

**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: | 34 |
| 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](#_Toc460336767)

[List of authors 5](#_Toc460336768)

[Table of Contents 6](#_Toc460336769)

[Abbreviations 7](#_Toc460336770)

[1 Introduction 9](#_Toc460336771)

[1.1 Objectives and Overview 9](#_Toc460336772)

[1.2 Structure 11](#_Toc460336773)

[2 Core Framework Specification update 12](#_Toc460336774)

[2.1 Runtime Specification Update (Paulo) 12](#_Toc460336775)

[2.1.1 Runtime Architecture 12](#_Toc460336776)

[2.1.2 Runtime Main Procedures 19](#_Toc460336777)

[2.1.3 Runtime Implementation Considerations 47](#_Toc460336778)

[2.2 Messaging Framework Specification Update (Steffen) 54](#_Toc460336779)

[2.2.1 Messaging Node Architecture 54](#_Toc460336780)

[2.2.2 Vertx Specification 56](#_Toc460336781)

[2.2.3 Node.js based Messaging Node Specification 58](#_Toc460336782)

[2.2.4 Matrix.org based Messaging Node Specification 62](#_Toc460336783)

[3 New Features specification 66](#_Toc460336784)

[3.1 Runtime Trust Management Specification (Ricardo Chaves/Nuno) 66](#_Toc460336785)

[3.2 P2P Protofly Specification (Paulo) 66](#_Toc460336786)

[3.3 QoS Control specification (Marc) 66](#_Toc460336787)

[3.4 Multiparty WebRTC Connections specification (Arnaut) 66](#_Toc460336788)

[3.5 Interworking with Legacy Services (Anton) 66](#_Toc460336789)

[4 Conclusions 67](#_Toc460336790)

[5 References 68](#_Toc460336791)

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.

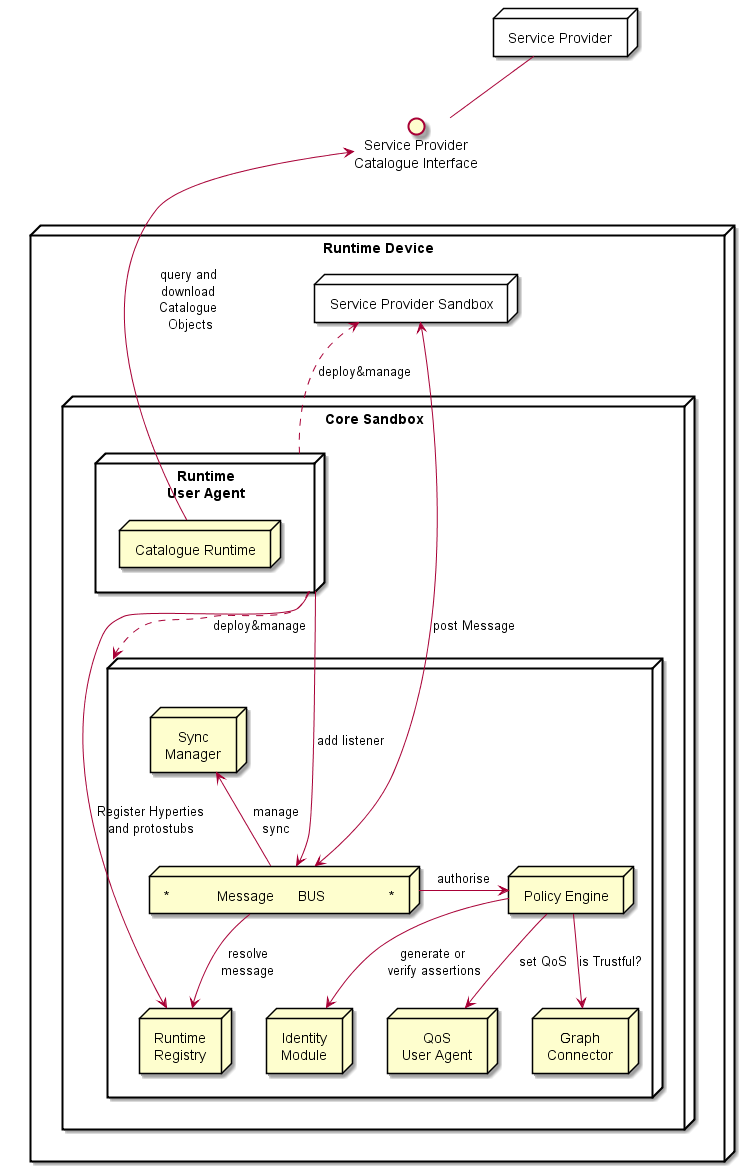


Figure 1 – Runtime Architecture

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.

