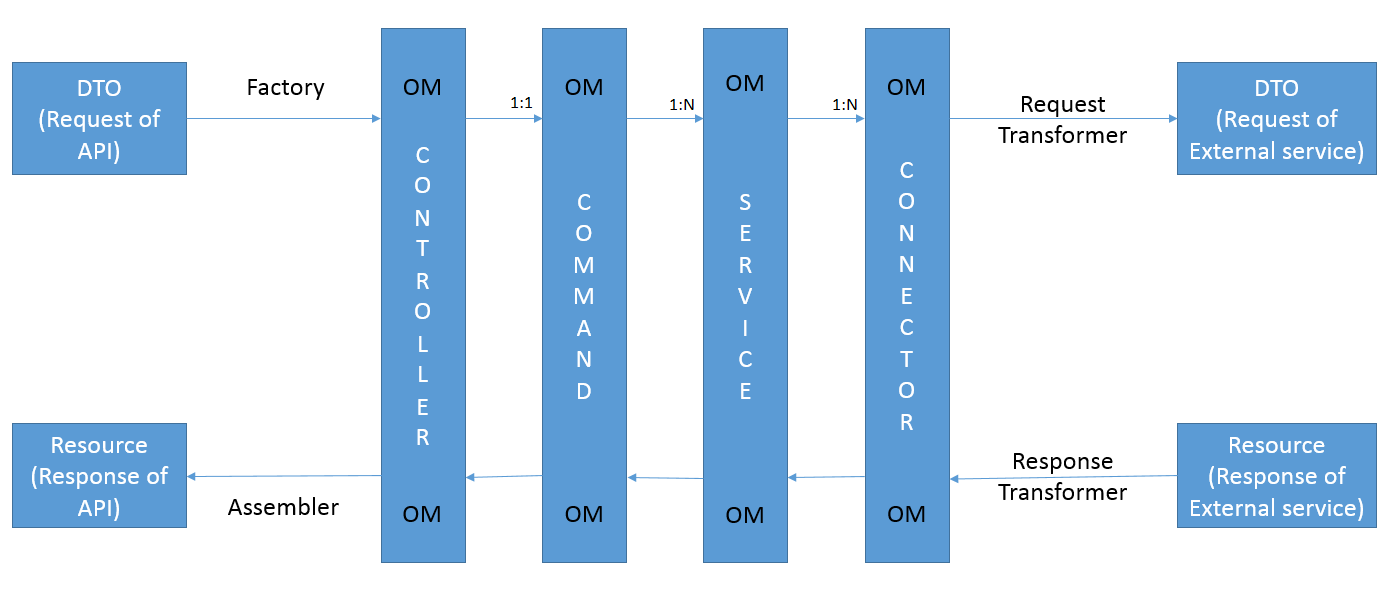
* Creato da [Midena, Andrea](https://confluence.gft.com/display/~A04K), ultima modifica di [Costa Sifuentes, Gabriel](https://confluence.gft.com/display/~glss) il [ago 09, 2019](https://confluence.gft.com/pages/diffpagesbyversion.action?pageId=192854977&selectedPageVersions=11&selectedPageVersions=12)

[Vai all’inizio dei metadata](https://confluence.gft.com/display/ISPP/How+to+develop+with+BEAR#page-metadata-start)

The BEAR is a framework of the office DSI of Intesa San Paolo bank.

**ARCHITECTURE of BEAR**



* **Controller**: This layer is the entry point of the API.
* **Command**: This layer contains the business logic of the API.
* **Service**: This layer contains the implementations of the interfaces with the external services.
* **Connector**: This layer contains the implementations to call the external services.
* **DTO**: Object that represents the request of the API or of the external service.
* **Resource**: Object that represents the response of the API or of the external service.
* **OM (object model)**: Object model to pass the data from one layer to another. In the case that the API uses only connections at the DB, this can be the direct representation of the DB table.
* **Factory**: Used to transform the DTO of the API into an OM.
* **Request** **Transformer**: Used to transform the OM into DTO of the external service.
* **Response** **Transformer**: Used to transform the Resource of the external service into an OM.
* **Assembler**: Used to transform the OM into the Resource of the API.

