They will provide you with the groundwork to build an app.

- Rename the “impl.model” as needed/desired.
- Use `cmd` to run “mvn -U generate-sources” in the “impl.model” directory.
Summary

• What we learned
  • How to configure the service mesh
  • How to configure the application
  • How to generate templates
• How to go on
  • How to test out application
  • How to build our application