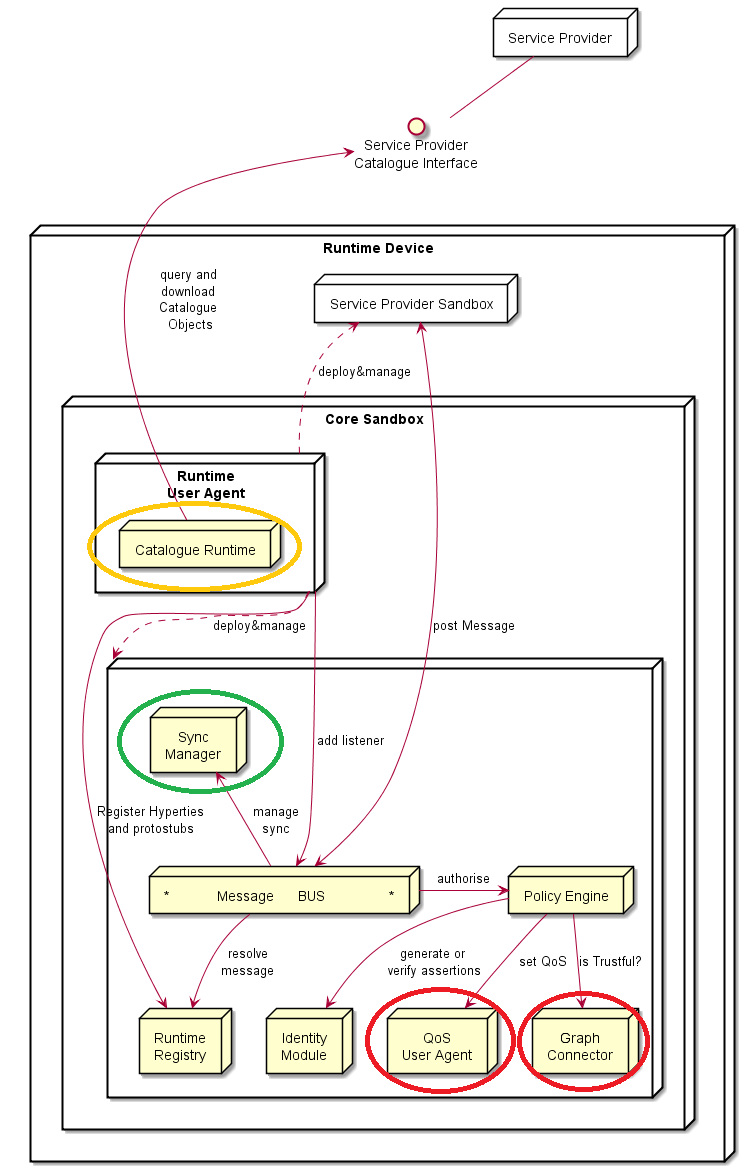


Figure 2 - Changes in the Hyperty Runtime Core

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)

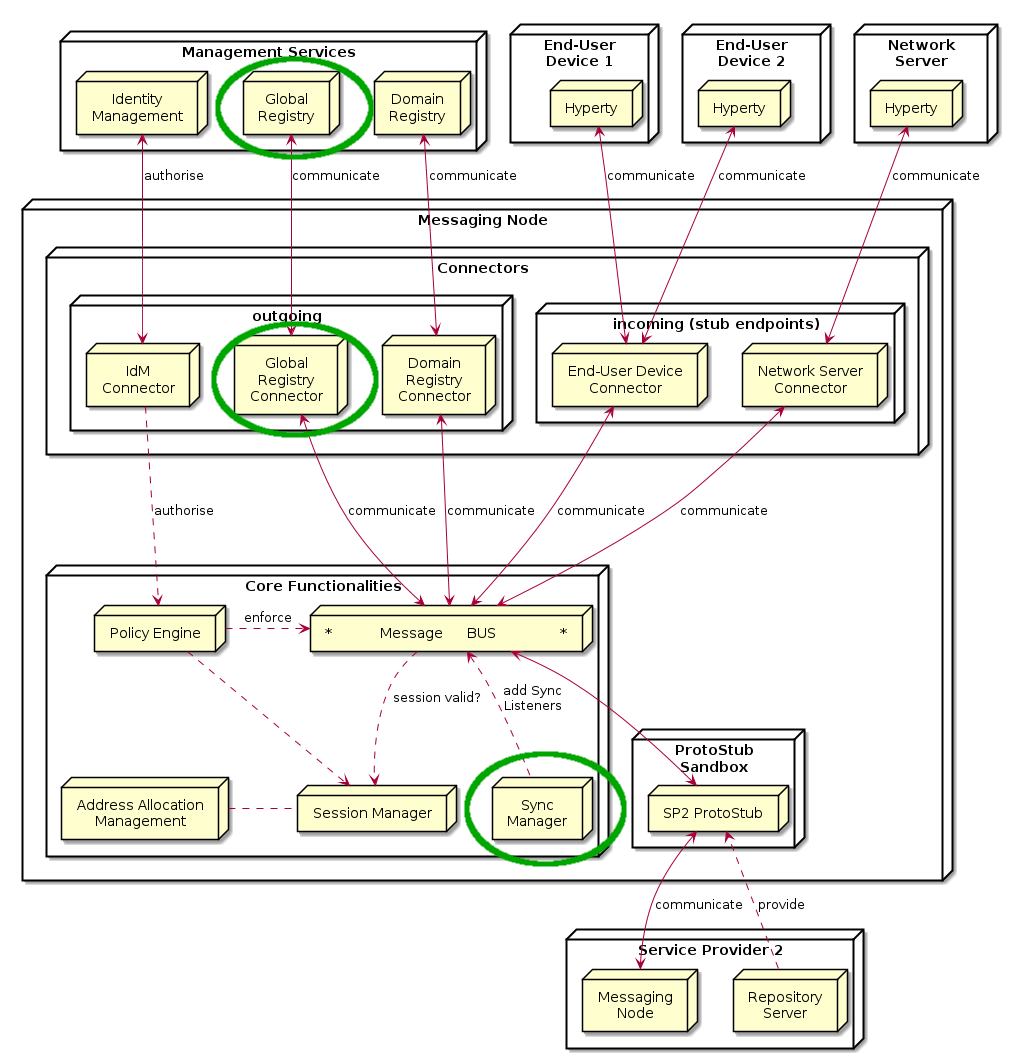
This Chapter contains the functional design of the Messaging Node Architecture which enables messaging communication among Hyperty instances running in different Runtime devices.

