ARTWIN
INDUSTRY & CONSTRUCTION 4.0 SOLUTIONS

Preliminary platform specifications

April, 2020

This project has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under Grant Agreement no 856994.

AUTHORS	
Name	Organisation
Bilal Al Jammal, Laurent Roullet, Lionel Natarianni, Sylvaine Kerboeuf	NBL
Jerôme Royan, Carole Bonan, David Tardivel, Loïc Touraine	BCM
Michal Polic	CTU
Alexander Werlberger	HOL
Nischita Sudharsan, Andreas Hutter	SIE

ARtwin has received funding from the European Union' Horizon 2020 research and innovation programme under Grant Agreement no. 856994.

Project Information

TITLE	ARtwin - An AR cloud and digital twins solution for industry and construction 4.0
DURATION	October 2019 – September 2022 (36 months)
WEBSITE	artwin-project.eu
COORDINATOR	Q-PLAN INTERNATIONAL ADVISORS PC
CONTACT PERSON	Anastasia Matonaki, matonaki@qplan-intl.gr
PROJECT OVERVIEW	ARtwin aims at improving productivity and product quality, by deploying an ARCloud platform that will offer a wide variety of cutting-edge AR-based services to industry and construction 4.0.
CONSORTIUM	1. Q-PLAN INTERNATIONAL ADVISORS PC www.qplan-intl.gr – Greece 2. B-COM http://www.b-com.com – France 3. SIEMENS AKTIENGESELLSCHAFT https://new.siemens.com/global/en.html – Germany 4. CESKE VYSOKE UCENI TECHNICKE V PRAZE http://www.cvut.cz/en - Czech Republic 5. NOKIA BELL LABS FRANCE http://www.bell-labs.com/ – France 6. HOLO-INDUSTRIE 4.0 SOFTWARE GMBH http://www.holo-light.com/contact.html - Germany 7. ARTEFACTO SAS http://www.artefacto-ar.com/en/ – France

ARtwin has received funding from the European Union' Horizon 2020 research and innovation programme under Grant Agreement no. 856994.

Table of Contents

List of figures

List of tables

Executive Summary

The purpose of this document is to:

- propose a preliminary architecture of ARtwin system able to comply with ARtwin functional (services), operational (lifecyles) and performance (measurements) requirements as given by Task 2.1,
- realize a “360° inventory” in target Infrastructures-as-a-Service where the platform will be deployed (embedded cloud, edge cloud, central cloud) and supported components.

The proposed architecture follows *TM Forum* recommendations based on digital platform approach. The first section introduces this approach and shows why this approach is suitable and can be applied to ARtwin. The second section provides some definitions that will be used when talking about the platform, its components and capabilities. It gives an insight on how a platform is built, edited, and deployed. Furthermore, it provides information about the different actors and roles involved in the ARtwin architecture. The third section provides a non-comprehensive list of services, platforms, and environments that will be either developed or used during the project. Finally, section 4 concludes the document.

The architecture proposed in this document will subsequently provide inputs to platforms development and tests in tasks 3.2 and 3.3.

1 Introduction

ARtwin is a very complex system that encompasses several domains of expertise and requires the interaction between multiple actors. Managing such systems with agility is not always trivial. Thus, this section first introduces an architectural approach based on platforms technology and shows its benefits and applicability to ARtwin.

1.1 TM Forum approach

This section introduces the concepts related to platforms and applied by the TM Forum (TMF). As stated in [IG1157], platform architectural concepts address two key agility challenges of the digital transformation journey of service providers and enterprises.

1. Agility in participating in 'Digital Platform based Business Models' through use of reusable concepts and components integrated using exposed external services through Open APIs. This is the TM Forum's Digital Platform Reference Architecture (DPRA), detailed in the next three sub-sections 1.1.1, 1.1.2, and 1.1.3.
2. Agility in internal operations through use of Platform based IT Architecture for OSS/BSS where these Platforms expose management services integrated and orchestrated through Open APIs. This is the TM Forum's Open Digital Architecture (ODA), detailed in 1.1.4.

1.1.1 Why platforms?

Companies like Apple, Facebook and others keep growing thanks to *network effects* of the technology they build. The key of their success is the development of technology-enabled platforms that combine both demand and supply to disrupt existing industry structures. These technology platforms create new ways of consuming goods and services and enable easy access for both producers and consumers. 'Platform-based' business models have proven to be very powerful at exploiting the digital economy.

In the *Platform Revolution* book [PAC16], a platform is defined as "a business based on enabling value-creating interactions between external producers and consumers. The platform provides an open, participative infrastructure for these interactions and sets governance conditions for them. The platform's overarching purpose to consummate matches among users and facilitate the exchange of goods, services, or social currency, thereby enabling value creation for all participants."

A platform business model essentially disaggregates the pipeline model by allowing a business platform to be inserted between the platform producer and platform consumer (Figure 1).

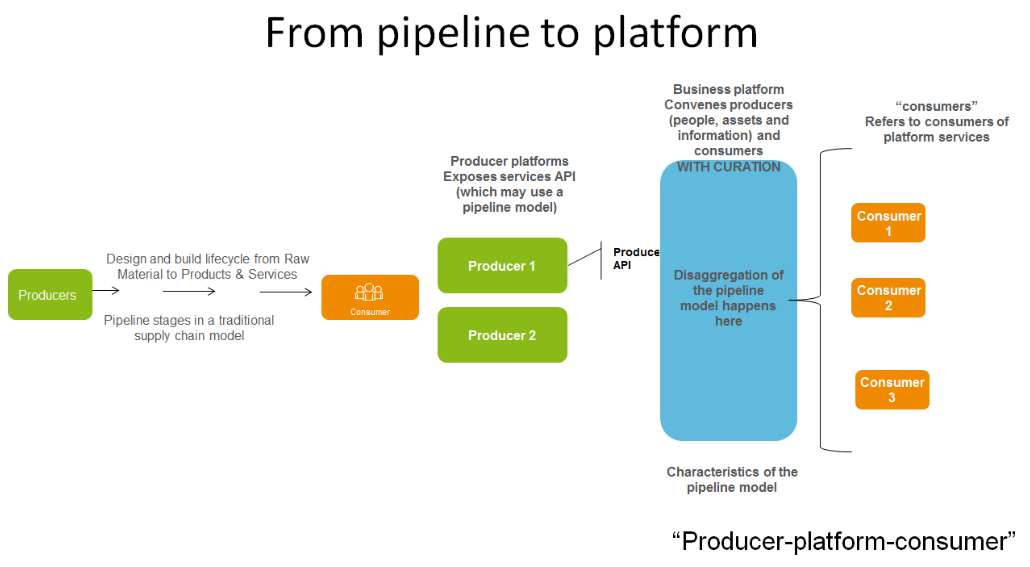

From pipeline to platform
Business platform Convenes producers (people, assets and information) and consumers WITH CURATION
"consumers" Refers to consumers of platform services
Producer platforms Exposes services API (which may use a pipeline model)
Producers
Design and build lifecycle from Raw Material to Products & Services
Pipeline stages in a traditional supply chain model
Consumer
Producer 1
Producer 2
Produce API
Disaggregation of the pipeline model happens here
Consumer 1
Consumer 2
Consumer 3
Characteristics of the pipeline model
"Producer-platform-consumer"

Figure 1 - Pipeline to Platform

1.1.2 *What is a platform?*

1.1.2.1 *Platform Concept*

The concept of Platform is derived from work in MIT Sloan Institute [Mit17] as part of a management science approach to:

- Align IT and business objectives,
- Move to reuse of applications,
- Improve the agility with which IT solutions can be constructed and evolved.

Platforms, as a concept, can be used by service providers (SP) to rationalize and transform both their digital system applications, and organization to common business objectives. A platform is a grouping of services, people and roles.

What & how: Platforms have managerial significance by exposing coherent blocks of business functionalities ("the What") realized through a combination of systems, people, processes and information ("the How"). Platform concept relates to, but goes beyond, Service Oriented Architecture (SoA).

Recursion: To help understand the recursive nature of platforms, consider IBM BlueMix: it exposes Watson Analytics as part of its platform capabilities. However, Watson Analytics can be exposed as a platform itself. It is this dynamic composability that makes the platform approach so powerful, and a major enabler for agility.

Multi-sided platform: Examples of Multi-sided Platforms include Amazon Web Services (AWS) and IBM BlueMix. Platforms enable companies to assume the role of Curators, bringing together Consumers and Producers in a very flexible and easy to use manner. This pushes platforms beyond the definition of a well-enabled SOA service by emphasizing the platform's role in creating business value, not just a function.

1.1.2.2 *Platform Formal Definition*

A platform is formally defined if:

- <u>It exposes easy-to-use APIs</u>: standardizing APIs (e.g., TM Forum Open APIs) is a critical success factor for realizing the full value of Platform Business Models. Standard APIs lower the cost of participating in Platform businesses to customers (write once, integrate everywhere), and the investment required by Platform Owners themselves in leveraging legacy internal capabilities.
- <u>It exposes searchable capabilities</u> as coherent and compose-able blocks of business functionalities and operational patterns, exposed or published in a catalog accessible through one of the Catalog APIs (e.g., compliant with TM Forum Catalog API). Capabilities editors may be business actors like organizations, governments, industry groups, and so on.
- <u>It is modeled</u> as structural hierarchies and layers, associated usage rules, capabilities and APIs. Modeling is managed through a Management API (e.g., TM Forum DSM-API).

