### Hyperty Messaging Framework overview

This document gives an overview on the Messaging Framework technical solution used to support Hyperty's interaction through the higher level [Data Synchronisation Reporter - Observer communication mechanism](p2p-data-sync.md).

Hyperties cooperate each other with a Resource Oriented Messaging model implemented by a simple Messaging Framework. The Hyperty Messaging Framework, supports different messaging patterns including publish/subscribe and request/response messaging patterns. The higher level [Reporter - Observer communication pattern](p2p-data-sync.md) works on top of these basic messaging patterns.

The Message delivery is based on a simple message Router functionality that performs a lookup for listeners registered to receive the Message (the ["Message.to" Header field](https://github.com/reTHINK-project/dev-service-framework/blob/develop/docs/datamodel/message/readme.md#to) is the only information looked up for). The Message is posted to all found listeners, which can be other Routers or end-points (Hyperties). Thus, the Hyperty Messaging Framework is comprised by a network of Routers where each Router only knows adjacent registered Routers or end-points.



Hyperty Messaging Delivery Network

Listeners are programmatically registered and unregistered by Routing Management functionalities, which decide the listeners to be added according to a higher level view of the Routing Network.



Hyperty Message Routing Management

The Messaging Framework works at three layers:

At the Runtime Sandbox level where Hyperties are executing, message delivery is provided by the [MiniBUS component](https://github.com/reTHINK-project/dev-runtime-core/blob/master/src/bus/MiniBus.js).

At the Runtime level where Sandboxes are hosted (e.g. in a Browser or in a NodeJS instance), message delivery is provided by the [Message BUS component](https://github.com/reTHINK-project/dev-runtime-core/blob/master/src/bus/MessageBus.js), which is an extension of the MiniBUS.

At Domain Level, message delivery is provided by the [Message Node](msg-node.md) functionality by using the [Protofly mechanism](#protocol-on-the-fly-protofly-and-protostubs), i.e. communication between Message BUS and Message Nodes and among Message Nodes are protocol agnostic. This also means that the Message Node can be provided by any Messaging solution as soon as there is a [Protostub available](#protocol-on-the-fly-protofly-and-protostubs). Currently, a [Vertx Message Node](https://github.com/reTHINK-project/dev-msg-node-vertx), a [Matrix Message Node](https://github.com/reTHINK-project/dev-msg-node-matrix) and a [NodeJS Message Node](https://github.com/reTHINK-project/dev-msg-node-nodejs) are provided. These are just reference implementations of Message Nodes and anyone is free to develop its own Message Node. Check the [Message Node design guide](msg-node-design.md) and the [Protocol Stub specification](stub-specification.md) for more details.



Adhoc Messaging Oriented Middleware Routing Layers

At runtime level (MessageBUS and MiniBUS), it is used a standard CRUD based [JSON Message Model](../datamodel/message/readme.md), which is easily mapped into Restfull APIs.

More detailed description about the Hyperty Messaging Framework can be found for:

* [Protocol on-the-fly](protofly.md) mechanism
* [Message Node](msg-node.md) functional description
* [Peer to peer Message Delivery](p2p-msg-delivery.md) procedure
* [Data Synchronisation Reporter - Observer communication mechanism](p2p-data-sync.md)