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. Instead, a more detailed specification is provided for each messaging solution selected during the procurement activity namely for Vertx.io, Node.js and Matrix.

### Messaging Node Architecture

The Messaging Node functional architecture is presented in the figure below and it comprises three main types of functionalities including the Core Functionalities, Connectors and Protocol Stubs.

Compared to the previous version two additional components have been added to the Messaging Node Architecture, a Sync Manager as well as a connector for the Global Registry.

  
Figure 3: Messaging Node Architecture

#### Core Functionalities

##### Message BUS

The Message BUS routes messages to internal Messaging Node components and external elements by using Connectors or Protocol Stubs. It supports different communication patterns including publish/subscribe communication.

##### Policy Engine

The Policy Engine provides Policy decision and Policy Enforcement functionalities at Domain level for incoming and outgoing messages in cooperation with authentication and authorisation provided by Identity Management functionalities. It also provides authorisation / access control to the Message BUS.

##### Session Management

Session Management functionalities are used to control messaging connections to service provider back-end services. For example, when user turns-on the device and connects to its domain, providing credentials as required by Identity Management functionalities. In general, each message should contain a valid token that is generated when the client connects to the Messaging Node. It also manages the registry of protocol stubs and connectors supported by the Messaging Nodes to support the routing of messages to these components.

##### Address Allocation Management

The Address Allocation Management functionality handles the allocation of messaging addresses to instances of Hyperties and Synchronization Data Objects in cooperation with Session Management. These addresses are valid for at least the lifetime of a session. They are used by the Sync Manager and Message BUS to take routing decisions.

The specification of the messages to manage address allocations can be found at “[Address-allocation-messages](https://github.com/reTHINK-project/dev-service-framework/blob/master/docs/specs/messages/address-allocation-messages.md)” in github.

It is also responsible for the allocation of messaging addresses to foreign Hyperty Instances i.e. Hyperty Instances that are provided from external domains but that use the protofly concept to interact with Hyperty Instances served by this Messaging Node. For example, if the Messaging Node is implemented by core IMS or a simple SIP Proxy/SIP Registry, it requires the management of a pool of SIP addresses to be allocated to clients that have no account in the IMS HSS or in the SIP registry.

##### Sync Manager

The Message Node Sync Manager is in charge of handling requests from Runtime Sync Managers in order to setup the Data Sync Stream routing path in the Message Node.

The specification of the messages to manage routing pathes for data object synchronization messages can be found at “[Data-sync-messages](https://github.com/reTHINK-project/dev-service-framework/blob/master/docs/specs/messages/data-sync-messages.md)” in github.

##### Protocol Stub

In special situations e.g. when the download of external software (protocol stubs) into end-user devices is not allowed or not possible due to technical constraints, it should be possible to have interoperability between Messaging Nodes from different domains, by using the protofly concept between the Messaging Nodes.

Thus, a Protocol Stack to be used to communicate with another Messaging Node can be deployed into the runtime of the Messaging node.

#### Connectors

Connectors implements protocol stacks used to interoperate with external elements . In general there are connectors for outgoing access to components inside or outside the own domain and on the other hand endpoints listening for incoming connections from external entities, like hyperty runtimes on end-user/network- devices.

All types of Connectors can be supported by using protocol on-the-fly concept, giving more flexibility for the integration of the Messaging Node in the Service Provider infra-structure.

##### IdM Connector

The IdM connector provides access to the domains Identity Manager. The IdM functionalities support the Session Manager for a general Access Control and the Policy Manager for the validation of identity tokens in messages and the enforcement of routing policies.

##### Domain Registry Connector

The Domain Registry Connector handles messages for the registration, un-registration and lookup of Hyperties and Data Objects in the domain registry. The specification of these messages can be found at “[Registration messages](https://github.com/reTHINK-project/dev-service-framework/blob/master/docs/specs/messages/registration-messages.md)” in github. The Domain Registry Connector mainly acts as a “relay” between the hyperty runtimes and the domain registry. It does not actively process the messages and responses. This connector is mandatory because the direct access to the Domain registry from hyperty runtimes should be restricted.