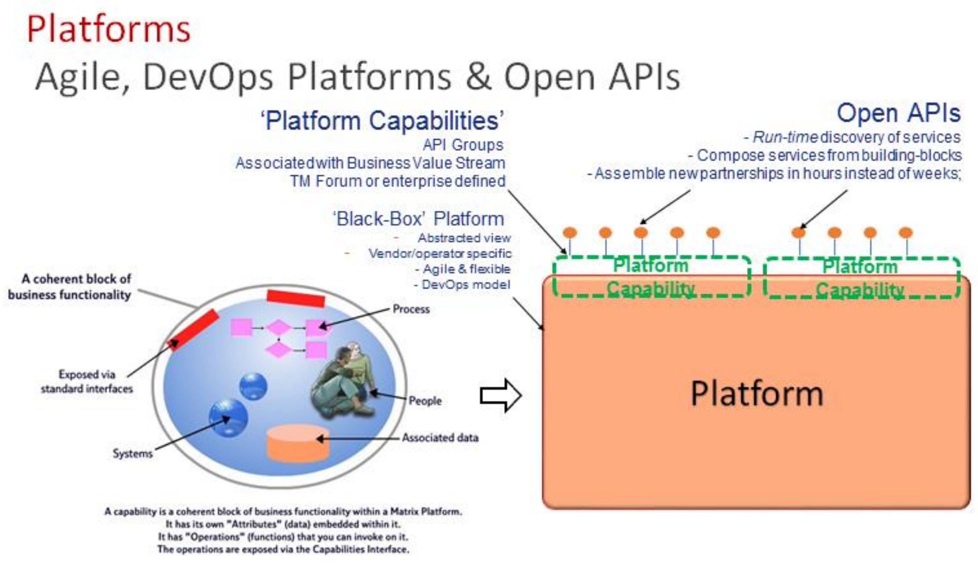

Platforms
Agile, DevOps Platforms & Open APIs
'Platform Capabilities'
API Groups
Associated with Business Value Stream
TM Forum or enterprise defined
'Black-Box' Platform
- Abstracted view
- Vendor/operator specific
- Agile & flexible
- DevOps model
Open APIs
- Run-time discovery of services
- Compose services from building-blocks
- Assemble new partnerships in hours instead of weeks;
Platform Capability
Platform Capability
Platform
A coherent block of business functionality
Exposed via standard interfaces
Systems
Process
People
Associated data
A capability is a coherent block of business functionality within a Matrix Platform.
It has its own "Attributes" (data) embedded within it.
It has "Operations" (functions) that you can invoke on it.
The operations are exposed via the Capabilities Interface.

Figure 2 – Example of Platform formal definition

1.1.2.3 *Platform Technical Characteristics*

As a consequence of the formal definition, a platform bears several implementation technical characteristics:

- Is data-driven and uses analytics to drive intelligent / cognitive business processes.
- Is configurable, extensible and software defined.
- Provides context awareness of services, customers and resources, augmented by analytics.
- Scales and changes components behavior without impacting other components (e.g. micro-services).
- Is event-driven with policy management and closed-loop automation.
- Is runnable anywhere.
- Has a model-driven architecture.
- Supports heterogeneous multi-technology, multi-vendor and multi-operators.
- Supports Intent-based services compose-able at different layers of abstraction.
- Supports layered and distributed orchestration.
- Has built-in security.

1.1.3 Platform Layers

A typical platform is made of different layers, each being a platform in itself:

- Business Layer,
- Platform as a Services Layer (PaaS),
- Infrastructure as a Service (IaaS) layer.

1.1.3.1 Business Layer

The Business Platform View supports Platform Business ecosystems described in the book *Platform Revolution* by "enabling value-creating interactions between external providers and consumers."

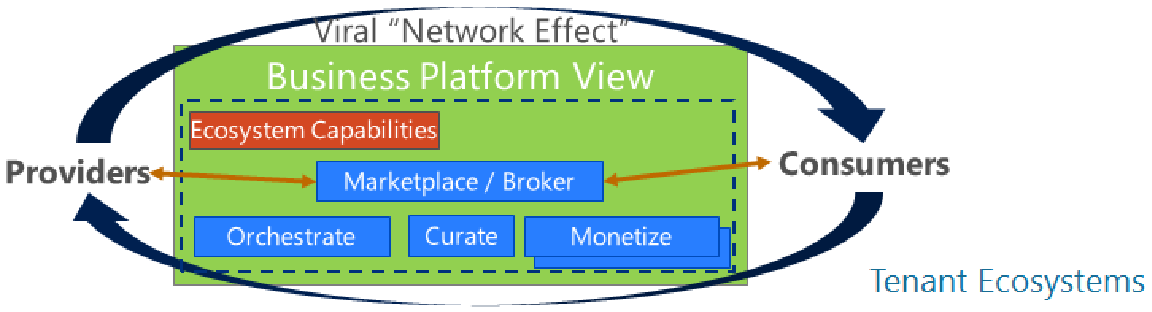


Figure 3 - Business Platform View

The Capabilities are:

- Edition of new tenants where we consider 3 types of tenants:
  1. **Platform Operator Tenant**:
     - Performs all commercial functions necessary to operate the business platform, which is typically a business by itself.
     - Provides a marketplace or portal, systems to on-board developers, customers, partners, suppliers etc., a catalogue to all 1st and 3rd party DevOps Tools, PaaS & IaaS offerings, service endpoints, finished services, VNFs etc.
     - Performs all the functions of fulfilment, service assurance including metrics and telemetry, usage collection, charging/rating, bill presentment and payments/settlements.
  2. **Provider Tenant**:
     - Is any party not owning the business platform itself, even if they are in the same company.
     - On-boards, instantiates, orchestrates, and manages services that digital ecosystems create and use to operate their business.
     - For example, all the BSS/OSS that any given CSP or DSP would need to run their business would fall into to this category.

3. **Consumer Tenant:**
   - Is typically a developer who consumes IaaS and PaaS services offered to create the necessary business services.

   Note: the final end-user is not one of the tenants as it interfaces with the operational system, not with the edition of the system.

- Edition of new interactions between tenants. e.g. smart contracts
- Billing: monitoring usages, logging call detail record (CDR), issuing and paying bills, etc.

1.1.3.2 PaaS layer

In general, platform services are consumed without explicitly managing the underlying Compute, Storage and Networking resources. In mature cloud platforms, particularly hyper-scale cloud platforms, the management of the underlying Infrastructure services is handled automatically via the PaaS capabilities.

Examples of PaaS capabilities are:

- Intelligence & Analytics
- Data
- Media
- Integration
- IOT
- Messaging & Hub
- Identity & Security
- API Management
- Cybersecurity
- Developer
- Monitoring & Management
- Web & Mobile

1.1.3.3 IaaS layer

IaaS platform capabilities are for those situations where solution architects and developers take an explicit role defining the Compute, Storage and Networking resources to run their applications. Unlike PaaS where the developer can simply specify the services to be consumed and performance targets desired and let the platform allocate and deallocate resources automatically, in IaaS the developers assume responsibility for the entire design and its operational configuration.

Examples of IaaS capabilities are:

- Compute capability
- Networking capability
- Storage capability

1.1.4 Architecture Design Principles

To enter in the digital markets of tomorrow, service providers need to deliver and operate digital services like 5G, IoT or AR services, in a flexible way onto modern platforms, by exploiting NFV and SDN capabilities. Service providers need also to adapt their systems for operationalizing and monetizing infrastructure platforms and platform business models. Transformation of operational and business support systems (OSS/BSS) must meet a set of key business requirements:

- Lower cost of operation
- Support for flexible business models
- Ecosystem-capable for creating offers delivered in partnership by multiple service providers
- Business agility
- Multi-vendor interoperability support
- Agile governance
- Ability to exploit the flexibility of cloud

The Open Digital Architecture (ODA) embraces a new vision for OSS/BSS and defines a blueprint for transformation by delivering a model- and data-driven architecture that relies on the use of metadata, management components (microservices) and a clear set of normalized APIs. Indeed, ODA is a platform blueprint for BSS/OSS, capable of hosting diverse business capabilities, operating models and business models.

All following sub-sections refer to ODA.

1.1.4.1 Functional Architecture

In ODA, the architecture functions are grouped into 3 systems with their own and independent lifecycles and 5 blocks of closely interrelated services grouping, as depicted in Figure 4 taken from [IG1167]:

- Engagement Management (people, organizations, internal and external).
- Party Management.
- Core Commerce Management (technology / production independent BSS functions. i.e. handling commercial processes and has no detailed knowledge of production technologies).
- Production which includes traditional Service and Resource Management Functions.
- Intelligence Management that coordinates intelligence knowledge across all the ODA Functional blocks using a set of patterns including closed control loop, policy and AI/ML.

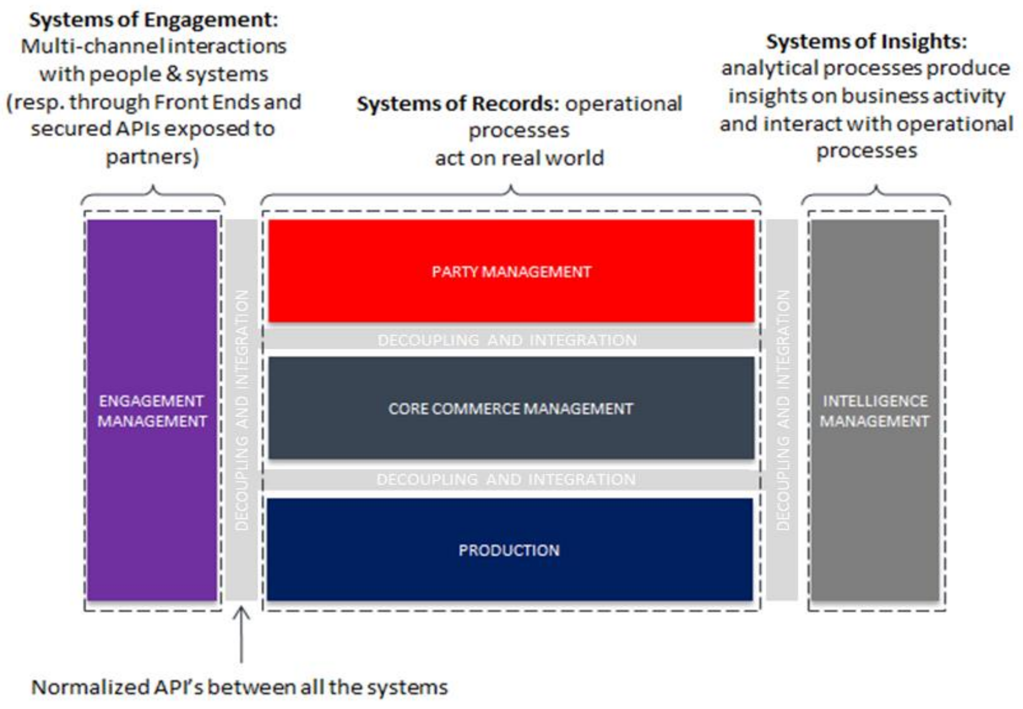

Systems of Engagement: Multi-channel interactions with people & systems (resp. through Front Ends and secured APIs exposed to partners)
Systems of Records: operational processes act on real world
Systems of Insights: analytical processes produce insights on business activity and interact with operational processes
ENGAGEMENT MANAGEMENT
DECOUPLING AND INTEGRATION
PARTY MANAGEMENT
DECOUPLING AND INTEGRATION
CORE COMMERCE MANAGEMENT
DECOUPLING AND INTEGRATION
PRODUCTION
DECOUPLING AND INTEGRATION
INTELLIGENCE MANAGEMENT
Normalized API's between all the systems

Figure 4 - ODA Functional Architecture

In addition, the Decoupling & Integration Block handles the separation of functional borders based on the blocks and the integration between them. D&I gathers non-business functions (standardized APIs, Message routing) that are needed at design and runtime to enable the combinations of services that the business requires.

