

**Deliverable D3.5**

**Hyperty Runtime and Hyperty Messaging Node Specification**

|  |  |
| --- | --- |
| Editor: | Paulo Chainho, PT Inovação |
| Deliverable nature: | (R) Document, Report |
| Dissemination level: (Confidentiality) | Public (PU) |
| Contractual delivery date: | 30/09/2016 |
| Actual delivery date: | 30/09/2016 |
| Suggested readers: | Service providers’ designers and developers |
| Version: | 0.1 |
| Total number of pages: | 22 |
| Keywords: | Fixed: Communication networks, media, information society  Free: Hyper-linked entities, Future Internet architecture, Trustful communications, Social trustful networks, WebRTC, Independent Identity, IdP, Peer-to-peer communications,H2H, M2M, Graph ID |

***Abstract***

This Report contains a detailed specification of reTHINK Core Framework components comprised by the runtime environment where Hyperties are executed and the messaging nodes used to support messages exchange between Hyperties. This specification is sustained by a very comprehensivee work in terms of state of the art research and procurement of existing open source that will be used to demonstrate the feasibility of the radical reTHINK concepts. The core of this report contains a detailed specification of the Hyperty Runtime API and of the main procedures to support use cases, requirements and concepts defined in previous reports, providing the basis for the implementation tasks.

[End of abstract]

Disclaimer

This document contains material, which is the copyright of certain reTHINK consortium parties, and may not be reproduced or copied without permission.

All reTHINK consortium parties have agreed to full publication of this document.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the reTHINK consortium as a whole, nor a certain part of the reTHINK consortium, warrant that the information contained in this document is capable of use, nor that use of the information is free from risk, accepting no liability for loss or damage suffered by any person using this information.

*This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 645342. This publication reflects only the author’s view and the European Commission is not responsible for any use that may be made of the information it contains.*

Impressum

Full project title: **Trustful hyper-linked entities in dynamic networks**

Short project title: **reTHINK**

Number and title of work-package: **WP3 – Core Framework Implementation**

Number and title of task: **Task 3.1 Procurement and Specifications**

Document number and title: **D3.1 Hyperty Runtime and Hyperty Messaging Node Specification**

Editor: **Paulo Chainho, Altice Labs**

Work-package leader: **Paulo Chainho**, company: **Altice Labs**

Copyright notice

2016 Participants in project RETHINK

This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 Unported License. To view a copy of this license, visit [http://creativecommons.org/licenses/by-nc-nd/3.0http://creativecommons.org/licenses/by-nc-nd/3.0http://creativecommons.org/licenses/by-nc-nd/3.0](http://creativecommons.org/licenses/by-nc-nd/3.0/)

Executive summary

This document describes the technical details and the information needed by developers to start prototyping reTHINK Core Framework, which is comprised of the runtime environment where Hyperties are executed and the messaging nodes used to support messages exchange between Hyperties. This document takes as input the conceptual foundations, data models and interfaces definitions from deliverables D2.1 (The reThink Framework Architecture) and D2.2 (the reTHINK Data Model). This report complements deliverable D4.1 (Management and Security features specifications), which specifies reTHINK Support Services, namely: Policy Management, Governance, Identity Management, Graph Connector, and Hyperty Directory services (Catalogue and Registry). The core of this document is dedicated to the detailed specification of the Hyperty Runtime describing in detail, the Hyperty Runtime architecture and the Core Runtime components required to support the execution of Hyperties. The Hyperty Runtime architecture follows a security by design approach since it was highly influenced by a careful security analysis where different types of components are executed in isolated sandboxes. Thus, components downloaded from a specific Service Provider are executed in sandboxes that are different from the sandboxes used to execute components downloaded from another service provider. Communication between components running in different sandboxes is only possible through messages exchanged through a Message Bus functionality provided by the Hyperty Runtime Core Sandbox. The access to the Message BUS functionality is controlled by a Policy Engine which is also located in the Core Runtime sandbox. On the other hand, and according to the ProtoOFly concept introduced in D2.1, the protocol stub is executed in isolated sandbox and provides the bridge for the Hperty Runtime to communicate with associated Service Provider.

The design of the Hyperty Runtime APIs progressed along the design of the main procedures to be performed in order to validate it with the most important use cases that were already used in D2.1 and originally described in D1.1. Thus, basic procedures (e.g. message routing and Hyperty deployment), Identity Management Procedures (e.g. registration and login of users) and Human to Human communication procedures were detailed, including the definition of the data sets and messages as defined in D2.2. The Hyperty Runtime design was also partially validated with Machine to Machine communication and Human to Machine communication use cases, which will be fully reported in D3.2.

Special attention was given on the design of components involved in the Reporter-Observer data synchronisation communication pattern introduced in D2.2, which complements the ProtOFly concepts to support seamless interoperability between domains at service layer. The access control to synchronised objects, through the Reporter-Observer communication pattern, is enforced by the Core Policy Engine. More sophisticated and proprietary data synchronisation algorithms can be used, by enabling the deployment of other Policy Enforcer in the Hyperty Runtime, which will be executed in isolated sandboxes.

A reference design for the Messaging Node Architecture is also provided in this report. Since the protocol-on-the fly concept is used together with the message model defined in D2.2, it is not required to specify in detail the Messaging Node APIs to guarantee interoperability between different domains.

Together, the Hyperty Runtime and the Messaging Node specifications are based on a set of design principles to support Hyperty Instance Mobility (between Network Interfaces and also between Devices), Data Object portability (between Hyperty Instances) and group communication. These characteristics are supported by the usage of different virtual addresses separately allocated to Hyperty Instances and Data Objects, which are agnostic of the network addresses. Hyperties communicate each other by publishing messages on the target Hyperty Instance virtual address, or, in case the Reporter-Observer communication pattern is used, on the synchronised data object virtual address. Any Hyperty Instance granted with authorisation to listen on those virtual addresses, will receive the messages. The separation of concern design principle was also used in order to let Hyperty developers focus on its service logic and leaving business related decisions to product managers, as well as giving the users more control on how service is delivered. As a consequence of this principle, by default, the different security tokens used (including ID Tokens and Access Tokens) are handled by the Core Runtime and not by the Hyperty Instances.

The reTHINK Core Framework detailed specification is achieved by a comprehensive effort on web runtime design state of the art research with special attention given to Security in Web Runtime and relevant W3C and IETF standards. A comprehensive report about the procurement of existing open source solutions to be used to prototype reTHINK Core Framework components, is also presented, mainly in terms of Web Runtime Solutions and Real Time Messaging Solutions.

Taking as input the procurement report, some solutions were selected and some implementation considerations are presented for the Hyperty Runtime and for the messaging solutions.

Some preliminary design guidelines are provided for the implementation of the Hyperty Service Framework. The Hyperty Service Framework is a Software Development Toolkit (SDK) that will feature a comprehensive set of application program interfaces (APIs) and JavaScript libraries to facilitate the development of Hyperties.

It should be noted that the Network Platform specification supporting Specialised Network Services is an ongoing work that will be reported later in D3.4, as originally planned.