##### Global Registry Connector

The role of the Global Registry Connector is comparable to the connector for the Domain Registry. It acts as a relay between the hyperty runtimes and the Global Registry. This Connector is optional. It might be required in cases where the runtime itself might not be able to establish an own connection to the Global registry. In such cases it can use the Connector running on the MN of its home-domain to access it.

The specification of the messages for the interaction with the global registry can be found at “[Global Registry messages](https://github.com/reTHINK-project/dev-service-framework/blob/master/docs/specs/messages/global-registry-messages.md)” in github.

##### End-user Device / Network Server Connectors

These Connectors provide the “server-side” for connections that are initiated by protocol stubs running inside of Hyperty runtimes. These runtimes can either be running on end-user devices (e.g. in a browser or stand-alone) or on network-server devices, for example on an embedded system that supports an IoT use case.

A simple technical example for such a connector is a Websocket server that waits for connection requests from externally deployed stubs and handles them. The types of required server-side connectors correlates to the types of stubs that the MN needs to support. If a stub, for instance, needs to establish a REST like communication with the MN than the MN must operate a connector that implements the REST server endpoint.

### Vertx Specification

#### Core Functionalities

* Main objective of core functions are to **connect**, **intercept**, **process**, **filter** and **deliver** messages. Messages are JSON objects that should have 2 blocks, HEADER and BODY, and are processed from different components of core. Inbound messages should be intercepted and processed in the Pipeline before deliver in to the Message Bus.
* Pipeline components will implement a simple interface that we can reuse from io.vertx.core.Handler<E> replacing E with a PipelineContext object. Using the Vert.x Handler has the advantage to be compatible with io.vertx.ext.web.Router, that can be a replacement for the Pipeline.
* Outbound messages should be processed in a Pub/Sub system. If message BODY block are for CRUD operations, there should be a Pub/Sub protocol for object/model subscriptions, where this should be processed. The address scheme of the vertx EventBus is not enough for this functionality. We need to control the Pub/Sub functionality better than what vertx provides with the address scheme! Hyperties need to subscribe to objects/collections not just addresses.

##### Pipeline

Pipeline functionality is to **intercept**, **process** and **filter**. The Pipeline configuration can reflect the concept of activity diagrams, controlling the path flow of the message that is dependent of the message type. This concept is generic enough to contemplate different message flows in the future. This is a new component to be developed which is similar to Vert.x Router but without the URL addressing scheme. The io.vertx.ext.web.Router class could be a possible candidate for Pipeline functionalities, however the Router is hard coded to work with HTTP protocols, and there is no need for static configurations of routing schemes. The alternative is to implement a simple Pipeline system instead of using the Router, fewer dependencies and better decoupled from the protocol.

##### Session Management

Session Management is one of the Pipeline handlers that will intercept messages and verify the sessionID. A session instance is linked to a connection resource (WebSocket, SockJS) if authorized. Every message header is intercepted, session token is verified and if exist, a "user" or other identification URL is replaced in HEADER. The JSON object is forwarded to "Access Control" handler.

##### Address Allocation Management

This is not a Pipeline handler (it doesn't process messages), but it's used by the "Session Management" to allocate Hyperty identification URL's that will be linked to a Session when the Hyperty is connected. This will be used to translate Hyperty and URL address into the correspondent Connector Resource.

##### Access Control

This handler is able to analyze the HEADER (identification URL from "Session Management") and BODY blocks and decide if the message should be forwarded to the "Message Bus" or denied. There is a possibility to add a rule engine in this step, but it's not specified for now, what kind of rule engine.

##### Message BUS

Main objective of the MB is to **deliver** the message, being independent of the cluster node that has the connection to the destination. Vert.x EventBus can be used directly for the Message Bus component. Important headers of the original JSON (like the identification URL) must be forwarded to io.vertx.core.eventbus.Message.headers() map.

#### Protocol Stub Sandbox

The Protocol Stub sandbox will be managed by a ProtocolStubManager class that loads, registers and removes protocol stubs on request. If ProtoStubs are in JavaScript, the sandbox model could be implemented using the java NashornScriptEngineFactory and controlling the available API's with ClassFilter.

#### Connectors

##### End User Device Connector

The aim of this Connector is to enable interaction with Hyperty instances running in the end-user device. This component will need to interact somehow with the Protocol Stub sandbox to achieve this, since the communication protocol will not be standardized. It will need to implement a simple protocol for sending and receiving requests. In itself it is not responsible for processing communication requests, which is left to the protocol stack. It merely forwards messages to and from the Hyperty instance.

##### Network Server Connector

The aim of this Connector is to enable interaction with Hyperty instances running in a network server. This component will need to interact somehow with the Protocol Stub sandbox to achieve this, since the communication protocol will not be standardized. It will need to implement a simple protocol for sending and receiving requests. In itself it is not responsible for processing communication requests, which is left to the protocol stack. It merely forwards messages to and from the Network server.