- **Engagement Block:** Interaction point between the enterprise and its ecosystem, which can be through different channels for different experiences.
- **Party Management Block:** Deals with "WHO" and "WHY". Handles parties involved in business processes. These Parties can be Customers, Business Partners or Employees.
- **Core Commerce Management:** Deals with "WHAT". In charge of Product Offerings and Products Catalogue oriented processes, functions and repositories
- Independent from technical concerns to be able to decouple commercial and technical evolutions
- **Production Block:** Deals with "HOW ". Responsible for the lifecycle management of Customer Facing Services (CFS) and Resource Facing Services (RFS) regardless of the technology type or the operational domain or factory where it originates, including third parties.
- Does not need to know the details of a product nor the customer, contract owner, or payer requesting its exposed service capabilities.
- **Intelligence Management:** Provides "closed-loop" capabilities to package and consume data. Relies on analytical data to develop insights for decision support, or intent-based actions or advice.

1.1.4.2 *Components*

ODA functions can be composed and implemented into ODA components that expose TM Forum Open APIs and are deployable into an operational runtime environment. The architecture of ODA components, as presented in Figure 5, is composed of several functions:

- Core Function: represents the business functionality of the ODA component, i.e. a support function with a dedicated business purpose. It is possible to integrate with other ODA Components and Services over standard external interfaces.
- Security function: responsible for access control to all the component's interfaces, authenticating the accessing user and authorizing the use of the component's different functions according to authorization rules.
- Management/operation Function: responsible for the operations of the component (start, stop, health-check, alarming, etc.), for configuration of the functions within the component and for performing maintenance of the component (preventive maintenance and corrective maintenance)
- Notification/reporting: responsible for log and keep statistics over component operations.

Security, Management/Operations and Notification/Reporting are nearly identical for all ODA components and are based on a common set of generic and technology independent APIs, composed of subsets of the TM Forum Open APIs.

The environment dependence/requirements contain dependencies on the run time environment such as Containers, Virtual Machines and dependencies with other ODA components executing in the run time environment.

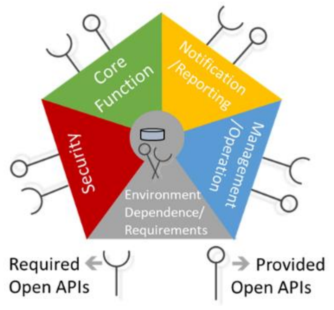


Figure 5 - ODA components

Execution Platform

The **Execution Platform** of the ODA component represents the environmental functionality necessary for the functions to execute and work together. Specifications in the ODA Component will not point out a specific platform or a specific technology.

The Execution Platform contain specifications for

- required technical functions to support and execute the ODA Component functions and APIs. Typical technical functions are data storage, inter-function communication, technical interfaces to support the APIs' protocols.

- non-functional requirements to be used in the implementation and deployment phases. Examples of non-functional requirements are execution characteristics like real-time, interactive (best effort) or batch, specific requirements for Scalability, Deployment, redundancy of the component or its data.

1.1.4.3 Services

To achieve the needed agility, any service provider needs decoupling between the business offers to its customers and the technical solutions behind the products. It needs to integrate multiple technologies and manage them as services offered as products to its customers. Composing sophisticated services from heterogeneous technology domains requires integration decoupling of the “what” (i.e. service) from the “how” (i.e. technical functions that realize these services). Conversely, any technology solution should then evolve independently from the service and the commercial offer. By applying ODA concept of decoupling the lifecycle of Production Functions from the lifecycle of Core Commerce Function, a consumer tenant can benefit in business agility.

These design principles are illustrated with the example in Figure 6:

- Service is Network-as-a-Service (NaaS) and tenant is a CSP.
- Decoupling operational and business support systems (OSS/BSS) from the network itself, NaaS is interpreted broadly covering both connectivity services and end point services like applications (web servers, IMS servers, IoT data hubs, etc). CSP gains ability to create multiple products by assembling multiple reusable NaaS (exposed services) and save by the same occasion integration time and costs.
- These services are exposed through open APIs that hide the detailed network and other asset resources. Each operational domain of CSP exposes NaaS APIs as CFS. NaaS APIs offers a unified, standard way to manage network-exposed services using TM Forum Open APIs. This is detailed in TMF909 API Suite Specification for NaaS v3.0.1 and cover APIs for service catalog, service qualification, service ordering, service activation and configuration, service test, service problem and usage consumption for service billing.
- Multiple technical domains for multiple technologies are mapped to an operational domain. They contain Resource Facing Services RFS, Resource Functions and Resources.

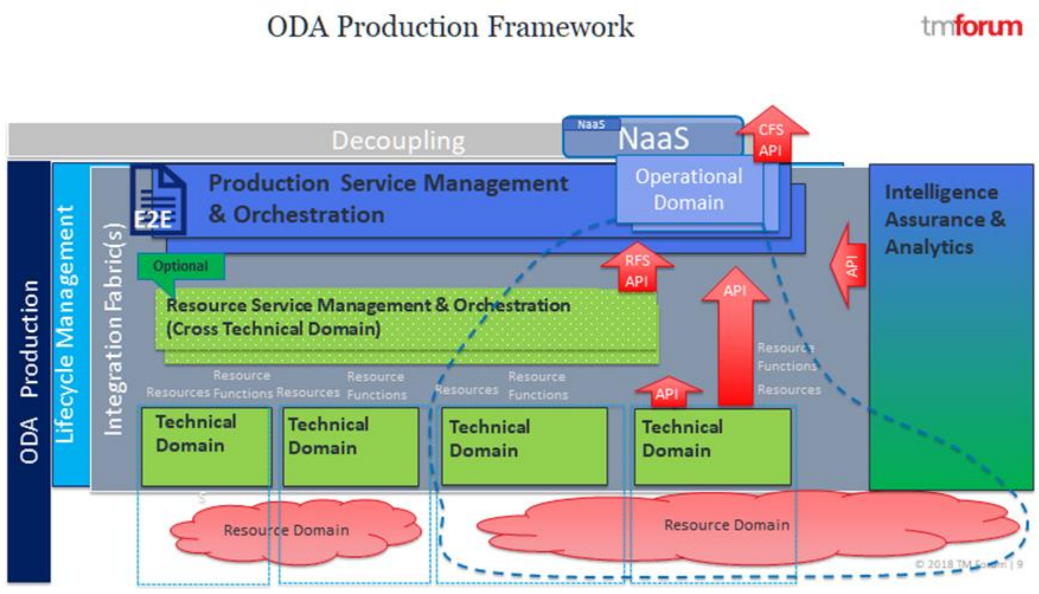


Figure 6 - Application of Network-as-a-Service in ODA platforms

1.2 Applicability

This section explains how TM-Forum-based approach is applied to ARtwin. ARtwin solution targets the deployment of an ARtwin platform to offer a wide variety of cutting-edge AR-based services tailored for Industry and Construction 4.0, deployed at large scale in major factories and construction sites. Such AR-based services will be delivered at speed and in a reliable and secure manner thanks to the new 5G connectivity service. To achieve those first-in-class features, the ARtwin solution will be built by applying the TM Forum principles as follows:

- **A well-enabled ARtwin platform.** ARtwin is a multi-sided platform where various partners involved in the business chain of industry and construction 4.0, spanning from the component developers, to the component vendors, to the operator of the platform and finally to the industry and construction partners as the tenants of the platform, need to interact. The overall ARtwin platform needs to be decomposed into various “sub-platforms”, each sub-platform consuming platform capabilities of another sub-platform: Pipeline, AR platform, 5G platform, BIM, etc. The interconnections between sub-platforms need to be dynamically established at deployment or instantiation time, respecting the dependency of one sub-platform deployment on another one. As a conclusion, only the platform model approach enables the required reconfiguration agility. ARtwin platform will be built as a *platform of platforms*.
- **Well-designed business services** need to be exposed to customers of ARtwin platform using intent abstractions:
    - Inspired by the NaaS example, we suggest reusing RFS and CFS abstractions to decouple services offered by the ARtwin platform from any underlying technical implementations.
    - Industry and construction parties acting as tenants of ARtwin platform will be able to mix and match these technology neutral services to compose new and sophisticated services for their own business purpose.

    - Alternatively, a CSP acting as a tenant of the ARtwin platform may offer to industry and construction 4.0 enterprises turn-key AR-enriched communication services composed from 5G connectivity services and AR services.
- **Well-designed architecture** is needed to orchestrate ARtwin platform lifecycle:
    - Given the complexity of the various targeted deployments (e.g. distributed cloud, AR devices exotic “execution environments”, etc.), we suggest reusing ODA design principles for tackling them in a scalable way and hiding all the complexity.
    - Given the complexity of the various targeted services, we suggest using unified and standardized APIs.