The BEAR microservices are composed of maven multi modules. In all microservices there are these modules:

* lib-model-<name of microservice>: contains all Entity objects and the object model
* ms-<name of microservice>: contains all the classes
  + controller
  + command
  + service
  + connector
  + transformer
  + assembler
  + factory
* one module for each external service to call in the microservice and one module to manage the drools rules. the prefix of this module are:
  + **lib-ms**: if the module contains the objects to call the APIs in other microservices.
  + **lib-rest**: if the module contains the objects to call the external services of type rest.
  + **lib-rest-bridge**: if the module contains the objects to call host services.
  + **lib-ws**: if the module contains the objects to call soap services.
  + **lib-kie**: module where there are the drools rules. To create the drools rules, follow this page: [How to create Drools Rules](https://confluence.gft.com/display/ISPP/How+to+create+Drools+Rules).

**PACKAGING**

The **packaging** to follow with BEAR is:

* for module **lib-model-<name of microservice>**:
  + com.intesasanpaolo.bear.lmbe0.**<name of microservice>**.
    - **model**: contains all Entity objects and these other packages
      * **domain**: contains all enums.
      * **converter**: contains all converters for enums.
      * **bin**: contains all object model.
* for module **ms-<name of microservice>**:
  + com.intesasanpaolo.bear.lmbe0.**<name of microservice>**.
    - **assembler**: contains the class to transform the OM into a Resource of the API. **All classes** contained within it **must extend** the class **BaseResourceAssemblerSupport.**
    - **command**: contains the command class. Usually there is one command for each API. **All classes** contained within it **must extend**the class **BaseCommand.**
    - **config**: contains the configuration classes.
    - **connector**: contains the connectors to call the external services. Usually there is one connector for each service. Check [How to create a Connector class](https://confluence.gft.com/display/ISPP/How+to+create+a+Connector+class).
    - **controller**: contains the RestController, one controller for each business functions (example: in the microservice core-fido-garanzia there is one controller for Fido, one for Garanzia, etc...).
    - **dto**: contains the objects for the DTOs of the APIs. **The main object of the DTO must extend BaseRequestDto** and **implement** the interface **Serializable.**
    - **exception**: contains the classes that represent the custom exception (usually there is one exception class for each microservice).
    - **factory**: contains the classes to transform the DTOs of the APIs into OM.
    - **listener**: contains the classes involved in an event. Check [How to create Listener of Event](https://confluence.gft.com/display/ISPP/How+to+create+Listener+of+Event).
    - **mapper**: contains the mapper between the objects.
    - **resource**: contains the RESOURCE objects of the API. **The main object of the RESOURCE must extend BaseResource** and **implement** the interface **Serializable**
    - **service**: contains the services interface with the connectors, usually there is one service for each API. **All classes** contained within in **must extend**the class **BaseService**.   
      In this package there is an other package named **facade** where there are the classes to call the connectors, one class for each connector.
    - **utils**: contains the utility classes.

**DEVELOPMENT**

To develop a new API follow these steps:

* Create the Controller where it is defined the interface of the API;
* To create the swagger of the interface of API. Use the maven plug-in **swagger-maven-plugin;**  
    
  Here is an example:

**Plug In To generate the swagger** Espandi fonte

* Create all modules necessary to call external services;
* Create all Connectors to call the external services;
* Create the Facades to implement the calls at the connector;
* Create the Services and implement one method for each facade you need;
* Create the Commands so you can implement the business logic using the services;
* Create the Factories to transform the API DTOs into OM;
* Create the Assemblers to transform the OM into RESOURCE of API
* Use in the Controller the Factory, Command and Assembler implemented to complete the API