##### Registry Connector

The Registry provides an interface for registration and deregistration of Hyperty instances, as well as for keeping the published information up to date. For each Hyperty instance, the Registry stores data (hyperty location, type, description, start-time, presence information of user) that enables other applications to contact it. The implementation of the Registry service is thought to be basically a distributed database. It will provide service interfaces for CRUD operations to allow users to retrieve data for a given GUID, publish (i.e. create, update, and delete) their own information on the ring. To verify authenticity and integrity of the published data, digital signatures will be applied. The Connector will expose the available interfaces of the Registry Services to users of managing Hyperty instances. This will have to be implemented as a standalone application with an adapter interface to the Event Bus for encoding and decoding messages and deployed as a fat executable JAR which contains all the dependencies it needs to run on Vert.x.

##### IdM Connector

This Connector is to provide functionalities for interacting with the remote Identity Management Functionalities. As hyperties need to be linked to an end-user identity when downloaded and instantiated on a device, an Identity Module should be present on the device. This module at minimum should act as an identity selector for the user and as a secure local repository for identity tokens provided by IdPs

If the connector is thought to provide authentication and authorisation, Vert.x offers Auth APIs (Common, JDBC, JWT and Shiro).

There is also a library for authentication and discorvery, [vertx-pac4j](https://github.com/pac4j/vertx-pac4j) [125]. This vertx module provides multiple authentication mechanisms (OAuh, CAS, HTTP, OpenID, SAML2.0 and OpenIDConnect) for different IdPs.

### Node.js based Messaging Node Specification

For each [functional block](file:///C:\Projectos\reTHINK\WP3\git\core-framework\docs\deliverables\d31\msg-node-architecture.md) existing Node.js modules were identified, which can be either reused or extended.

#### Core Functionalities

This section attempts to match the functional blocks of the Messaging Node architecture to features and functional blocks of the Node.js and Redis architecture.

##### Message BUS

The message bus can be implemented with [Redis](http://redis.io/) [63]. Redis is an open source (BSD licensed), in-memory data structure store, used as database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.

###### Usage of Redis with Node.js

[Redis integrate a PUB/SUB mechanism](http://redis.io/topics/pubsub) [126]:

SUBSCRIBE, UNSUBSCRIBE and PUBLISH implement the Publish/Subscribe messaging paradigm where (citing Wikipedia) senders (publishers) are not programmed to send their messages to specific receivers (subscribers). Rather, published messages are characterized into channels, without knowledge of what (if any) subscribers there may be. Subscribers express interest in one or more channels, and only receive messages that are of interest, without knowledge of what (if any) publishers there are. This decoupling of publishers and subscribers can allow for greater scalability and a more dynamic network topology.

Redis can be used to add scalability/redundancy to the Messaging Node as the different components of the architecture can easily be splited on different servers. This Pub/Sub mechanism is simple to use and It can also facilitate the development and the integration of new connectors

Communication between Node.js and Redis can be managed by a Nodes.Js Redis [client module](https://github.com/NodeRedis/node_redis) [127]. Redis instance can be a single instance or a Redis cluster.

##### Access Control

User connection to Node.js connectors can be authenticated on the Node.js module. Socket.io integrate a way to authenticate incoming request, authenication component will have to be develop on Node.js connectors.

This component is able to analyse HEADER (identification URL from "Session Management") and DATA blocks and decide if the message should be forwarded to the "Message Bus" or denied.

[PassportJs](http://passportjs.org/) [105], which is an interesting middleware that could enable us to add third party authentication should be used. An authentication can also be done between Node.js and Redis.

##### Session Management

For a complete session management on Node.js, it will be interesting to use [ExpressJS](http://expressjs.com/) [128] which is a Web framework for Node.js.

##### Address Allocation Management

This component will have to be developed on a Node.js server

##### Protocol Stub & Connectors

Connectors will be Node.js processes to be developed. The protoStub/protoFly mechanism Goal can be used to facilitate the integration with other servers.

###### IdM Connector

This Connector is to provide functionalities for interacting with the remote Identity Management Functionalities. Node.js can easily interact with OAuth servers in order to authenticate and authorize users.

The authentication against the Identity Provider has to be done at the beginning.

###### Registry Connector

The implementation of this Connector requires further study.

###### End-User Device Connector

Communication between Users and Node.js can be managed by socket.io Socket.io is a popular Node.js library to handle connections at application level. It can use Websocket and it falls back to HTTP automatically if WS connectivity is not possible.

###### Network Server Connector

The implementation of this Connector requires further study.

###### Node Sandbox framework

[Node-sandbox](https://www.npmjs.com/package/node-sandbox) module [129] allows to run untrusted code outside of the main node process. The code can be interfaced with code running in the sandbox via RPC (or any library that works over the node Stream API).