This applicability in a general sense is also confirmed in the following detailed preliminary specifications.

2 Definitions, Terminology, and Methodologies

2.1 Platform of platforms

Platform of Platforms (PoP) is a concept initially developed in NGPaaS EU project (2017-2019) allowing lifecycle management of other platforms. This section introduces this technology.

2.1.1 Purpose

1. **Managing independent and heterogeneous systems**

A PoP is typically a system consisting of multiple independent and heterogeneous systems. In Artwin, the elements and workflows will be heterogeneous in part because they will come from a variety of sources. The independent AR/VR and 5G telco systems components will run on different hardware/software platforms, developed using different languages and designed according to different methodologies, business cases and partners “know how”. Hence, an important concern of PoP system design, construction, and evolution is interoperability which will entail integrating heterogeneous elements and engineering perspectives. In PoP, interoperability must be addressed and managed at several levels:

- At technical level, interoperability needs to address the conventional syntactic and semantic interoperability.
- Beyond technical level, a broader interoperability includes social, organizational, and legal levels to support the integration of data and processes.
- For this, an infrastructure is needed to combine development, deployment, and operational support for generalized interoperability between organizational teams.

2. **PoP as a Platform Factory**

The PoP can be considered as a “platform factory” that enables generating, assembling, storing, and deploying new platforms. PoP offers a “Built-to-order” mechanism that exposes each platform on a repository following the marketplace concept. Platforms themselves are composed of RFBs that will be introduced later in the next section. A platform is edited, built and stored in a platform repository using a version control system. Figure 7 shows the Platform building workflow.

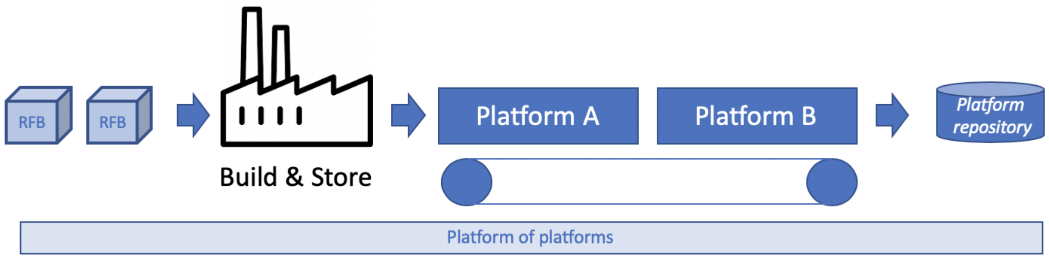


Figure 7 - Build and store platforms

2.1.2 Design

1. **Composition by RFBs**

In the last decade, Service-based architecture has been gaining momentum in the software industry, evolving towards “Microservice Architecture” which Martin Fowler describes as:

"a particular way of designing software applications as suites of independently deployable services. While there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.” [LF14]

This architecture pattern was found to be particularly interesting in large-scale software due to its inherited capability to solve many problems that comes with developing and operating complex distributed systems.

In a microservice architecture, each microservice is a component of a larger software or service. Flower defines a component as:

*“a unit of software that is independently **replaceable** and **upgradeable**.”*

Similarly, platforms tend to be very complex systems that need to be componentized to achieve a higher level of flexibility and agility in design, development and operations.

Reusable Functional Blocks (RFB) are the components of platforms. It is the logical representation of a function that helps to decompose a complex system into a set of simple one-job components. It is an abstracted definition of the component that contains information about it rather than its implementation. An RFB is composed and structured of two main building blocks: *metadata* and *function*. The metadata block describes the functional and operational characteristics of the function implemented by the RFB. The function block contains information about the function itself, its format, configuration parameters and so on. For instance, in this section you can provide a software image that can be executed as a virtual machine, a container or an executable file according to the targeted execution environment. The *execution environment (EE)* is also an important element of an RFB definition. It specifies the target environment where the function should be deployed and run. Logically, the EE is represented below the function block in Figure 8, but in the implementation, it is specified in the metadata section.

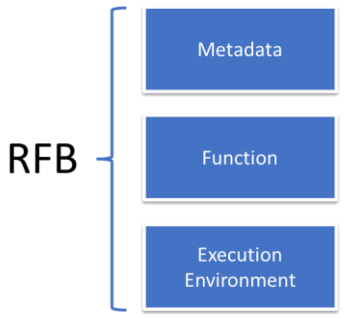


Figure 8 - RFB Building Blocks

RFBs features include:

- Platform independence
  - RFBs remove any dependencies between the function and its execution environment (e.g. OSM, Kubernetes, Openstack, AWS…)
  - RFBs can be realized via variety of software functions and programming languages (polyglot Infrastructure).
- Recursion
  - RFBs can be composed in graphs to provide services or to form other RFBs
  - RFB has a multilevel hierarchy: Domain, Parent, child & Leaf

This composability can be used to describe any part of the system, from service to execution environment. We use the composition graph at the level of Services, Platforms, Infrastructures (Figure 9).

Service
PaaS
IaaS

Figure 9 - IaaS, PaaS and SaaS

2. Separation of Concerns by Abstraction

Two main ideas behind the PoP are to provide isolation and abstraction. Since PoP means horizontally assembling and connecting several platforms and business entities isolated from each other through domains, we also consider abstraction from upper layers to lower layers. Abstraction means removing complexity by making a clean separation between deployment technologies in execution environments from functional level and RFB implementation at platform level. On the upper layer, the service layer is connecting RFBs and platforms horizontally.

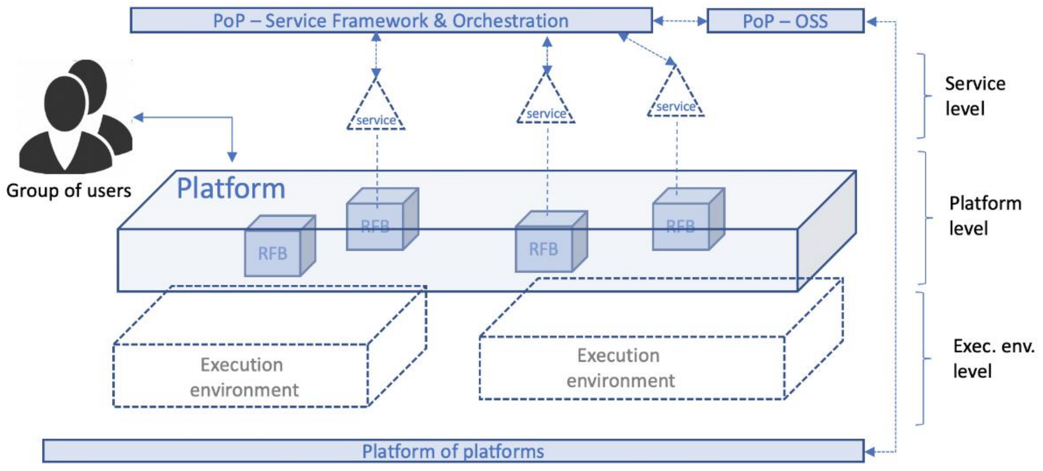


Figure 10 - Platform Description

3. Service Definition

As explained in section 1.1.4.3, services define the business functions performed by the Platform, the supported service level agreements (SLA), and organizational policy constraints.

To support these requirements, Service framework (SFW) is implemented as a customizable catalogue where the consumer tenant can choose services from. Each service points to a blueprint which can further decompose in deployable RFBs. The PoP is then further responsible to deploy the decomposed service. If the consumer tenant requests a use-case oriented application or service still not in the catalogue, that component should be first developed involving the provider tenant. This is in fact the common situation in a telco-grade environment.

4. Execution Environment

Execution Environment is a generic and abstracted term to describe an environment used for deploying services or platforms. Traditionally, software used to be developed to run on a specific hardware. The hardware needed to be taken into consideration at the very early stages of software development lifecycle even before writing any line of code. Leveraging the concept of reusability (*Write Once, Run Anywhere* introduced by Oracle in the 90s', or *Build Once, Deploy Anywhere* that emerged with containers) software is nowadays being decoupled from the environment on which it is intended to run, whenever it is possible.

PoP has this concept at its core. Ideally, a platform is built agnostically from where it is going to run, allowing its operator to configure or specify this environment as its *Execution Environment.* As platforms are usually deployed on infrastructure resources, the type of execution environment of a platform is usually an infrastructure, whether physical (e.g. bare metal servers) or virtual (VMs). For example, if we consider a Kubernetes cluster as a platform, we can configure a set of servers (nodes) to be the target of deploying this cluster as Execution Environment.

Services apply the same concept and specify the platforms on top of which they run as an Execution Environment. For example, if we want to deploy a container-based AR rendering service, its execution environment can be the Kubernetes cluster platform.

5. Domains

The PoP supports the concept of a Domain to group-related clusters and servers. Some organizations use separate domains for each business unit or capability, others use it to group computational workload. How it is used is up to the organization.

For each domain deployed on PoP, a Platform Blueprint contains a logical definition of a domain. For this reason, it is typical for a separate blueprint to be created in RDCL Studio for each unique platform domain type.

In ARtwin, domains can be organized by business applications, by AR technology, etc.

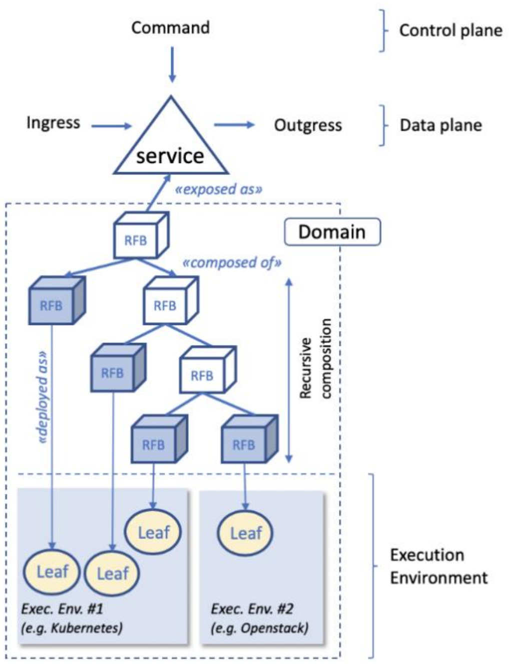


Figure 11 - Service, Domain, and Execution Environment

2.1.3 Operations

1. Platform Deployment, Orchestration and Monitoring

PoP manages the full deployment lifecycle of Platforms by abstracting targeted deployment environments or technologies from platform definition as shown in Figure 12.