List of authors

|  |  |
| --- | --- |
| Company | Author |
| PTIN | Paulo Chainho  Vitor Silva  Luis Duarte |
| Fraunhofer |  |
| Quobis |  |
| Apizee |  |
| Deutsche Telekom |  |
| Orange Labs |  |
| INESC-ID |  |
| IMT |  |
|  |  |

Table of Contents

[Executive summary 3](#_Toc466480175)

[List of authors 5](#_Toc466480176)

[Table of Contents 6](#_Toc466480177)

[Abbreviations 7](#_Toc466480178)

[1 Introduction 9](#_Toc466480179)

[1.1 Objectives and Overview 9](#_Toc466480180)

[1.2 Structure 11](#_Toc466480181)

[2 Core Framework Specification update 12](#_Toc466480182)

[2.1 Runtime Specification Update 12](#_Toc466480183)

[2.1.1 Functional Summary 12](#_Toc466480184)

[2.1.2 Main Changes performed in Phase 1 14](#_Toc466480185)

[2.1.3 Main Specification Updates for Phase 2 (around 5-10 pages) 15](#_Toc466480186)

[2.2 Messaging Framework Specification Update (Steffen) 16](#_Toc466480187)

[2.2.1 Hyperty Messaging Framework Overview 16](#_Toc466480188)

[2.2.2 Protocol on-the-fly 16](#_Toc466480189)

[2.2.3 Data Synchronization mechanism (Reporter-Observer) 16](#_Toc466480190)

[2.2.4 Messaging Node 16](#_Toc466480191)

[2.2.4.4 Available reference Implementations 16](#_Toc466480192)

[2.3 Service Framework Specification Update (Marc) 16](#_Toc466480193)

[2.3.1 Service Framework Specification Overview 16](#_Toc466480194)

[1.1.2 DataObjectChild 25](#_Toc466480195)

[3 New Features specification 28](#_Toc466480196)

[3.1 Runtime Trust Management Specification (Ricardo Chaves/Nuno) 28](#_Toc466480197)

[3.2 P2P Protofly Specification (Paulo) 28](#_Toc466480198)

[3.3 QoS Control specification (Marc) 28](#_Toc466480199)

[3.4 Multiparty WebRTC Connections specification (Arnaut) 28](#_Toc466480200)

[3.5 Interworking with Legacy Services (Anton) 28](#_Toc466480201)

[4 Conclusions 29](#_Toc466480202)

[5 References 30](#_Toc466480203)

Abbreviations

|  |  |
| --- | --- |
| API | Application Programming Interface |
| COAP | Constrained Application Protocol |
| CRUD | Create, Retrieve, Update and Delete |
| CSP | Communication service provider |
| DDoS | Distributed Denial of Service Attacks |
| DoS | Denial of Service |
| H2H | Human to Human communication |
| ICE | Information and Content Exchange |
| IETF | Internet Engineering Task Force |
| JSON | JavaScript Object Notation |
| LWM2M | LightweightM2M |
| M2M | Machine to Machine communication |
| ORTC | Object Real-Time Communications |
| QoS | Quality of Service |
| REST | Representational State Transfer |
| STUN | Session Traversal Utilities for NAT |
| TURN | Traversal Using Relay NAT |
| UML | Unified Modelling Language |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| W3C | World Wide Web Consortium |
| WHATWG | Web Hypertext Application Technology Working Group |
| SPPE | Service Provider Policy Enforcer |
| PEE | Policy Enforcer Engine |
| TRAM | TURN Revised and Modernized |
| HTTP | Hypertext Transfer Protocol |
| TCP | Transmission Control Protocol |
| QUIC | Quick UDP Internet Connections |
| XMPP | Extensible Messaging and Presence Protocol |
| ORTC | Object Real-Time Communications |
| COAP | Constrained Application Protocol |
| LWM2M | Lightweight M2M |
| SDT | Smart Device Template |
| HGI | Home Gateway Iniative |
| SFU | Selective Forwarding Unit |
| MCU | Multipoint Control Unit |
| TLS | Transport Layer Security |
| MQTT | MQ Telemetry Transport |
| WAC | WebRTC Application Controller |
| AAA | Authentication, Authorization and Accounting |
| OSS | Operations Support System |
| BSS | [business support systems](https://en.wikipedia.org/wiki/Business_support_system) |
| RCS | Rich Communication Services |
| UC | Unified Communications |
| CRM | [Customer Relationship Management](https://en.wikipedia.org/wiki/Customer_relationship_management) |
| JSONoWS | JSON over Web Sockets |
| IdP | Identity Provider |
| TCB | Trusted Computing Base |
| PDP | Policy Decision Point |
| PEP | Policy Enforcement Point |

# Introduction

## Objectives and Overview

Project reTHINK proposes a radical transformation on how real time communication services are thought. reTHINK concepts and architecture represents a significant paradigm change for the communication services domain. The reTHINK approach enables the fulfilment of real-time communications requirements that so far have been considered impossible to achieve: trustful identities, interoperable endpoints, agility of introducing new services, and fast moving innovation. Previous Deliverables D2.1 [38] and D2.2 [15] have already started enlightening the path to reach such objectives. A new web service paradigm, the so-called Hyperlinked Entities - Hyperties – was introduced to enable a global network of trustful services executing in web runtime environment, on end-user devices or edge-network servers. Communication between Hyperties is based on the protocol-on-the-fly (ProtoFly) concept that avoids creating or modifying standard network protocols, but utilizes instead standard APIs. Interoperability between Hyperties and Support Services (Registry, Catalog, and Identity Management) are assured by a detailed and extensible data model, combined with the principle of Hypermedia as the Engine of Application State (HATEOAS) as defined in D2.2.

This report provides a detailed specification of reTHINK Core Framework components comprised by the runtime environment where Hyperties are executed and the messaging nodes used to support messages exchange between Hyperties. This report complements deliverable D4.1 (Management and Security features specifications)[109], which specifies reTHINK Support Services, namely: Policy Management, Governance, Identity Management, Graph Connector, and Hyperty Directory services (Catalogue and Registry). Thus, and according to reTHINK Architecture [38], the scope of this report includes the specification of the Messaging Node providing reTHINK Messaging Services and the specification of the Hyperty Runtime that will be included in User Devices and Application Servers to deliver User Hyperties and Network Side Hyperties (See Figure 1).



Figure 1 - Specification Scope

It should be noted that the Network Platform specification supporting Specialised Network Services will be reported later in D3.4, as originally planned.

The reTHINK Core Framework specification provided in this report, is compliant with reTHINK Data Model, Hyperty Management interfaces, Stream Interface and Messaging Interface designed in D2.2 [15]. It should be noted that, according to Protocol On-the-fly concept, the Messaging Interface is defined by the Message Model defined in [15].

Besides the Architecture requirements reported in D2.1 [38] additional specific requirements to Core Framework functionalities were analysed.

The specification of the Hyperty Runtime and the Messaging Node is sustained by a very comprehensive work in terms of state of the art research and procurement of existing open source that will be used to demonstrate the feasibility of the radical reTHINK concepts.

An exhaustive study of relevant IETF, W3C standards and others that facilitate the fulfillment of previously analysed requirements, is reported. Special attention was given to the research on security in Web Runtime. In parallel, existing open source solutions to be used to develop Hyperty Runtime and Messaging Nodes was researched, experimented and selected.

Three solutions to implement the Messaging Node were selected, in order to evaluate in reTHINK testbeds, interoperability between different Hyperties domains that use different Message Nodes, namely Vertx, Node.js and Matrix.

The experimentations performed on JavaScript engines and WebRTC implementations have shown to be very difficult to extend existing runtimes like V8 or Chromium to natively support Hyperties runtime. On the other hand, such approach would also not promote the adoption of Hyperty Runtime by the end-users since it would demand the installation of new platforms to replace popular browsers like Chrome or Firefox. Instead, it was decided to make Hyperty Runtime compliant with existing runtime solutions notably with existing Web Browsers like Chrome and JavaScript platforms like Node.js.