#### Node.js implementation architecture

**Architecture : Node.js and Redis :**

Here is the description of the architecture with Redis :

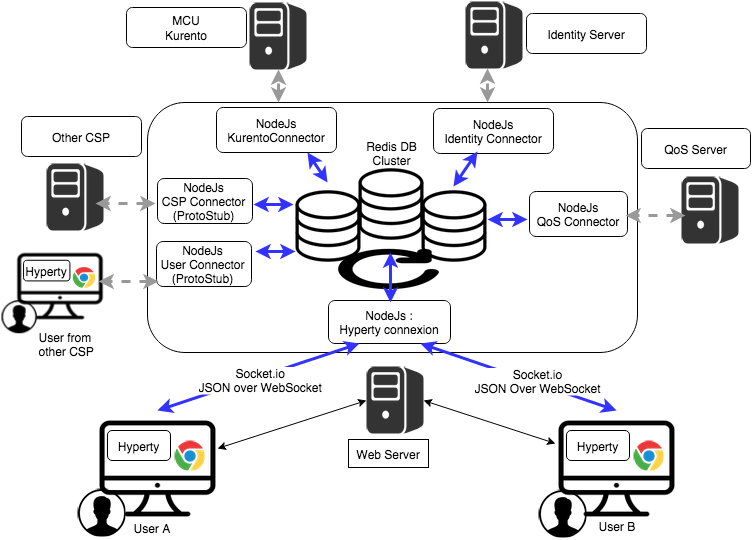


Figure 53: Messaging Node implementation with Node.Js and Redis

**Architecture : Integration in ReThink :**

Following architecture shows the target integration with the different components of the ReTHINK projet:

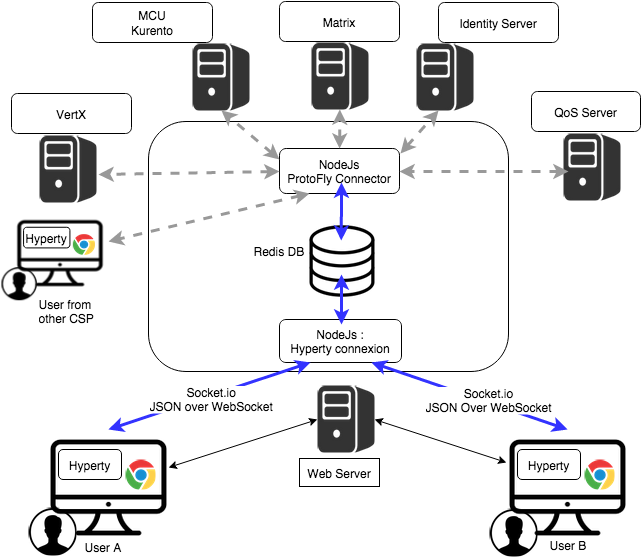


Figure 54: Integration of Node.Js based Messaging Node implementation with reTHINK

**Architecture : Integration in ReThink with Actors:**

Following architecture shows the actors in the architecture to understand the decomposition of work to be done and the interaction with other partners :

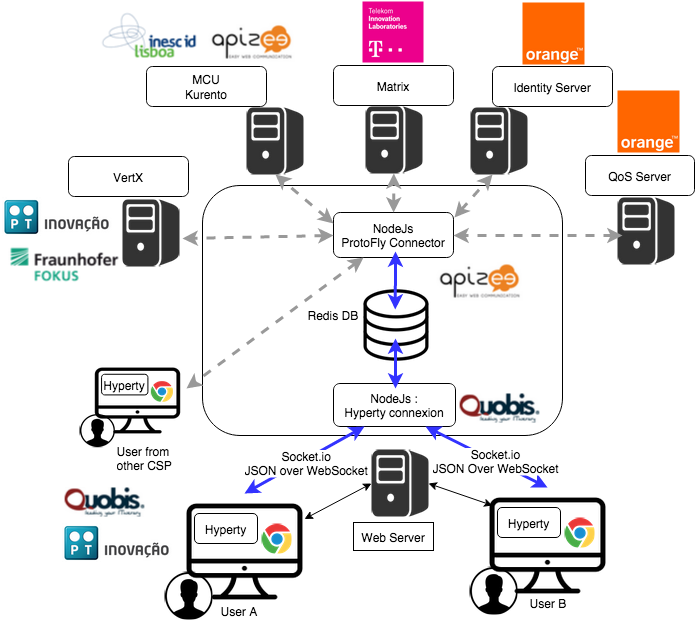


Figure 55: Integration of Node.Js based Messaging Node implementation with reTHINK partners

### Matrix.org based Messaging Node Specification

This section matches the requirements for the functional blocks of the Messaging Node architecture to features and functional blocks of the matrix.org architecture. Functional gaps are identified and proposals for extensions to the standard Matrix.org Homeserver are made in order to fill these gaps.

#### Protocol Stub and Connectors

Protocol Stubs and Connectors are means to make a Messaging Node interoperable with foreign signalling protocols.

A Protocol Stub is the core entity of the Protocol-on-the-fly concept. It is a downloadable piece of JavaScript code that is executed in the client's runtime and performs the required adaptations on the messaging protocol. In a Protocol-on-the-fly based communication relation there is always one side in the client role (i.e. the side that downloads the stub) and the other side in the server role (the side that the stub connects to).