- PoP creates, for each platform instance, a separate model defining all specific environment configuration information.
- This infrastructure-independent model enables:
  - Provisioning consistent platforms across all environments regardless of infrastructure type (on premise and on cloud).
  - A built-in and centralized way for operating and monitoring platforms.

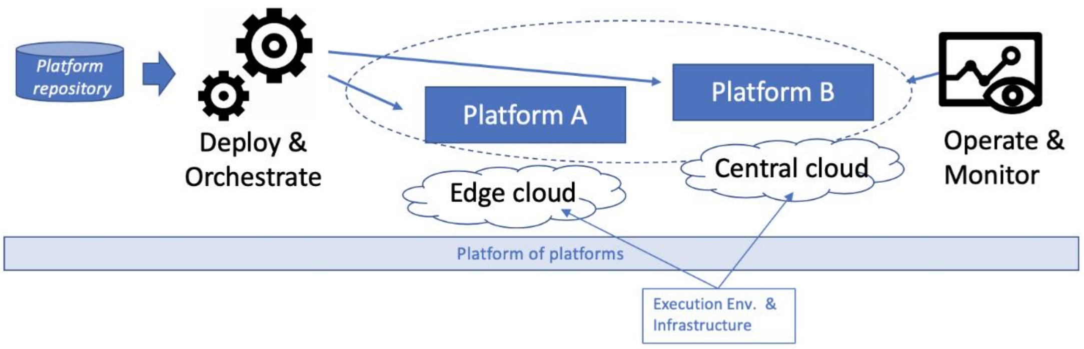


Figure 12 - Deploy, orchestrate, operate and monitor

2. **PoP Continuous Integration/Continuous Delivery Pipeline**

The PoP provides CI/CD pipeline capability (Figure 13) that enables platform developers to manage the full development lifecycle and bring DevOps practices to platform developers. The build process is orchestrated by a Continuous Integration (CI) Server such as Jenkins. During this process, the CI Server will periodically poll the source code repository, such as GitLab, for any code changes (or commits) since the previous build.

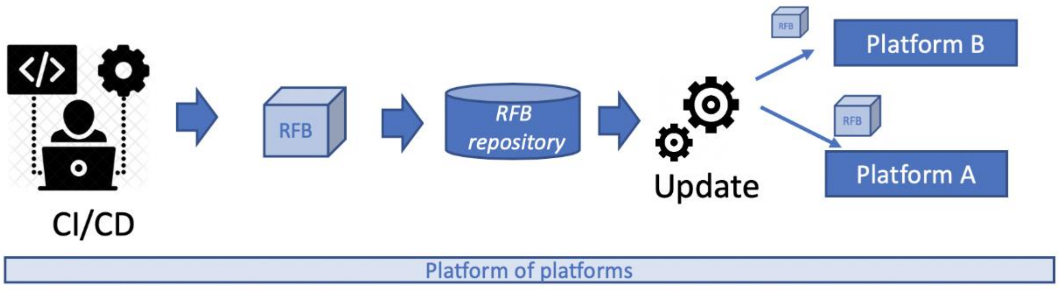


Figure 13 - Continuous integration and Continuous delivery in PoP

2.2 Platform Views

TM-Forum's approach proposes 3 types of views to different stages of platform and service management lifecycle:

- The *contractual view* is where roles are defined and access delivered.
- The *edition view* is where new components can be added and platforms edited.
- The *operational view* is where platforms are being deployed and operated.

Figure 14 shows the interactions between multiple actors, each having access only to the view that corresponds to their role and function.

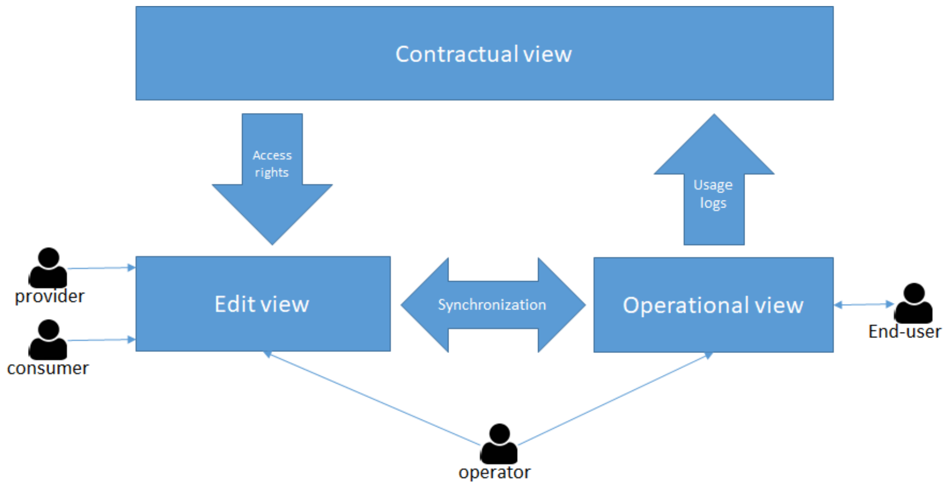


Figure 14 - Multiple roles and multiple views

2.2.1 *Contractual view*

This view defines and provides access permissions. Role-Based Access Control (RBAC), aka non-discretionary access control, is an access control system that secures access to resources. It relies on the fact that in many organizations, access is often granted based on employee's functions rather than their ownership of the resource or object being accessed. For instance, all tellers in a bank have the same level of access to same information which is owned by the bank itself and not a specific teller. Therefore, a Role-Based Access Control policy decides about access requests based on the functions assigned to a user within its organization. In a nutshell, RBAC assigns permissions to roles and not directly assigned to users. When a user is created, it is then assigned to the role that corresponds to its function and holds the appropriate permissions. Figure 15 illustrates this concept.

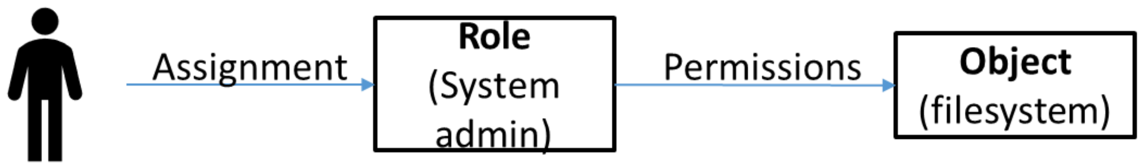


Figure 15 - Users and Permissions in RBAC

The ANSI INCITS 359-2004 standard defines 4 reference models, but we are only interested in the Core reference model to build a good understanding of it. The Core RBAC reference model defines element sets and relations that are considered essential and therefore must be present in any implementation.

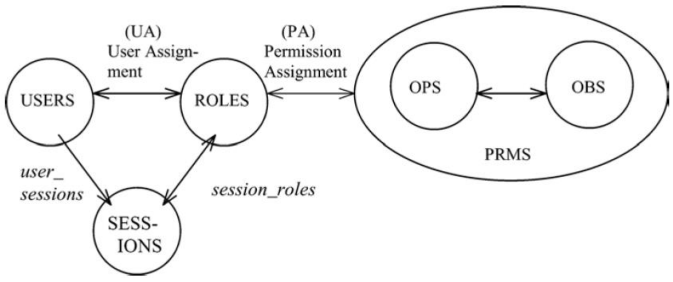


Figure 16 - Core RBAC Reference Model. Source: ANSI INCITS 359-2004.

As shown in Figure 16, this model identifies five basic data elements: users (USERS), roles (ROLES), objects (OBS), operations (OPS), and permissions (PRMS). RBAC's fundamental idea is that permissions should be granted to roles and users being assigned to different roles. In this sense, roles are technically "a means for naming many-to-many relationships among individual users and permissions". Sessions (SESSIONS) on the other hand are mapping between a user and a subset of roles that get activated after the user's authentication.

- **User:** A *user* is defined as a human being. Although the concept of a user can be extended to include machines, networks, or intelligent autonomous agents, the definition is limited to a person in this document for simplicity reasons.
- **Objects:** an *object* can be any system resource subject to access control, such as a file, printer, terminal, database record, etc.
- **Operations:** An *operation* is an executable image of a program, which upon invocation executes some function for the user.
- **Permissions:** *Permission* is an approval to perform an operation on one or more RBAC protected objects.
- **Role:** A *role* is a job function within the context of an organization with some associated semantics regarding the authority and responsibility conferred on the user assigned to the role.

The many-to-many relationship (indicated by the bidirectional arrow) between roles and users, and roles and permissions, gives us a high level of flexibility and granularity in defining the access policy. This also enables the application of the least privilege principle, where users/roles have access to nothing more than what they need to perform their job.

Section 2.3 shows some examples of RBAC resources that can exist in ARtwin.

2.2.2 Edition view

The Edition View provides all frameworks (Business Process, Data, Application and APIs) to guide the development of the actual systems before deployment. The steps are as follows:

1. **Conception**: modelling the components that comprise a Platform, how Platforms expose capabilities and how Platforms are interconnected.
2. **Specification**: defining the services offered by the Platform as a Service (PaaS), and specifically the functionality of APIs which expose the Actualization Platform capabilities.

3. **Implementation**: developing those exposed APIs for one or more software implementations, e.g. Microsoft Azure, AWS, 5G Access Network, Openstack, etc. with the supporting operations processes, policies etc.
4. **Deployment**: deploying an instance of the edited platform on an IaaS environment; thus, allowing tenant SaaS services to run on the deployed Platform instance using open APIs.

