OMNIS-TO-OMNIS BIDIRECTIONAL INTERCONNECTION AND ASYNCHRONOUS CONTENT MANAGEMENT

Francesco Del Dotto - fdotto@softpi.com

When is one **Omnis Studio** instance just not enough?

Three situations that answer to this question:

1. Long and complex, heavy-load tasks performed by JS Client sessions

2. A centralized multi-purpose service developed with Omnis Studio that has to be accessed remotely by many single-purpose server apps

3. Two or more Omnis Web Server instances that need to talk to each other but has to stay on separated environments

Long and complex tasks

- Huge PDF prints that need extensive calculations, picture management or tasks with big files involved can slow down the user’s workflow

- An user that can queue more time-consuming tasks to a «ghost user» that does their job can work on the main instance, waiting for results

Remote access to a multi-purpose Omnis Web Server

- If a set of general tasks and methods is used in multiple libraries, in order to update them we need to redeploy every one of them.

  If they are contained in a dedicated instance, their deployment can be done just once

- It is possible to create a modular set of Omnis App Servers working together to various extents, giving to each instance a specific designation, unlike Load Sharing

Interaction between Omnis instances

- If multiple instances can talk between them it is possible to create a system where asynchronous and direct interaction is needed.

  One example of that is a text chat or any other message system natively integrated in a library, that allows custom interactions with the library code

- It can help with situations that need to sandbox the user library and access directly data from other instances

- It can let libraries access other OS and their exclusive functions using a «twin» instance installed elsewhere

How it works?

The **Omnis Interface Reference** has three basic functions:

- Variable GET
- Variable SET
- Method execution

How it works?

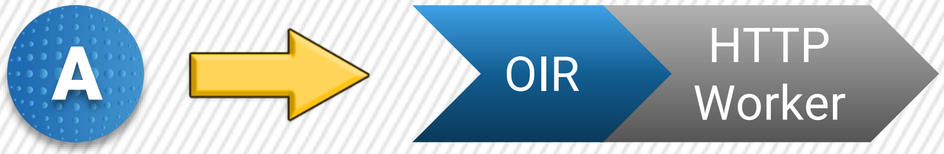
The OIR functions are managed with a three-part framework deployed on each side:

1. An HTTP OW3 Worker that sends the POST requests containing the data
2. A RESTful service that receives the requests and does a basic translation of the messages and calls the Object interface
3. An Object interface that collects the GET/SET and execution requests and prepares them for transmission when sending data and executes sync code when receiving data

How it works?

The OIR functions are managed with a three-part framework deployed on each side:

- GET
- SET
- Method



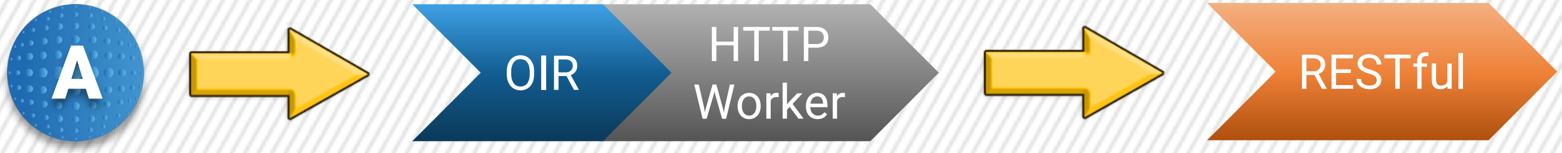
The request is sent to the OIR Object that use the HTTP Worker to call the RESTful service

How it works?

The OIR functions are managed with a three-part framework deployed on each side:

- GET
- SET
- Method

- Form $ident
- Task $ident
- Variable value
- Method parameters



The request is sent to the OIR Object that use the HTTP Worker to call the RESTful service

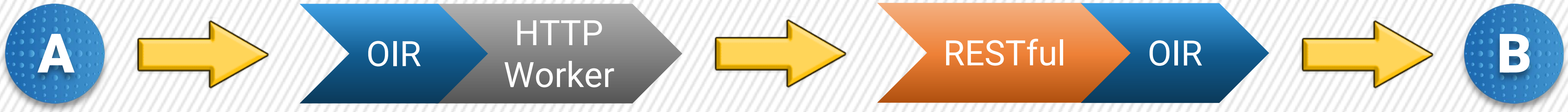
Sender «address» is sent to the other instance in order to trace it back

How it works?

The OIR functions are managed with a three-part framework deployed on each side:

- GET
- SET
- Method

- Form $ident
- Task $ident
- Variable value
- Method parameters



A

OIR

HTTP Worker

RESTful

OIR

B

The request is sent to the OIR Object that use the HTTP Worker to call the RESTful service

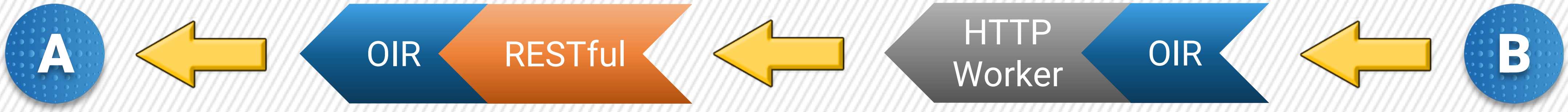
Sender «address» is sent to the other instance in order to trace it back

The receiveing OIR stores the data of the variable in a Startup_Task list, along with its original «address» or execute the request method from an Object or Code class

How it works?

It also works backwards!

- Original form $ident
- Original task $ident
- New variable value
- Method parameters



The receiveing OIR sets a task or instance variable that belongs to the original form or task

The OIR calls for a SET of a modified variable or a return method that starts a download on a client of the first Web Server

How it works?

What is it that runs at the core?

- The SET function relies on a Task list on the receiveing end that stores origin task, origin form name and value of a variable contained in the sending Web Server

- The values are stored as kRow type columns, in order to store any type of data inside of them

- A method can be executed using previously set variables and/or passing parameters to it

- Parameters and variable values are passed as JSON content. Task ident, form ident,variables info and method names are passed as header data

What about security?

- The interconnection is based on RESTful services and HTTP workers, so any connection can be managed and filtered using Omnis code, in-house encryption or any other method one can use to protect their RESTful web services

- The outer access to data is limited to writing values inside the task «master» list. The code can be written with this in mind and avoiding exposure to unwanted content

Can we have this natively?

Building an **Omnis Instance Reference** setup can be a burdensome task, especially in the beginning. Having some form of access to other instances, even still based on RESTful and HTTP OW3 Workers (technologies already natively supported by Omnis Studio) would clearly make this simpler.

Having this kind of capability in some native extent would be useful at least for accessing variables or calling methods in a similar fashion to what OLE Automation does.

If implemented, this could be a solution that leads to another level in Omnis programming that allows an interconnection capacity and flexibility in loada and task management hard to reach as of now.

Thank you!

Francesco Del Dotto - fdotto@softpi.com