A Messaging Node has to support both operation modes in order to provide full bi-directional interoperability.

##### Matrix as Protocol-on-the-fly client

The most appropriate feature that Matrix provides for this purpose is the concept of "Application Services". An Application Service is an implementation of a special service function that can be attached to a Homeserver (HS). Based on certain patterns, messages are filtered and forwarded to the Application Service that performs application specific tasks. This concept is quite comparable to Application Servers in the IP Multimedia Subsystem (IMS) framework. It can, for example, be used for aggregation and accounting purposes, but also for the implementation of "breakout" communication to other types of messaging infrastructures it is well suited.

A special dedicated Application Service is proposed that will implement a Protocol-on-the-fly client engine to allow the "breakout" to different signalling domains that provide a Protocol-on-the-fly stub. Such an Application Service will be a very flexible mechanism for interdomain collaboration.

##### Matrix as Protocol-on-the-fly server

In order to support the server role in the Protocol-on-the-fly architecture, a specialized Matrix Protocol Stub needs to be implemented that connects to a Homeserver. Since the Matrix Homeserver has a well-documented API and the Matrix message format allows the transport of arbitrary payload, this implementation should be straight forward. The implementation can make use of the SDK's that are available for Matrix client developers. These SDK's encapsulate a lot of the internal complexity for REST based communication.

##### Connectors in Matrix

Connectors also play the role of protocol adapters, which makes them comparable to protocol stubs. The difference is that they are not downloaded to the Messaging Node clients. Instead they are executed in the scope of the Messaging Node itself. Such Connectors are intended to connect with different "legacy" clients that don't support the Protocol-on-the-fly concept.

Also for the implementation of such connectors the concept of Application Services seems well suited. The matrix.org developer community has implemented this as a proof of concept that connects the Matrix ecosystem with the Internet Relay Chat (IRC) world. Messages that contain a specially prefixed address are filtered out, converted to IRC messages, forwarded to the corresponding IRC client and vice versa. This can be used as pattern for the implementation of additional adapters.

#### Core Functionalities

##### Message Bus

The Message Bus is responsible for the routing of messages to internal Messaging Node components and external elements by using Connectors or Protocol Stubs. This routing shall support different communication patterns including publish/subscribe communication.

These main routing requirements are fulfilled out-of-the-box by standard matrix features. In order to route messages to internal Messaging Node components it will be required to provide such components with virtual identifiers that can be used internally to address them.

##### Access Control

The main task of the access control is to enforce manageable policies to the forwarding of individual messages. For example, a single type of message shall be blocked if a special combination of sender and/or receiver matches.

Matrix.org requires registration/subscription and login of users in order to exchange any messages with other users. These authentication and authorisation methods however always apply to a complete user- and communication session, that means to ALL messages that are exchanged in a session scope. This concept does not provide an access control on a "per message" base.

The matrix developer community already discusses the integration of a "policy service", but so far this integration in not yet specified.

In order to achieve a "per message"-policy enforcement without deeper changes in the matrix core, we propose the introduction of a message proxy as first step of the message flow. This proxy has the task to check the messages and to apply the policies. It would forward messages according to the policies and should reject the rest. A potential bypassing of this proxy must be avoided by appropriate network configurations.

The design of this message proxy component should be closely coordinated with the MessagingStub that is used to connect to this Matrix based Messaging Node, because the proxy will be the first contact point for the stub.

The following figure gives an overview of the intended architecture of the Matrix based Messaging Node.