Because RFBs are the main elements to compose platforms, *RDCL* is a web tool that provides a graphical user interface to work with RFBs. It allows us to create RFBs, compose and edit platforms using those RFBs, and deploy platforms on top of their Execution Environment. As mentioned before, the composition pattern is used at almost all the levels of the stack from infrastructure (IaaS) to services (SaaS). Hence, the tool enables us to use the same workflow to not only to compose platforms, but also infrastructures and services.

The edition of platforms and services in RDCL is done using a *graph* representation. Considering the example of Kubernetes as PaaS, its graph is illustrated in Figure 17. There are two categories of RFB; the first being the logical grouping and then the components that will be deployed, the leaf RFB's. Only hierarchical links are used between components, as there are no hard dependencies between the order in which the components are deployed.

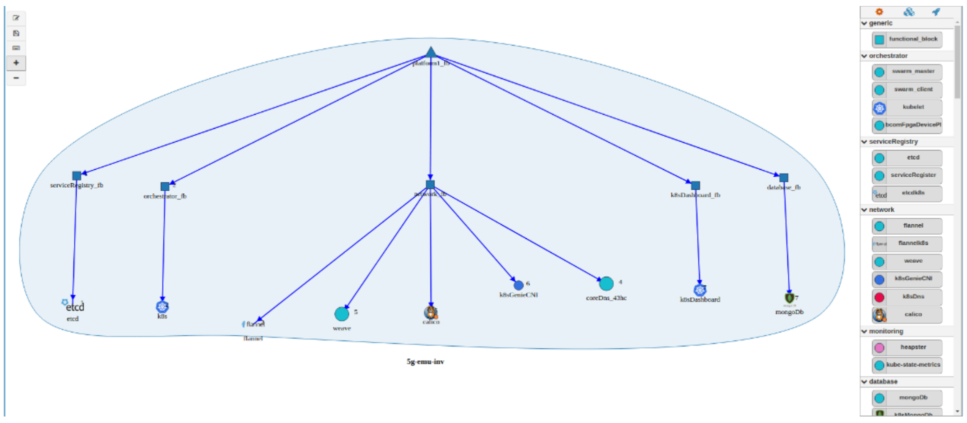


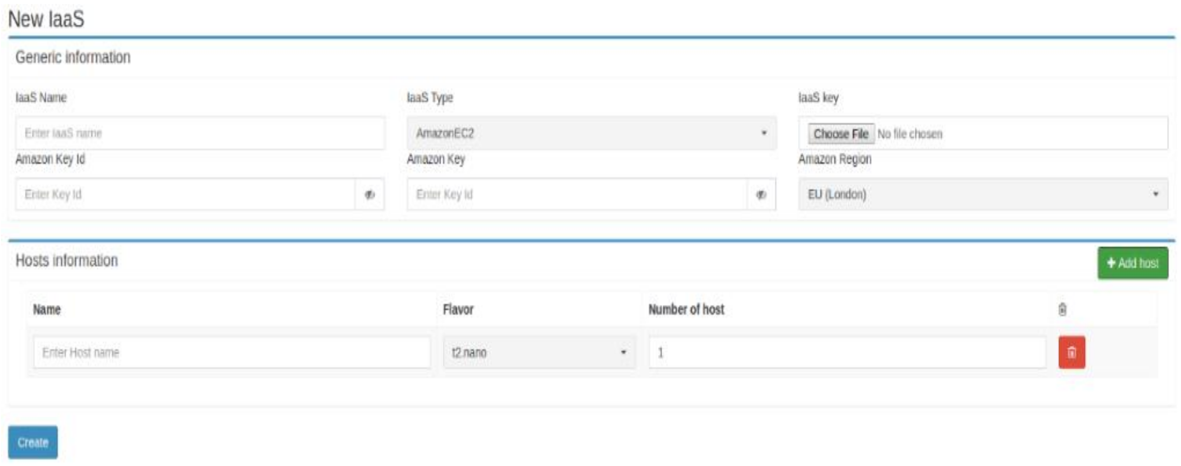
Figure 17 - Kubernetes Platform Composition

Disaggregating Kubernetes PaaS into components is important to offer the flexibility to support new features like new network plugin without having the overhead to modify the full code. In this example, there are three essential classes of RFBs:

- **Service registry**: because Kubernetes support only ETCD as service registry so ETCD is mandatory here.
- **Orchestrator**: We are running Kubernetes PaaS so Kubernetes is our orchestrator.
- **Networking**: Kubernetes support a lot of Network CNI such as Flannel, Weave, OVS, Calico… so that in network logical group, we can compose network RFB with any supported CNI. Besides, Kubernetes also need a DNS to discover service in Kubernetes, so a DNS is also necessary in this network RFB.

Those are basic but mandatory logical RFBs. We can also add to our Kubernetes cluster other RFB like dashboard, database or image registry.

Also, as an illustrative example, Figure 18 shows an Amazon Web Service (AWS) Elastic Compute Cloud (EC2) resource declaration in RDCL. You can choose the region of this AWS IaaS and provide key id/key so that RDCL is authorized to manipulate your AWS EC2 account. Then, you can add machines to this IaaS and choose the flavor for your machines. Because we must pay for the running machine in AWS EC2, at beginning we will not run any machine in this IaaS. So, this step is considered as an IaaS declaration. The machine in Amazon will be run when you start to use it and will be destroyed the case you have nothing running in that machine. If you have a permanent running machine AWS EC2, you can also declare it within this page.


New IaaS

Generic information

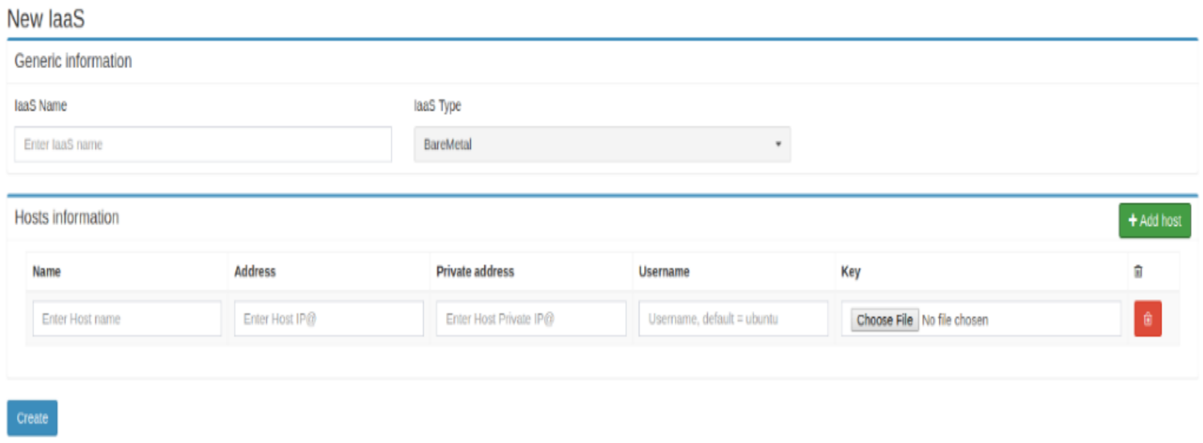
IaaS Name: Enter IaaS name
IaaS Type: AmazonEC2
IaaS key: Choose File No file chosen
Amazon Key Id: Enter Key Id
Amazon Key: Enter Key Id
Amazon Region: EU (London)

Hosts information

Name	Flavor	Number of host
Enter Host name	t2.nano	1

Figure 18 - AWS declaration in RDCL

In Figure 19, another type of IaaS declaration is supported, a bare metal IaaS. With this type, you can declare your local infrastructure to the tool with connectivity information such as IP address, username, key pair.


New IaaS

Generic information

IaaS Name: Enter IaaS name
IaaS Type: BareMetal

Hosts information

Name	Address	Private address	Username	Key
Enter Host name	Enter Host IP@	Enter Host Private IP@	Username, default = ubuntu	Choose File No file chosen

Figure 19 - Bare Metal machine declaration in RDCL

After the declaration of IaaSes is completed, the IaaSes can be composed to build an inventory which is a REE (RFB Execution Environment) of a PaaS. Figure 20 depicts the interface where composition of machines builds an inventory. On the right side is the list of all available machines (the machines that you declared). Dragging-and-dropping a machine into the canvas area adds that machine to your inventory. Then they can be aggregated into group (rectangle) to facilitate the usage of the machines. After the creation of an inventory, it can be used as REE of a PaaS so that when a PaaS is deployed, it will be deployed on machines in this inventory.