The Runtime design enables reuse of most of the core runtime components through different platforms including Browsers, Standalone Mobile Application, Network Side Application Servers and more constrained M2M/IoT standalone devices. The Hyperty Runtime architecture follows a security by design approach where different types of components are executed in isolated sandboxes. Communication between different sandboxes is only possible through a Message Bus and is subject to access control. Communication with remote Hyperties is provided by protocol stubs executed in isolated sandboxes.

The design of the Hyperty Runtime APIs is validated with the most important use cases that were already used in D2.1 and originally described in D1.1. The Hyperty Runtime procedures were described for basic procedures (e.g. message routing and Hyperty deployment), Identity Management Procedures (e.g. registration and login of users) and Human to Human communication. Although, the Hyperty Runtime is designed to also support Machine to Machine communication and Human to Machine communication use cases, its procedures will be fully reported in D3.2.

The Messaging Node Reference Architecture is described to provide some guidelines for Messaging Node implementation. Thanks to the protocol-on-the fly concept, a detailed specification of Messaging Node APIs as provided for the Hyperty Runtime, is not required. Instead, a more detailed specification is provided for each messaging solution selected during the procurement activity namely for Vertx.io, Node.js and Matrix.

The main functionalities to be provided by the Hyperty Service Framework, which will be used by Hyperty Developers, is provided at the end. The Hyperty Service Framework is a Software Development Toolkit (SDK) that will feature a comprehensive set of application program interfaces (APIs) and JavaScript libraries to facilitate the development of Hyperties.

The specification reported in this deliverable, provides the basis for the implementation tasks but it is expected to be adjusted and to be completed along the implementation phase.

The final specification for Messaging Node and Hyperty Runtime will be reported in D3.3 (Hyperty Runtime and Hyperty Messaging Node Phase 2 – Dec 2016).

## Structure

This report starts with an introduction and, in Chapter 2, requirements that are more specific to the reTHINK Core Framework are clearly identified. In chapter 3 a summary of the State of the Art and Procurement work is given. The full State of the Art and Procurement report can be found in Annex A. The core part of this report is located in Chapter 4, which details the specification of the Hyperty Runtime, and in Chapter 5, the specification of the Messaging Node. This reports concludes with a short description of functionalities to be provided by the Hyperty Service Framework to be used by Hyperty Developers.

# Core Framework Specification update

## Runtime Specification Update

This section contains an overview of the Hyperty Runtime specification, where Hyperties are executed. It provides a summary of functionalities provided, main changes performed in phase 1 since the initial specification and the specification of updated for phase 2.

### Functional Summary

The main functionality provided by the Runtime is the safe execution of Hyperties. Different types of components (see Figure 1) with different origins are deployed and executed in isolated sandboxes including Runtime Core Components, Hyperties and Protostubs.

The Runtime Core functionalities are comprised by:

The **Runtime User Agent** that manages the lifecycle of the Runtime itself as well as of Hyperties and Protostubs, including the deployment, update and removal of these functionalities.

The **Runtime Registry** handles the registration of all available runtime components including Core components, Service Provider Sandboxes and each component executing in each sandbox like Hyperty Instances, Protocol Stubs and Applications.

The **Runtime Catalogue** manages the descriptors of deployable components and Hyperty Data Object schemas that are downloaded from the Service Provider Catalogue.

The **Message Bus** supports local message communication in a loosely coupled manner between Service Provider sandboxes including Hyperty Instances, Protocol Stubs and Policy Enforcers.

The **Runtime Identity Module** manages ID and Access Tokens required to trustfully manage Hyperty Instances communication including the generation and validation of Identity assertions.

The **Policy Engine** provides Policy decision and Policy Enforcement functionalities for messages intercepted from the Message BUS.

The **Sync Manager** handles data synchronisation streams used by Hyperties to communicate each other.

The **QoS User Agent** Manages network QoS in the runtime.

The **Graph Connector** is a local address book maintaining a list of trustful communication users.

The only important Runtime APIs to be used by Applications and Hyperties are:

* Runtime User Agent APIs that are used by Applications to instantiate Hyperties and Protostubs
* Message Bus APIs that are used by Hyperties to send and receive messages

The remaining APIs are internal to the core runtime, thus developers of Hyperties and Applications have not to deal with them.

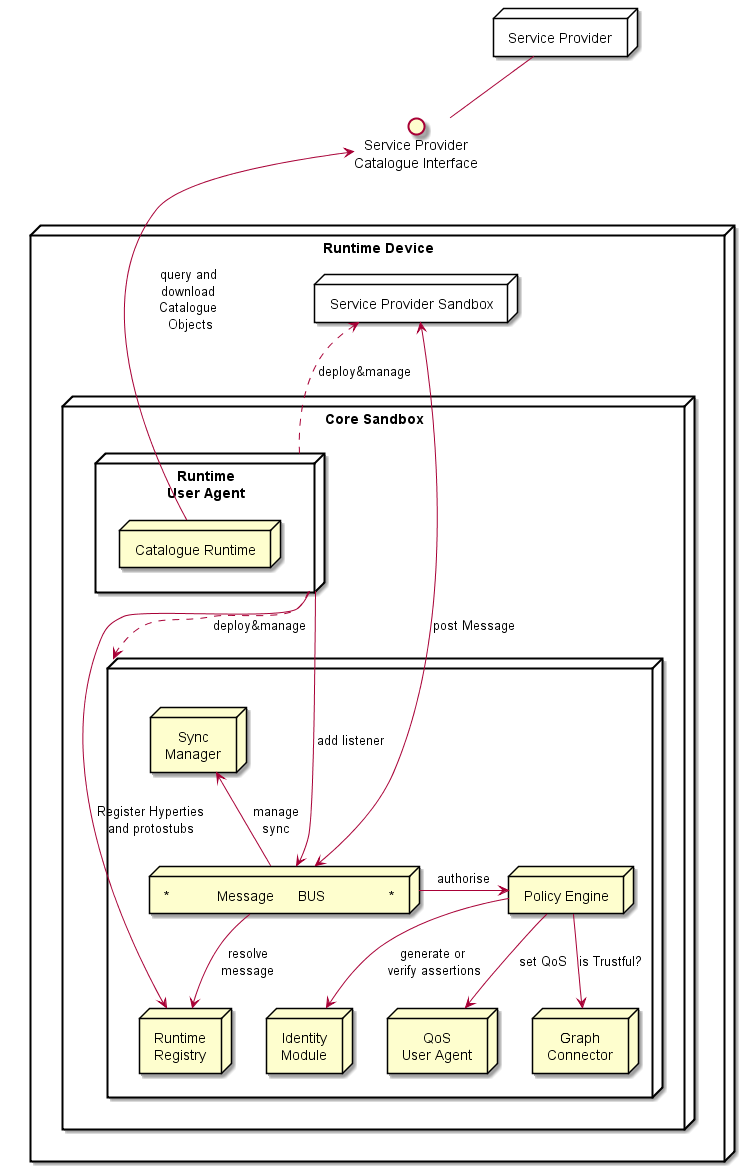


Figure 1 – Runtime Architecture

For further information, the full specification is publicaly available in the Github:

* the functional specification at ? .
* the Runtime Main procedures at ? .
* the detailed definition of messages at ?.
* the APIs at ?.

### Main Changes performed in Phase 1

As highlighted in Figure 2, comparing with the original specification, the **Sync Manager** was addedin in order to support Hyperty Data Objects synchronisation by handling creation and subscriptions requests.

In addition, the **CatalogueProtostub** from the **Runtime User Agent** was renamed to **Runtime Catalogue** and has a few more functionalities. It manages the descriptors of deployable components and Hyperty Data Object schemas that are downloaded from the Service Provider Catalogue via the Catalogue Service interface. The Runtime Catalogue ensures synchronisation with Back-end Catalogue servers.