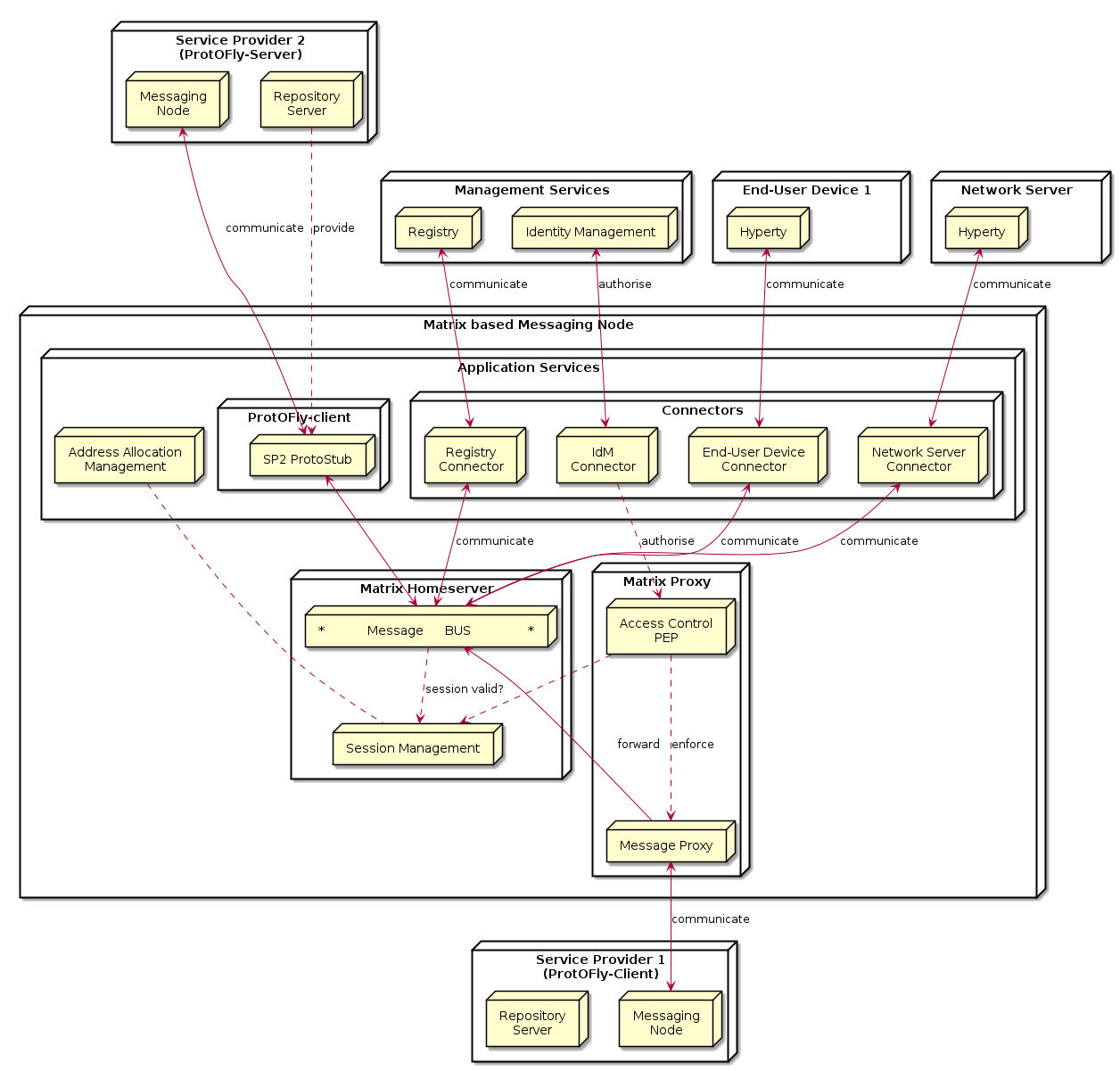


Figure 56: Matrix Messaging Node Architecture

##### Session Management

The requirements regarding session management as described in the Messaging Node architecture can be separated in three aspects which are handled in the following sub-chapters:

* User session control,
* Communication session control, and
* Stub and connector management.

###### User session control

In order to use matrix based messaging users have to be registered/subscribed with a matrix HomeServer. Matrix provides an API for the subscription of new users with their HomeServers. This API can be used to provision accounts also programmatically, when required.

In order to establish a communication session with other peers, users have to pass a login sequence. During this sequence an access token is generated which is valid for this login session. This access token must be present in all sub-sequent requests during this user session. No mandatory authentication methods are specified. This is left as implementation specific for the particular HomeServers. The specification lists following standard methods:

* m.login.password,
* m.login.recaptcha,
* m.login.oauth2,
* m.login.email.identity, and
* m.login.dummy.

The HomeServer Client API provides means to request the supported methods before login.

###### Communication session control

Communication sessions between two or more users require a valid user session. Communication sessions are always based on "rooms". Each room is identified by a unique room-id. Messages are sent to room-ids and not to individual users. Users must explicitly create or join rooms in order to send and receive messages. Some rooms might be open - others may require an invitation by the creator of the room. Rooms are persistent, i.e. they exist also if not all room members are currently logged in. The message history is maintained by the Matrix HomeServers and can be requested by clients.

###### Stub and connector management

Matrix.org provides powerful means to connect, federate, and synchronise Matrix HomeServers from different domains. The resolution of the peer HomeServers connectivity is done via DNS. The message exchange between them is secured by encryption mechanisms.

However - for the interoperability with non-Matrix infrastructures there is no "golden" way. The selected and most appropriate approach is via Application Services, as described before.

The "Stub and connector management" function is responsible for the management of the Application Services that implement the Protocol-on-the-fly clients and the connectors.

##### Address Allocation Management

In order to be addressable each hyperty instance should be treated as an individual client of the Messaging Node that registers with an own identity and needs a login before it can exchange messages. The Messaging Node allocates the identity of a hyperty during the registration/subscription process. The allocated identity serves then as a messaging address for domain internal communication.

External Hyperties from foreign domains (that might use different communication protocols and identifiers) will need an address representation in the Matrix domain that is compatible with the local addressing scheme. The Messaging Node is responsible for the creation and assignment of such transient addresses for domain external entities.

Since we have identified Application Services as the most appropriate way of connecting to other signalling domains, also the management of such virtual transient addresses is in the responsibility of the corresponding Application Service. Each Application Service itself has to maintain an own namespace of virtual users and must be able to operate (send/receive) "on behalf" of such a virtual user.

# 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/" \l "/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