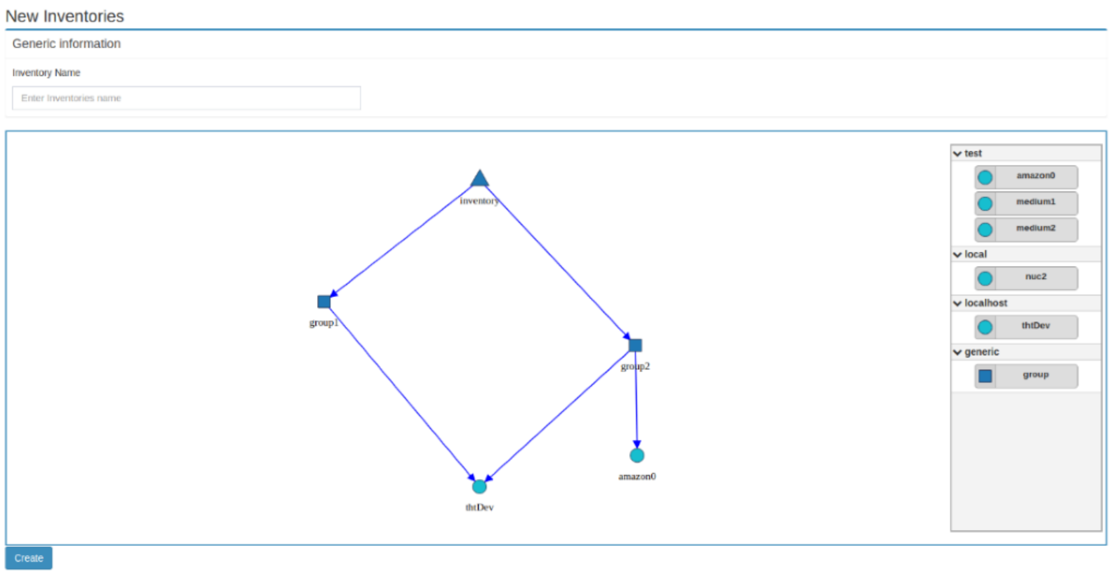


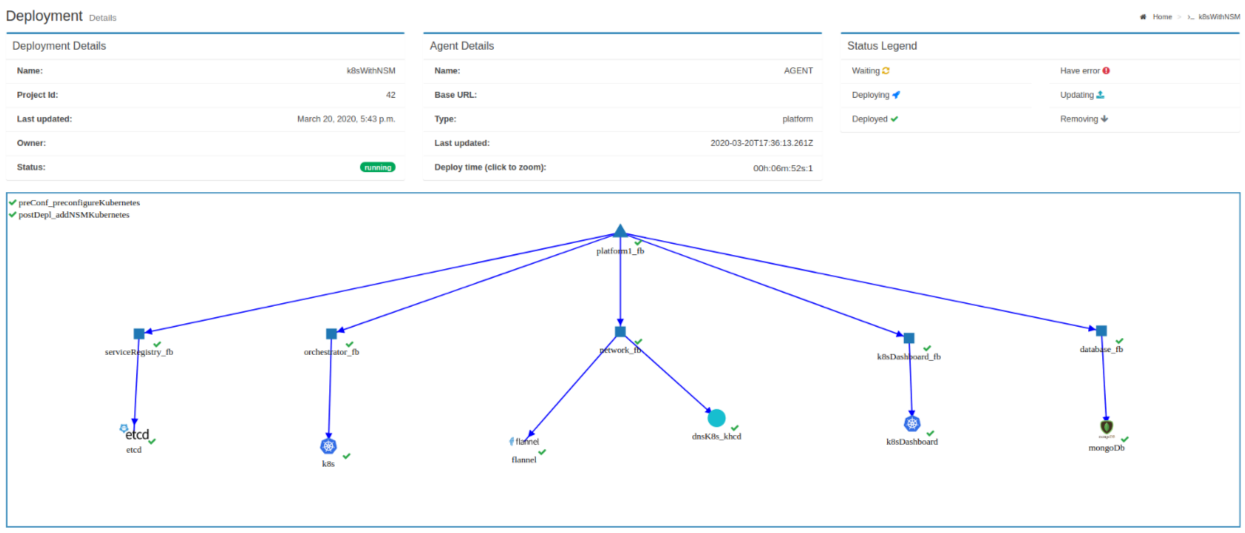
Figure 20 - IaaS composition

2.2.3 Operational view

This is where all the actual technical systems that are required by the designed platform are deployed and operated. This view contains mainly a Platform as a Services Layer (PaaS), and an Infrastructure as a Service (IaaS) layer. It is based on cloud computing architectures hosting IT Workloads and Network Workloads such as Network Function Virtualization. In this view we can find a network of data centers and Central Offices typically spanning hyper-scale cloud and 5G Access Networks. It is where software is deployed, functions chained together, work is accomplished and where the software enabled services are executed including network functions. It is also where all virtualized resources management occurs including the functions of a Virtual Infrastructure Manager (VIM) and Network Function Virtualization Infrastructure (NFVI).

This is where platforms operators can check the operational state of their platform and manage it in real time. They may even have access to the underlying IaaS environment depending on the permissions assigned to their role.

RDCL offers some visibility on the state of its deployments. Figure 21 shows how RDCL shows that Kubernetes cluster is in a “running” state. This means that all the RFBs that compose it were deployed successfully. RDCL will show an error when something goes wrong and an RFB deployment doesn’t complete. It also shows which RFB is having the error so we can know where to look for the error in the logs for troubleshooting purposes.


Deployment Details

Home > k8sWithNSM

Deployment Details	
Name:	k8sWithNSM
Project Id:	42
Last updated:	March 20, 2020, 5:43 p.m.
Owner:	
Status:	running

Agent Details	
Name:	AGENT
Base URL:	
Type:	platform
Last updated:	2020-03-20T17:36:13.261Z
Deploy time (click to zoom):	00h:06m:52s:1

Status Legend	
Waiting	Have error
Deploying	Updating
Deployed ✓	Removing

✓ preConf_preconfigureKubernetes
✓ postDepl_addNSMKubernetes

Figure 21 - Kubernetes in a running state. An example of operational state of a platform in RDCL

2.3 Roles and Actors

2.3.1 Definition

As mentioned earlier, roles are semantics that reflects a job description or functions an employee in an organization carries out. Roles can have the name of the job title and contain all the permissions required to perform the job, or the name of a single function and contains therefore only the permissions to allow this function to be performed. They are defined in RBAC.

Example of roles in ARtwin:

- Experts in Computer Vision (CV)
- Cloud expert
- Low-level component provider
- Pipeline designer
- Application developer

Actors, on the other hand, are related to the business or use-case. They have actions to take during a given scenario. They have one or several roles. An actor can be a partner in the project, or a third-party.

2.3.2 Mapping Roles over Actors

Table 1 provides a mapping between some roles and actors that are currently known or defined in ARtwin. It is based on the platform inventory provided in section 3.2 of this document. The table has few empty cells because at this stage of the project there is still come uncertainties about a few roles. This mapping will evolve throughout the course of the project.

Table 1 - Mapping Roles to Actors

List of actors / List of roles	Scoped platform	Holo-Light	B-COM	CTU	SIE	3rd party (Unity, Unreal…)

Component provider	PoP					NBL (to be completed)
Component provider	Remote Rendering Platform	• RR: Orchestration to nearest Edge Node • RR: Device rendering engine (Large 3D objects) • New components for Orchestration Service • New components for Remote Rendering Service • New components for User Interaction Services • New components for PLM/BIM (industrial usage in AR App) Data Services			• HoloLayer (HL): AR Backen Data-Management	• RR: Host rendering engine • LR: Device rendering engine
Component provider	ARCloud Platform		• Components provided with the SolAR framework • New components for the map update service	• New components for the relocalization service • New components for the map dense 3D mapping service • New components for the 3D segmentation service • New components for the CAD model correction service • New components for building new CAD models from anotated data	• HoloLayer (HL): AR Backen Data-Management	• Components for Tracking service (Among Hololens, ARKit, ARCore) • Components for relocalization (Among InLoc, Teaser++) • Components for 3D map reconstruction (Among AliceVision, COLMAP, Capturing Reality)
ARCloud platform editor	SOLAR [edition of AR pipelines that will feed PoP]	Yes	Yes	Yes	Yes	

Remote rendering platform editor	Hololight solution	Yes	Yes			
PoP editor						NBL
Remote rendering platform operator		Yes				
ARCloud platform operator			Yes			
PoP operator						NBL

2.4 Blueprints

This section introduces blueprints, their definition and usage in the architecture.

2.4.1 Generalities

Coming from the architecture world, the word *blueprint* is defined as "an early design plan for a building or machine". Likewise, we can define a blueprint as a high-level design plan used to describe a service, a platform, an infrastructure, or other systems. It provides an abstraction model of IT systems and objects. It can contain, in a descriptive fashion, information about the capabilities of a platform, its components (e.g. RFBs), its interfaces, inputs and outputs, metadata, execution environment, and so on.

In PoP, RFBs are included in the blueprint to form the internal components of execution environment, platforms, or services. They inherit some characteristics of RFBs like composition. For instance, we can have master service blueprint that is composed of an AR service blueprint that includes, itself, a tracking service blueprint. Figure 22 depicts this concept.

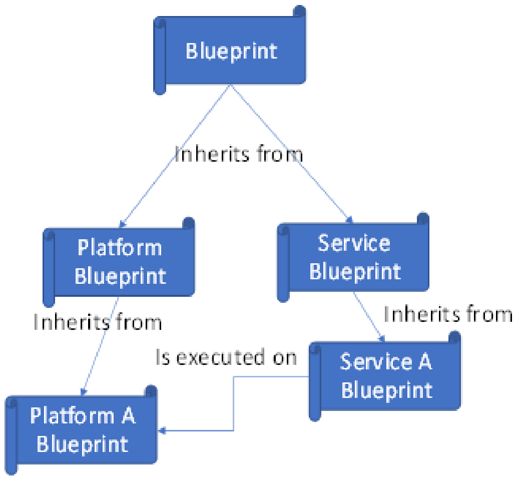


Figure 22 - Blueprints Composition

2.4.2 Service and Platform Blueprint Overview

Service in PoP are defined by formal and explicit formatted specifications or "Blueprint" that stand apart from the "Service Implementation". These Service Blueprint aims to produce a Service Descriptors that identify the logical operations or resources a client may use, their inputs and outputs, as well as their Pre-Conditions and Post-Conditions. Such Blueprints should enable client developers and non-engineering personnel to understand the capabilities of a service, and how to use them.

Blueprints must be described through a formal Service Descriptor file structures specified by blueprints using different formats (e.g. Yaml, Json, XML, …). The Service Descriptor built by blueprints are produced using an approach called "Contract-First ». The "Contract-First" approach, also known as Design by Contract (DbC) philosophy encourages PoP users to focus on the service Interface and client interactions first rather than the service's implementation.

ARtwin PoP Services, Components and Platforms are following the RFB recursive approach. They use a declarative approach to automation, meaning users simply define the **target state** of the project components (RFBs or Services) as well as target infrastructure to be deployed which, at the push of a button, is automatically provisioned by RDCL studio tool.