The **QoS User Agent** and the **Graph** **Connector** will be implemented / integrated in the Runtime during phase 2.

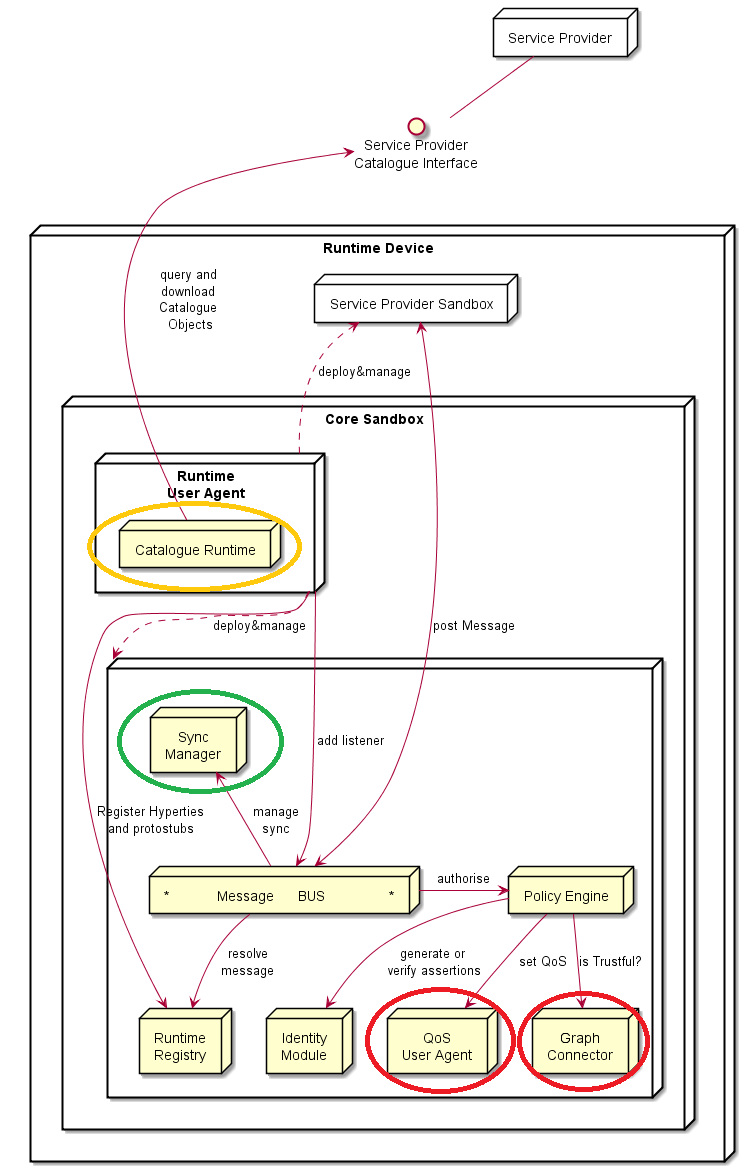


Figure 2 - Changes in the Hyperty Runtime Core

The specification of the Runtime Procedures was further elaborated and detailed, notably by fully specifying the messages to be used for each procedure [?].

### Main Specification Updates for Phase 2 (around 5-10 pages)

Highlight main changes and new specs to be implemented in phase 2 referring to "New Features specification" sections when appropriate. When possible it should refer to new components, APIs, messages and dynamic view MSC diagrams provided in the Github

## Messaging Framework Specification Update (Steffen)

This Chapter contains the functional design of the Messaging Node Architecture which enables messaging communication among Hyperty instances running in different Runtime devices.

Since the protocol-on-the fly concept is used together with the message model defined in D2.2, it is not required to specify in detail the Messaging Node APIs to guarantee interoperability between different domains.

### Hyperty Messaging Framework Overview

https://github.com/reTHINK-project/specs/tree/master/messaging-framework

<https://github.com/reTHINK-project/specs/blob/master/messaging-framework/readme.md>

(without the 4 sub-sections that are linked at the bottom of this page)

### Protocol on-the-fly

https://github.com/reTHINK-project/specs/blob/master/messaging-framework/protofly.md

### Data Synchronization mechanism (Reporter-Observer)

https://github.com/reTHINK-project/specs/blob/master/messaging-framework/p2p-data-sync.md

### Messaging Node

#### Functional Architecture

https://github.com/reTHINK-project/specs/blob/master/messaging-framework/msg-node.md

<https://github.com/reTHINK-project/specs/blob/master/messaging-framework/msg-node.md>Functional Architecture

#### General design recommendations

https://github.com/reTHINK-project/specs/blob/master/messaging-framework/msg-node-design.md

#### Stub speccification

https://github.com/reTHINK-project/specs/blob/master/messaging-framework/stub-specification.md

### Available reference Implementations

List the MNs and provide links to their github-pages.

#### Lessons learned

https://github.com/reTHINK-project/specs/blob/master/tutorials/msg-node-development-recommendations.md

<https://github.com/reTHINK-project/specs/blob/master/tutorials/msg-node-development-recommendations.md>

## Service Framework Specification Update (Marc)

### Service Framework Specification Overview

The reTHINK Service Framework provides a comprehensive set of application program interfaces (namly APIs) and JavaScript-based libraries to support the development of Hyperties. As such, the Service Framework is agnostic with respect to the underlying messaging node, modular in design, and to the widest degree agnostic to devices and their operating systems, such as Android, iOS, Raspberry PI, Linux, or Windows. It features a comprehensive set of application program interfaces (APIs) and JavaScript libraries to facilitate the development of Hyperties within the reTHINK architecture.

As such, the reTHINK Service Framework enables the design of the Hyperty Runtime APIs to be developer-friendly, i.e., the latter only have to focus on a few core functionalities, namely:

- MsgBUS.postMessage() that is used to post messages in order to communicate with other remote Hyperty Instances and with back-end reTHINK Support Services,

- Syncher API that is used to communicate through the Reporter-Observer communication pattern, and potentially

- the implementation of the hyperty init() function, used to activate the Hyperty Instance with required configuration parameters.

This chapter updates the initial specificaiton of the reTHINK Service Framework as given in [D3.1]. For briefness, unaltered, valid descriptions are cross-referenced and updates and additions are given where appropriate. Those additions and updates focus on:

\* Service Framework Address and Message Factory

\* Synchronizaiton among Hyperties (Syncer API)

\* Discovery and Identity Manager library,

\* QoS interface and LHCB library,

\* Runtime Capabilities Manager library

\* Storage Manager

From D1.2 user scenarios analysis the following additional functionalities were defined:

- Communication Service: the Hyperty Runtime already provides an API for H2H and M2M communication. Developers will be able to use this service directly from the Hyperty Runtime API

- Identity Service: a provider mechanism to access internal reTHNK IdP services or external IdPs (Google, Facebook, etc.)

- Data Storage functions: for storing persistent data

- Location functions: to access device specific context (e.g. GPS) to be used as context for different services

- User Entity Management: to manage one or multiple user profiles\*Notification service: for notifying triggered events

Functionaliy provided by the reTHINK Service Framework was then based on closing the gaps between existing functionality provided by open source JavaScript libraries and the complete functionality requirements coming from the use cases.

### Service Framework Specification updates

*Adapt content below to only provide a summary of changes including new libs added*

#### Service Framework Address and Message Factory

##### Address Resource Factory

The Address Resource Factory creates the different types of URLs required as specified [here](https://github.com/reTHINK-project/specs/tree/master/datamodel). It is compliant with the API described by [WHATWG](https://url.spec.whatwg.org/#api)

###### Address Data Model

All the attributes below are internally getter/setters functions. In the setter functions value validation will be included.

