## P2P Data Synchronisation: Reporter - Observer Model

This document gives an overview on how Hyperties cooperate each other through a Data Synchronisation model called Reporter - Observer. Details about how to develop Hyperties based on this model is provided in [this](development-of-hyperties.md) document.

The usage of Data synchronisation models in [Web Frameworks](https://www.meteor.com/ddp) looks very promising and is becoming very popular. The usage of the emerging [object.observe](https://developer.mozilla.org/pt-PT/docs/Web/JavaScript/Reference/Global_Objects/Object/observe) javascript API is making it even more appealing. However, current solutions require server-side databases that has an impact on performance and scalability.

Hyperty Reporter - Observer communication pattern goes beyond current solutions by using a P2P Synchronisation solution for JSON Data Objects, here called Hyperty Data Object or Sync Data Object. To avoid concurrency inconsistencies among peers, only one peer has granted writing permissions in the Hyperty Data Object - the **Reporter hyperty** - and all the other Hyperty instances only have permissions to read the Hyperty Data Object - the **Observer hyperty**.



Reporter-Observer Communication Pattern

The API to handle Hyperty Data Objects is extremely simple and fun to use. The Developer of the Hyperty Reporter just has to create the Data Sync object with the Syncher API,

syncher.create(\_this.\_objectDescriptorURL, [invitedHypertyURL], hello).then(function(dataObjectReporter) {  
  
 console.info('Hello Data Object was created: ', dataObjectReporter);

... and write on the object every time there is data to be updated and shared with Hyperty Observers:

dataObjectReporter.data.hello = "Bye!!";

On the Hyperty Observer side, Data Objects are subscribed with the Syncher API,

syncher.subscribe(\_this.\_objectDescriptorURL, ObjectUrl).then(function(dataObjectObserver) {}  
  
 console.info( dataObjectObserver);

... and the emerging [Object.observer() Javascript method](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/observe) is used to receive the stream of data changes coming from the Reporter Hyperty.

dataObjectObserver.onChange('\*', function(event) {  
 // Hello World Object was changed  
 console.info(dataObjectObserver);  
 });

### Hyperty Data Object URL address

The Hyperty Messaging Framework allocates to each new created Hyperty Data Object a Global Unique Identifier URL that is independent from the Hyperty instance creator and from the Hyperty Runtime, in order to support mobility of the Data Object between different Hyperty Runtimes and also to support delegation of the Reporter role to other Hyperty instances. However, at this point Reporter delegation is only supported between Hyperty instances from the same domain.

### Hyperty Data Object Schema

Each Hyperty Data Object is formally described by a json-schema that is identified by a Catalogue URL. This allows to check whether two different Hyperties are compliant by cross checking each supported Hyperty Data Object schema. At this point the following Hyperty Data Object schemas are defined:

* [**Connection Data Schema**](../datamodel/connection) : Hyperties supporting this schema are able to handle [WebRTC Peer Connections](https://developer.mozilla.org/en-US/docs/Web/Guide/API/WebRTC/Peer-to-peer_communications_with_WebRTC) between the Hyperty Runtime instances where they are running independently of the signalling protocol used. The URL Scheme for Connection Data Objects is "connection" (example: "connection://example.com/alice/bob201601290617").
* [**Communication Data Schema**](../datamodel/communication) : Hyperties supporting this schema are able to handle different communication types including Textual Chat, Audio, Video, Screen Sharing and File sharing. Such communication can be supported on top of WebRTC protocol streams by using the Connection Data Schema. The URL Scheme for Communication Data Objects is "comm" (example: "comm://example.com/group-chat/rethink201601290617").
* [**Context Data Schema**](../datamodel/context) : Hyperties supporting this schema are able to produce or consume Context Data, usually collected from sensors. The URL Scheme for Communication Data Objects is "ctxt" (example: "ctxt://example.com/room/temperature201601290617").

### Parent - Children Resources

In order to allow use cases like Group Chat where all involved Hyperties are able to write in the Sync Data Object, the Parent - Child Data Sync Objects is introduced.

A Data Object Child belongs to a Data Object Parent children resource and can be created by any Observer of the Data Object Parent as well as by its Reporter. The Reporter - Observer rules still apply to Data Object Child i.e. there is only one Reporter that can update the Data Object Child, which can be an Observer of the Data Object Parent, as mentioned earlier.



Parent - Child Sync

The creation, update and delete of an Data Object Child is performed in the Data Object Parent itself:

\*Data Object Child creation, update and delete code snippet\*

All other Hyperties observing or reporting the Data Object Parent, will be notified every time a new Data Object Child is created, updated or deleted:

\*Data Object Child creation, update and delete notification code snippet\*

At this point, Data Object Child can't also be a Data Object Parent of another Sync Data Object, i.e. Hyperty Data Object composition is limited to one level.

### Syncher and Sync Manager

This section, gives an overview on how the Hyperty Data Object synchronisation transparently works on top of the [Hyperty Messaging Middleware](hyperty-messaging-middleware.md). However, Hyperty developers don't have to know the technical details of this solution and can directly move to the [Hyperty Development Manual](development-of-hyperties.md).

The Hyperty Data Object synchronisation is provided by two components in the Runtime:

The [Syncher](https://github.com/reTHINK-project/dev-service-framework/blob/master/src/syncher/Syncher.js) is a singleton Component co-located with the Hyperty Instance, which is in charge of handling all required procedures to manage data synchronisation at the Hyperty instance side, as a Reporter or a Observer Hyperty.

The [Runtime Sync Manager](https://github.com/reTHINK-project/dev-service-framework/blob/master/src/syncher/Syncher.js) is a Core Runtime Component, which is in charge of handling authorisation requests to create Sync Data Objects from Hyperty Reporters and subscription requests to Sync Data Objects from Hyperty Observers. As soon as authorisation is granted the Sync Manager handles all required MessageBUS listeners in order to setup the Data Sync Stream routing path among Reporters and Observers. I.e., the Runtime Sync Manager provides a [Messaging Middleware](hyperty-messaging-middleware.md) Routing Manager functionality.

The [Message Node Sync Manager](https://github.com/reTHINK-project/dev-service-framework/blob/master/src/syncher/Syncher.js) is a Message Node functionality, which is in charge of handling requests from Runtime Sync Managers in order to setup the Data Sync Stream routing path between the Reporter Hyperty Runtime and Observers Hyperty Runtimes. I.e., the Message Node Sync Manager also provides a [Messaging Middleware](hyperty-messaging-middleware.md) Routing Manager functionality..



Routing Management for Hyperty Data Syncronisation

A detailed description of the Hyperty Data Synchronisation procedures are provided [here](https://github.com/reTHINK-project/core-framework/blob/master/docs/specs/runtime/dynamic-view/data-sync/readme.md)