* href - It allows to get and set the complete URL string.
* scheme - It allows to get and set the scheme of the URL. In traditional URL it defines the protocol for which the URL was intended for. It can be any of the schemes defined in reTHINK: domain, hyperty, hyperty-catalogue, hyperty-runtime, comm, ctxt, acct, user-uuid and user.
* username - It allows to get and set the username of the URL.
* password - It allows to get and set the password of the URL.
* host - It allows to get and set the host of the URL.
* hostname - It allows to get and set the hostname of the URL.
* port - It allows to get and set the port of the URL.
* pathname - It allows to get and set the pathname of the URL.
* search - It allows to get and set the query string of the URL.
* searchParams - It allows to get the query string of the URL according to the interface [URLSearchParams](https://url.spec.whatwg.org/" \l "urlsearchparams)
* hash - It allows to get and set the fragment identifier of the URL.

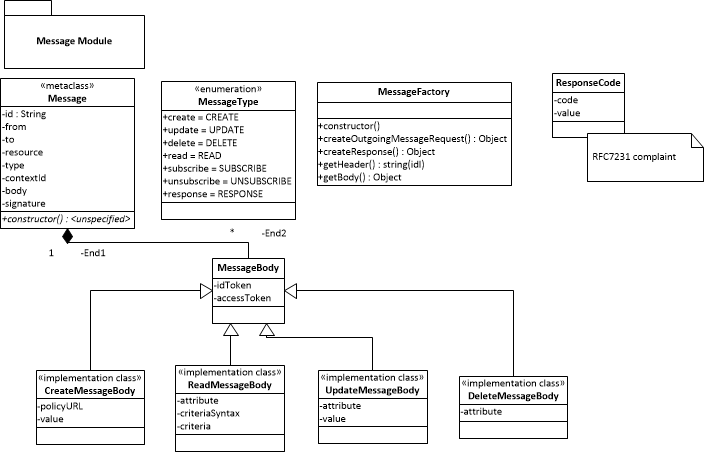
##### Functions

* constructor(url, base) - Constructor function
* schemeValidation() - this function validates if the URL is syntactically compliant with any of the schemes defined for reTHINK. It will also validate standard schemes.

##### Static functions

* urlToASCII(domain) - This static function return the URL in ASCII code (to be discussed if it's necessary)
* urlToUnicode(domain) - This static function returns the URL in unicode (to be discussed if it's necessary)

##### Message Resource Factory

[](https://github.com/reTHINK-project/specs/blob/master/service-framework/message%20module.png)

#### Class Message

The Message Class has following class attributes:

* id - the identifier to be used to associate Response messages to the initial request message
* type - from MessageType Enumeration class/variable
* contextID - GUID used to identify the context for example communication session
* from - URL of the Hyperty instance or assoiciated User
* to - one or more URLs of the recipeints
* resourceURL - the URL of the reTHINK Data Object resource associated with this message. Used for routing purposes.
* messageBody - from the MessageBody data object

#### MessageType (Enumeration)

var MessageType = new enums.Enum("CREATE", "UPDATE", "DELETE", "READ", "SUBSCRIBE", "UNSUBSCRIBE", "RESPONSE");

#### Class MessageBody

* idToken - optional attribute (JWT) for Identity assertion purpose
* accessToken - optional attribute (JWT) for access control purpose

##### Class CreateMessageBody extends MessageBody

* policyURL - URL from where to download the access policy control
* value - JSON formatted data to create (TODO: has this been specified on any document so far?)

##### Class ReadMessageBody extends MessageBody

* attribute- attribute in the object to be read
* value - value of the read attribute

##### Class DeleteMessageBody extends MessageBody

* attribute- attribute in the object to be deleted

##### Class UpdateMessageBody extends MessageBody

* attribute- attribute in the object to be modified
* value - new value of the attribute

##### Class ResponseMessageBody extends MessageBody

* code- a response code complaint to HTTP response codes (RFC7231)
* value - data value in JSON format (used as value to read message requests)

##### Class ResponseCode

Enumeration of all response codes according to RFC7231

#### Class MessageFactory

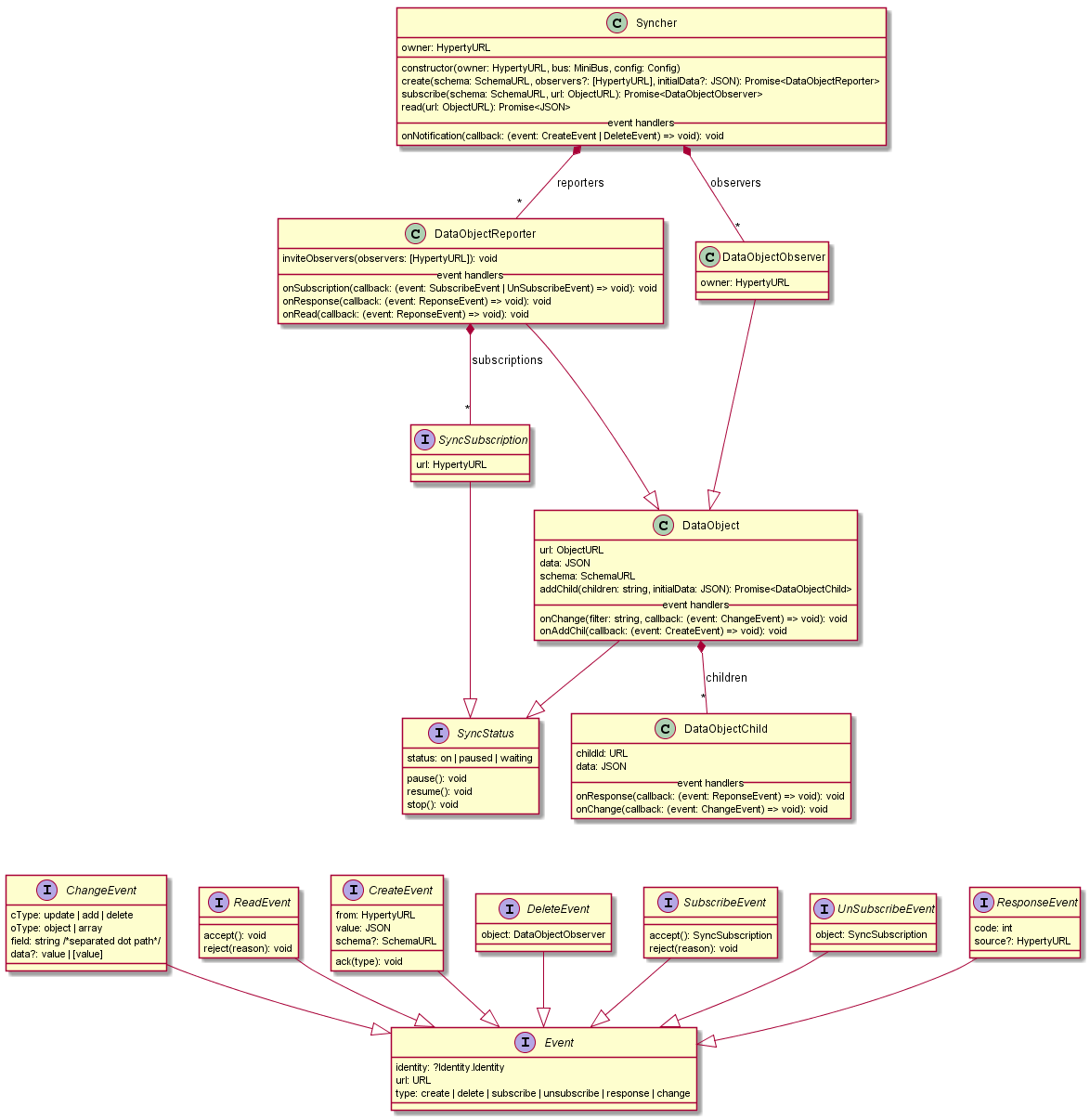
The MessageFactory creates messages according to the [Message Data Model](https://github.com/reTHINK-project/specs/tree/master/datamodel) to be sent through the Runtime Message Bus.

#### Methods

* constructor(...)
* createMessageRequest(URL.URLList to, MessageType type, Object extraHeaders, MessageBody body )
* createDeleteMessageRequest(Message data)
* createUpdateMessageRequest(Message data, String attribute, String value)
* createReadMessageRequest(Message data, String attribute, String criteria, String criteriaSyntax)
* createResponse(Message data, ResponseCode code)
* getHeader(Message data, String key) returns String
* getBody(Message data): returns JSON object

#### Synchronizaiton among Hyperties (Syncer API)

The Syncher API provides data object synchronisation among Hyperties. The synchronised Data Objects are JSON data objects. The Syncher API is depicted in the following diagram:

[](https://github.com/reTHINK-project/specs/blob/master/service-framework/SyncherAPI.png)

The Syncher is a singleton owned by a Hyperty Instance that uses it to communicate with other Hyperty instance through data synchronisation. The Syncher "owns" all DataObjects (DataObject class) used by its Hyperty Instance i.e. DataObject instances (creation, destruction) are managed by the Syncher and not by the Hyperty Instance. Each DataObject is addressed by a URL - ObjectURL - that is used by the Hyperty Messaging Framework to correctly route messages required to support the data synchronisation, via the MiniBUS component. When a new Data Object (Reporter or Observed) is created, the Syncher will add listeners in the MiniBus to receive messages targeting the ObjectURL. This means, the Syncher is the end-point associated to ObjectURL and not the Hyperty Instance.

According to the Reporter-Observer pattern, there are two types of DataObjects that each Syncher can manage:

DataObjectReporter - provides functions to handle DataObjects as a Reporter i.e. the data that is written in the object by the DataObject owner, is immediately propagated to all observers. It also handles requests from other Hyperty instance to subscribe (ie request to be an Observer) or to read the Data Object.

DataObjectObserver - provides functions to handle DataObjects as a Observer i.e. it handles a "copy" of the original Data Object which is updated as soon as the Reporter changes. Changes on the DataObject are notified to the Hyperty Instance Observers.

In addition, DataObjects can be SyncObjectParents with collections of DataObjectChild. Each collection is called DataObjectChildren. Either Reporter (DataObjectReporter) or Observers (DataObjectObserver) can create DataObjectChilds in a certain children collection (addChild() function).

##### Syncher API

This is the main class that manages the creation of Data Objects. It is a singleton i.e. only one instance is available per Hyperty instance. It's the owner of all kind of data objects that can be synchronised by the Syncher including Reported Objects (DataObjectReporter) and Observed Objects (DataObjectObserver).

##### Properties

* owner: HypertyURL of Syncher's Hyperty instance owner
* observers: [DataObjectObserver] Array of Observed Objects
* reporters: [DataObjectReporter] Array of Reported Objects

##### Methods

**constructor**

constructor(owner: HypertyURL, bus: MiniBus, config: Config)

Syncher is created one per Hyperty. The parameters are basically the same as the ones that are passed to the Hyperty constructor.

* owner: HypertyURL of Syncher's Hyperty instance owner
* bus: MiniBus interface to send and receive message, using postMessage and addListener
* config: Configuration data. The only required field for now is the runtimeURL

**create**

create(schema: SchemaURL, observers?: [HypertyURL], initialData: JSON): Promise<DataObjectReporter>

This Method is used to create objects to be reported i.e. the Hyperty Instance plays the Reporter role. The following parameters are used:

* schema: Hyperty Catalogue URL address that can be used to retrieve the JSON-Schema describing the Data Object schema
* observers: array of Hyperty instances that are invited to be an observer of the new Data Object
* initialData: data that is used to initialise the new Data Object
* return: Promise to a new Reporter. The reporter can be accepted or rejected by the PEP

**subscribe**

subscribe(schema: SchemaURL, url: ObjectURL): Promise<DataObjectObserver>

This Method is used to subscribe objects to be observed i.e. the Hyperty Instance plays the Observer role. The following parameters are used:

* schema: Hyperty Catalogue URL address that can be used to retrieve the JSON-Schema describing the Data Object schema
* url: the URL of the Data Object to be observed
* return: Promise to a new observer if accepted. It's associated with the reporter.

**read**

onChange(filter: string, callback: (event: ChangeEvent) => void): void

Setup the callback to process change events from the associated reporter.

* filter: Filter that identifies the field (separated dot path). Accepts \* at the end for a more unrestricted filtering.
* allback: callback function to receive events

### DataObjectChild

Child objects are returned from the DataObject.addChild. DataObjectChild are created in relation to a pre-existent path on the parent object schema. Child objects can be created from a Reporter or Observer and are shared between them.

##### Properties

* childId: URL address for a child, related to an ObjectURL
* data: JSON data for the object

##### Event Handlers

**onResponse**

onResponse(callback: (event: ReponseEvent) => void): void

Setup the callback to process response notifications of the child creates. Responses to DataObject.addChild

* callback: callback function to receive events

**onChange**

onChange(callback: (event: ChangeEvent) => void): void

Setup the callback to process change events from the associated reporter child.

* callback: callback function to receive events

##### Methods, Events and Handlers

Every object have methods, and event handlers to map to a pulling and push scheme. Methods fire actions and Handlers react to actions and respond accordingly. All events listed on the class diagram are intercepted in an event handler. From a functional perspective, methods like (accept, reject, wait, ...) are responses to an action. Since actions are represented by events, it makes sense that responses are directly related to them. Some rules:

* All events are inherited from the Event interface
* All handlers have method signature of "on<classifier>(..., callback)"

##### SyncStatus

It is used to get and control the status of a DataObject (local, remote, reporter or observer). The interface is not yet implemented, documentation should be updated accordingly from the provided implementation behavior.

**TODO** Maybe some kind of state machine diagram is needed to define better all the status, and the actions that activate the status transitions.

###### Properties

status: actual state based on the actions: pause, resume, stop, ...

###### Methods

pause: should pause the synchronization process, pause the mission of update messages between the reporter/observer link. resume: resume the synchronization process from a pause action. stop: probably the same as unsubscribe, so maybe this method is outdated.

##### SyncSubscription

A reference to a remote observer/subscription, associated to a HypertyURL.

###### Properties

url: HypertyURL of the observer.

#### Discovery and Identity Manager library

Hyperty discovery is performed through the Runtime Registry which returns the Hyperty Registry entry containing its Hyperty Runtime URL, the P2P Handler Stub instance URL and the catalogue URL of P2P Requester Stub. Identities are managed empoying a token-based access. A full dynamic view for discovery and identity management are provided [here](https://github.com/reTHINK-project/specs/tree/master/dynamic-view/discovery) and [here](https://github.com/reTHINK-project/specs/tree/master/dynamic-view/identity-management).

#### Runtime Capability Manager

The Runtime Capability Manager handles a JSON data object containing the capabilities of the Hyperty Runtime that will be used to take the most appropriate decisions according to Runtime Capabilities e.g. to select the most appropriate Protostub or Hyperty to be deployed in the runtime.

#### QoS interface and LHCB library.

An initial description of the QoS support is provided [here](https://github.com/reTHINK-project/specs/tree/master/qos). A full specification of the QoS framework will be given in the dedicated QoS deliverable.

# New Features specification

## Runtime Trust Management Specification (Ricardo Chaves/Nuno)

## P2P Protofly Specification (Paulo)

## QoS Control specification (Marc)

## Multiparty WebRTC Connections specification (Arnaut)

## Interworking with Legacy Services (Anton)

# Conclusions

This report provided a detailed specification of reTHINK Core Framework that comprises the Hyperty Runtime, where Hyperties are executed and the Messaging Node, which supports the messaging communication among Hyperty instances running in different devices.

The core of the document (Chapter 4 and 5) provided a detailed specification of the Hyperty Runtime architecture and the Core Runtime components required to support the execution of Hyperties. The Hyperty Runtime architecture was designed with a security by design approach where different types of components can be executed in isolated sandboxes.

The design of the Hyperty Runtime APIs were validated with detailed descriptions of the main procedures to be supported by the Hyperty Runtime, namely basic procedures (e.g. message routing and Hyperty deployment), Identity Management Procedures (e.g. registration and login of users) and Human to Human communication procedures.

At the end, detailed design was also validated from the data models and interfaces design specified in D2.2 and a few improvements were made.

The reTHINK Core Framework specification is sustained by a comprehensive state of the art research on web runtime and real-time messaging with special attention given to security as well as by an exhaustive work in terms of procurement of existing open source solutions to be used to prototype reTHINK Core Framework components. Taking as input the procurement report, some solutions were selected and some implementation considerations were made. This approach, positions reTHINK prototypes at the forefront of technology with its new functionalities. At the same time it also promotes a rapid and iterative prototyping of reTHINK Core Framework with optimised usage of resources, in order to provide in time, the required components to start the implementation of scenarios in WP5.

The specification will evolve along the implementation phase and it will be also completed with the definition of additional procedures required by the scenarios implementation tasks. Thus, additional procedures are expected to be defined to handle Machine to Machine communication and Human to Machine communication use cases (partial done at the time of this writing), as well as trust and context management procedures.

The Hyperty Runtime APIs were designed to be Developer friendly hiding many complexities from the developer. For example, the complex mechanisms required to manage ID and Access tokens is provided out of the box by the Core Runtime. The same applies to the mechanisms implemented by the Core Runtime to enable out of the box seamless interoperability by using the ProtOFly concept. Developers only have to deal with a couple of functions MessageBUS.postMessage() and the Syncher API. Nevertheless, the Hyperty Service Framework - an Hyperty Software Development Toolkit (SDK) - was also introduced in this report in order to further increase the levels of productivity of Hyperty developers.

The Network Platform specification supporting Specialised Network Services is an ongoing work that will be reported later in D3.4, as originally planned.

# References

[1] - [Barth, A.; Jackson, C.; Reis, C. and Team, Google Chrome. 2008. The Security Architecture of the Chromium Browser.](http://seclab.stanford.edu/websec/chromium/chromium-security-architecture.pdf)

[2] - [Nicholas Carlini, Adrienne Porter Felt, and David Wagner. 2012. An evaluation of the Google Chrome extension security architecture. In Proceedings of the 21st USENIX conference on Security symposium (Security'12). USENIX Association, Berkeley, CA, USA.](http://nicholas.carlini.com/papers/2012_usenix_chromeextensions.pdf)

[3] - [Garcia-Alfaro, J. and Navarro-Arribas, G. 2007. A Survey on Detection Techniques to Prevent Cross-Site Scripting Attacks on Current Web Applications., in Javier Lopez & Bernhard M. Hämmerli, ed., 'CRITIS' , Springer, , pp. 287-298 .](http://eprints.uoc.edu/research/bitstream/10363/605/1/JGA01.pdf)

[4] - [Scott, D. and Sharp, R. Abstracting application-level web security. 11th Internation Conference on the World Wide Web, pp. 396–407, 2002.](http://rich.recoil.org/publications/websec.pdf)

[5] - [Pietraszeck, T. and Vanden-Berghe, C. Defending against injection attacks through context-sensitive string evaluation. Recent Advances in Intrusion Detection (RAID 2005), pp.124– 145, 2005.](http://tadek.pietraszek.org/publications/pietraszek05_defending.pdf)

[6] - [Kirda, E., Kruegel, C., Vigna, G., and Jovanovic, N. Noxes: A client-side solution for mitigating cross-site scripting attacks. 21st ACM Symposium on Applied Computing, 2006.](https://iseclab.org/papers/noxes.pdf)

[7] - [Ismail, O., Etoh, M., Kadobayashi, Y., and Yamaguchi, S. A Proposal and Implementation of Automatic Detection/Collection System for Cross-Site Scripting Vulnerability. 18th Int. Conf. on Advanced Information Networking and Applications (AINA 2004), 2004.](http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1283902&abstractAccess=no&userType=instima)

[8] - [Hallaraker, O. and Vigna, G. Detecting Malicious JavaScript Code in Mozilla. 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS’05), pp.85–94, 2005.](http://www.cs.ucsb.edu/~vigna/publications/2005_hallaraker_vigna_ICECCS05.pdf)

[9] - [Jovanovic, N., Kruegel, C., and Kirda, E. Precise alias analysis for static detection of web application vulnerabilities. 2006 Workshop on Programming Languages and Analysis for Security, pp. 27–36, USA, 2006.](https://iseclab.org/papers/pixy2.pdf)

[10] - [Jim, T., Swamy, N., Hicks M. Defeating Script Injection Attacks with Browser-Enforced Embedded Policies. International World Wide Web Conferencem, WWW 2007, May 2007.](http://www2007.org/papers/paper595.pdf)

[11] - [Uwe Hansmann, Martin S. Nicklous, Frank Seliger, and Thomas Schaeck. 1999. Smart Card Application Development Using Java (1st ed.). Springer-Verlag New York, Inc., Secaucus, NJ, USA.](http://dl.acm.org/citation.cfm?id=555354)

[12] - [Pascal Urien. Cloud of Secure Elements Perspectives for Mobile and Cloud Applications Security. IEEE Conference on Communications and Network Security 2013 - Poster Session](http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6682733)

[13] - [Wojciech Mostowski and Erik Poll. 2008. Malicious Code on Java Card Smartcards: Attacks and Countermeasures. In Proceedings of the 8th IFIP WG 8.8/11.2 international conference on Smart Card Research and Advanced Applications (CARDIS '08), Gilles Grimaud and François-Xavier Standaert (Eds.). Springer-Verlag, Berlin, Heidelberg, 1-16. DOI=10.1007/978-3-540-85893-5\_1 http://dx.doi.org/10.1007/978-3-540-85893-5\_1](http://www.cs.ru.nl/E.Poll/papers/cardis08.pdf)

[14] - [Ankur Taly, Úlfar Erlingsson, John C. Mitchell, Mark S. Miller, and Jasvir Nagra. 2011. Automated Analysis of Security-Critical JavaScript APIs. In Proceedings of the 2011 IEEE Symposium on Security and Privacy (SP '11). IEEE Computer Society, Washington, DC, USA, 363-378. DOI=10.1109/SP.2011.39 http://dx.doi.org/10.1109/SP.2011.39](http://www-cs-students.stanford.edu/~ataly/Papers/sp11.pdf)

[15] - Deliverable D2.2 “Data Models and Interface Specification of the Framework ”, 30-08-2015

[16] - http://w3c.github.io/WebRTC-pc/

[17] - http://w3c.github.io/mediacapture-main/

[18] - http://www.WebRTC.org/

[19] - http://www.openwebrtc.org/

[20] - http://gstreamer.freedesktop.org/

[21] - https://developers.google.com/v8/

[22] - https://Node.js.org/en/

[23] - https://www.docker.com/

[24] - https://www.mozilla.org/en-US/firefox/os/2.0/

[25] - https://wiki.mozilla.org/WebAPI - Firefox Web-API status. (Last Update March 2015)

[26] - https://jitsi.org/Projects/JitsiVideobridge

[27] - http://xmpp.org/

[28] - http://www.kurento.org/

[29] - https://janus.conf.meetecho.com/

[30] - [Alessandro Amirante, Tobia Castaldi, Lorenzo Miniero, Simon Pietro Romano. 2015. Performance analysis of the Janus WebRTC gateway. In Proceedings of the 1st Workshop on All-Web Real-Time Systems](http://dl.acm.org/citation.cfm?id=2749223)

[31] - [Janus: a general purpose WebRTC gateway](http://www.rtc-conference.com/wp-content/uploads/gravity_forms/2-2f7a537445fa703985ab4d2372ac42ca/2014/09/Romano_Janus.pdf)

[32] - P. Chainho, et Al, FP7 Open Lab Deliverable D4.15, WONDER Assessment Report, April 2014

[33] - Paulo Chainho, Kay Haensge, Steffen Druesedow, Michael Maruscheke, “Signalling-On-the-fly: SigOfly, WebRTC Interoperability testbed in contradictive Deployement Scenarios”, Proc. 18th Int’l Conf. Intelligence in Next Generation Networks (ICIN), 2015.

[34] - https://github.com/hypercomm/wonder/wiki/Signalling-on-the-fly

[35] - https://raw.githack.com/hypercomm/wonder/master/docs/api/index.html

[36] - https://github.com/hypercomm/wonder/tree/master/src/libs

[37] - https://raw.githack.com/hypercomm/wonder/master/docs/api/symbols/MessagingStub.html

[38] Deliverable D2.1 “Framework Architecture Definition”, 31-07-2015.

[39] - [Meteor](http://docs.meteor.com/#/full/quickstart)

[40] - [Cookbook MVC](https://github.com/awatson1978/meteor-cookbook/blob/master/cookbook/model-view-controller.md)

[41] - [Meteorpedia](http://www.meteorpedia.com/read/Why_Meteor)

[42] - [AngularJS vs. Backbone.js vs. Ember.js](https://www.airpair.com/js/JavaScript-framework-comparison)

[43] - [Why Meteor](http://www.meteorpedia.com/read/Why_Meteor)

[44] - [Most Popular JavaScript Frameworks 2015](http://www.improgrammer.net/most-popular-JavaScript-frameworks-2015/)

[45] - [Peering through WebRTC with SocketPeer](https://hacks.mozilla.org/2015/04/peering-through-the-WebRTC-fog-with-socketpeer/)

[46] - [Web Components](http://www.w3.org/wiki/WebComponents/)

[47] - TURN rfc, https://tools.ietf.org/html/rfc5766

[48] - STUN rfc, https://tools.ietf.org/html/rfc5389

[49] - IETF TRAM, https://datatracker.ietf.org/wg/tram/documents/

[50] - coturn, https://github.com/coturn/coturn

[51] - [AngularJS](https://angularjs.org/)

[52] - [BackboneJS](http://backbonejs.org/)

[53] - [StapesJS](https://hay.github.io/stapes/)

[54] - http://en.wikipedia.org/wiki/Real-time\_database

[55] - http://www.leggetter.co.uk/real-time-web-technologies-guide/

[56] - http://www.matrix.org/

[57] - http://vertx.io/vertx2/

[58] - http://vertx.io/

[59] - https://www.rabbitmq.com/

[60] - http://www.amqp.org/

[61] - http://mqtt.org/

[62] - http://www.psyced.org/

[63] - http://redis.io/

[64] - https://xmpp.org/xmpp-software/libraries/

[65] - http://zeromq.org/

[66] http://www.w3.org/2012/sysapps/app-lifecycle/

[67] https://lists.w3.org/Archives/Public/public-sysapps/2015Apr/0001.html

[68] https://www.w3.org/community/trustperms/

[69] https://whatwg.org/

[70] http://www.w3.org/TR/CSP2/

[71] https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing\_Content\_Security\_Policy

[72] http://en.wikipedia.org/wiki/Content\_Security\_Policy

[73] http://w3c.github.io/push-api/

[74] http://thenewdialtone.com/WebRTC-browser-push-notification/

[75] http://datatracker.ietf.org/doc/draft-thomson-webpush-protocol/?include\_text=1

[76] http://www.w3.org/TR/workers/

[77] https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorker\_API

[78] https://github.com/slightlyoff/ServiceWorker/blob/master/explainer.md

[79] http://www.w3.org/TR/service-workers/

[80] https://jakearchibald.github.io/isserviceworkerready/

[81] https://http2.github.io/

[82] RFC7540 - Hypertext Transfer Protocol version 2

[83] Object RTC - http://ortc.org/

[84] draft-tsvwg-quic-protocol-01 : QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2 - http://tools.ietf.org/html/draft-tsvwg-quic-protocol-01

[85] http://www.w3.org/2012/sysapps/app-lifecycle/

[86] https://whatwg.org/

[87] https://lists.w3.org/Archives/Public/public-sysapps/2015Apr/0001.html

[88] http://www.w3.org/TR/CSP2/

[89] http://w3c.github.io/push-api/

[90] http://datatracker.ietf.org/doc/draft-thomson-webpush-protocol/?include\_text=1

[91] http://www.w3.org/TR/workers/

[92] http://www.w3.org/TR/service-workers/

[93] http://www.w3.org/2011/04/webrtc/

[94] https://w3c.github.io/webrtc-pc/

[95] http://www.w3.org/TR/mediacapture-streams/

[96] http://www.w3.org/TR/mediastream-recording/

[97] http://www.w3.org/TR/image-capture/

[98] http://w3c.github.io/mediacapture-depth/

[99] http://www.w3.org/TR/mediacapture-fromelement/

[100] http://www.w3.org/TR/audio-output/

[101] http://www.w3.org/TR/webrtc-stats/

[102] http://www.w3.org/TR/screen-capture/

[103] http://apirtc.com/api-docs/

[104] http://www.quobis.com/index.php?option=com\_content&task=view&id=285&Itemid=208

[105] http://passportjs.org/

[106] http://www.html5rocks.com/en/tutorials/es7/observe/

[107] http://w3c.github.io/WebRTC-pc/#identity

[108] https://nodesecurity.io/resources

[109] Deliverable D4.1, “Management and Security features specifications”, 30-09-2015.

[110] http://requirejs.org/

[111] https://github.com/crosswalk-project/crosswalk

[112] http://cordova.apache.org/

[113] https://github.com/eface2face/cordova-plugin-iosrtc

[114] https://github.com/alongubkin/phonertc

[115] http://beagleboard.org/bone

[116] http://weworkweplay.com/play/raspberry-pi-Node.js/

[117] http://beagleboard.org/Support/BoneScript

[118] http://www.armhf.com/node-js-for-the-beaglebone-black/

[119] http://www.armhf.com/download/

[120] http://cylonjs.com/

[121] http://gf3.github.io/sandbox/

[122] https://github.com/telefonicaid/lwm2m-node-lib

[123] http://samsung.github.io/iotjs/

[124] http://nuttx.org/

[125] https://github.com/pac4j/vertx-pac4j

[126] http://redis.io/topics/pubsub

[127] https://github.com/NodeRedis/node\_redis)

[128] http://expressjs.com/

[129] https://www.npmjs.com/package/node-sandbox