Within RDCL and PoP, the target state is captured in the *"service definition"*, which is divided into two layers:

- First, the **Service Blueprint**: this an environment agnostic specification used to define the service and platform topology and configuration of your Service deployment. This provides an abstraction layer over the underlying infrastructure, allowing you to define a single Blueprint for all environments.
- Second, the **Deployment Model**: this maps the Service Blueprint to the target execution environment and infrastructure and overlays the environment specific configurations

For each platform instance, we create a separate platform model to define all the environment specific configuration information. This approach provides a number of benefits, including:

- **Consistency** - Configuration across all execution environments, from Development through to Production, is managed through a centralized *Platform Blueprint* declaration file enriched with metadata.
- **Infrastructure Independence** - Enabling you to provision consistent platforms and services across all environments regardless of infrastructure type, on premise and on cloud
- **Reliability** - Through *automation,* PoP will reduce risk by bringing platforms to a desired state no matter what the initial state of the target platform is.
- **Flexibility** - Environment specific overrides can be applied through *Platform Models*, allowing for configuration differences to be safely managed.
- **Governance** - Platform Blueprint and Platform Models are *version controlled*. Release Pipelines provide complete control in promoting configuration changes across environments.
- **Simplicity** - Easy-to-use graphical editor (RDCL) for defining target state of Platform deployment configuration. There is no need to write scripts or have specialist skills in a programming language. It is using widely used declarative file format like Yaml or Json.

2.4.3 Blueprint Creation

There are four ways in which you can create a Platform Blueprint:

- **Wizard:** This guide, through RDCL Studio, the user through a simple process to capture the key design decisions for the platform and domain component topology and configuration and creates a corresponding Platform Blueprint.
- **Template:** Choose from a set of pre-defined RFBs certified templates to build your blueprint or use a template that you have previously created.
- **Introspection:** Use artefacts in PoP software repositories, also called version control, to discover and introspect an existing Platform Domain and create a corresponding Platform Blueprint.
- **Programming:** Use PoP REST API to programmatically produce and deliver Platform, Domain and RFBs.

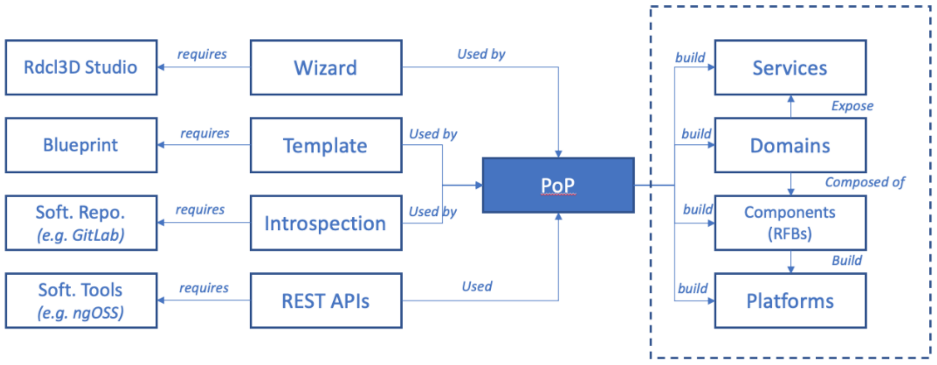


Figure 23 - Blueprints and Platforms

2.4.3.1 Using the RDCL Studio Wizard

RDCL Studio provides a wizard which guides the user through the end-to-end process of designing your Platform Blueprint. The wizard is used to capture key design decisions, such as:

- Which Execution Environment, version and components do want in your platform?
- What topology do you want?
    - A RFB composition that forms a separate multi-service platform layer?
    - A RFB composition that forms a separate service per product component?
    - An isolated monolithic component composes of RFBs?
- What constraints do you want to add to your platform?
    - For example, what is: The minimum and maximum cluster size?
- What are the performance requirements, Network, Bandwidth, number of cpu, memory?
- The target execution environment (virtual machines, containers, SoC, FPGAs, …)
- What additional platform capabilities do you want to configure? For example:
    - Do you want encrypted listen addresses for your exposed Services?
    - Do you want to persist Logs and Stores to the database or file system?
    - Do you require internal or external data sources for domains?

Based on the answers provided, PoP will generate compliant Platform Blueprint which can then be used to install and configure your Platform.

2.4.3.2 Using a Pre-Defined Template

Platform of Platforms comes with a catalog of certified Platform Templates, with "built-in" best practices for enterprise deployment. We can use these to create our initial Platform Blueprint.

Platform of Platforms also allows us to create our own Templates, which can be used to create our initial Service or Platform Blueprint. This can be useful if we have multiple domains in Production for the same Product Components. For example, we may have multiple Platform domains each designated to a particular line of business. Platform Templates allows us to capture a base "Platform Blueprint" or "Service Blueprint" that can be re-used to create the Blueprint for each domain.

2.4.3.3 Using Introspection

There are scenarios where we might want to reverse-engineer our platform or service blueprints from pre-existing environments. For such situations, PoP allows one to introspect the platform and discover the existing configuration and create a blueprint out of it.

2.4.3.4 Using REST API

This option allows to create service and Platform component using application programmable interfaces. It allows external system to build components using programming languages of choice.

2.4.4 AR Services Blueprints

2.4.4.1 AR Services Identification

In order to ease the adoption of the ARtwin platform, the services definition will follow the specification of the ETSI industry Specification Group called Augmented Reality Framework. This ISG has recently published the specification of a framework targeting Augmented Reality application [ETSI20]. By defining the functions and sub-functions of an Augmented Reality system as well as the relations between the functions (reference point), this architecture offers a good starting point to define the AR/VR services to be implemented into the ARtwin platform (see Figure 24).

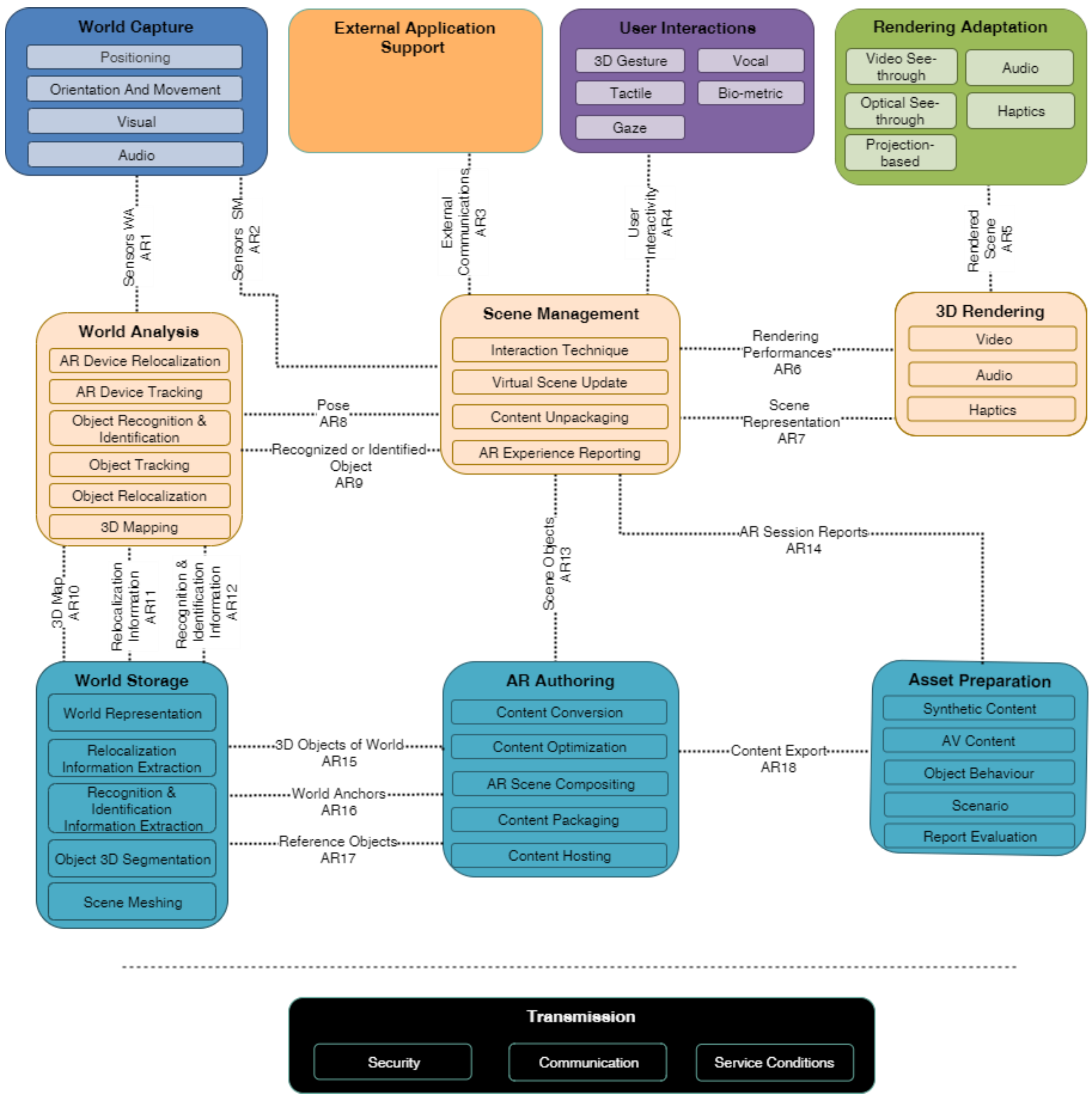


Figure 24: Functional architecture of an Augmented Reality system published by the ESTI/ ISG Augmented Reality Framework (copyright ETSI).

In addition, the specification document provides an implementation example of the architecture illustrating an ARcloud solution in the context of a factory inspection use case (see Figure 25). Similar to the use case reported by Siemens in the ARtwin project, it involves the reconstruction, updating and sharing of a 3D map of the factory used by the AR devices to locate themselves in order to display in real-time the information produced by the wide variety of sensors located into the factory. It involves two kind of AR device, a first one with high computing capabilities, and a second one with low computing capabilities requiring to move some them on the edge cloud (3D rendering, relocalization, mapping, etc.).

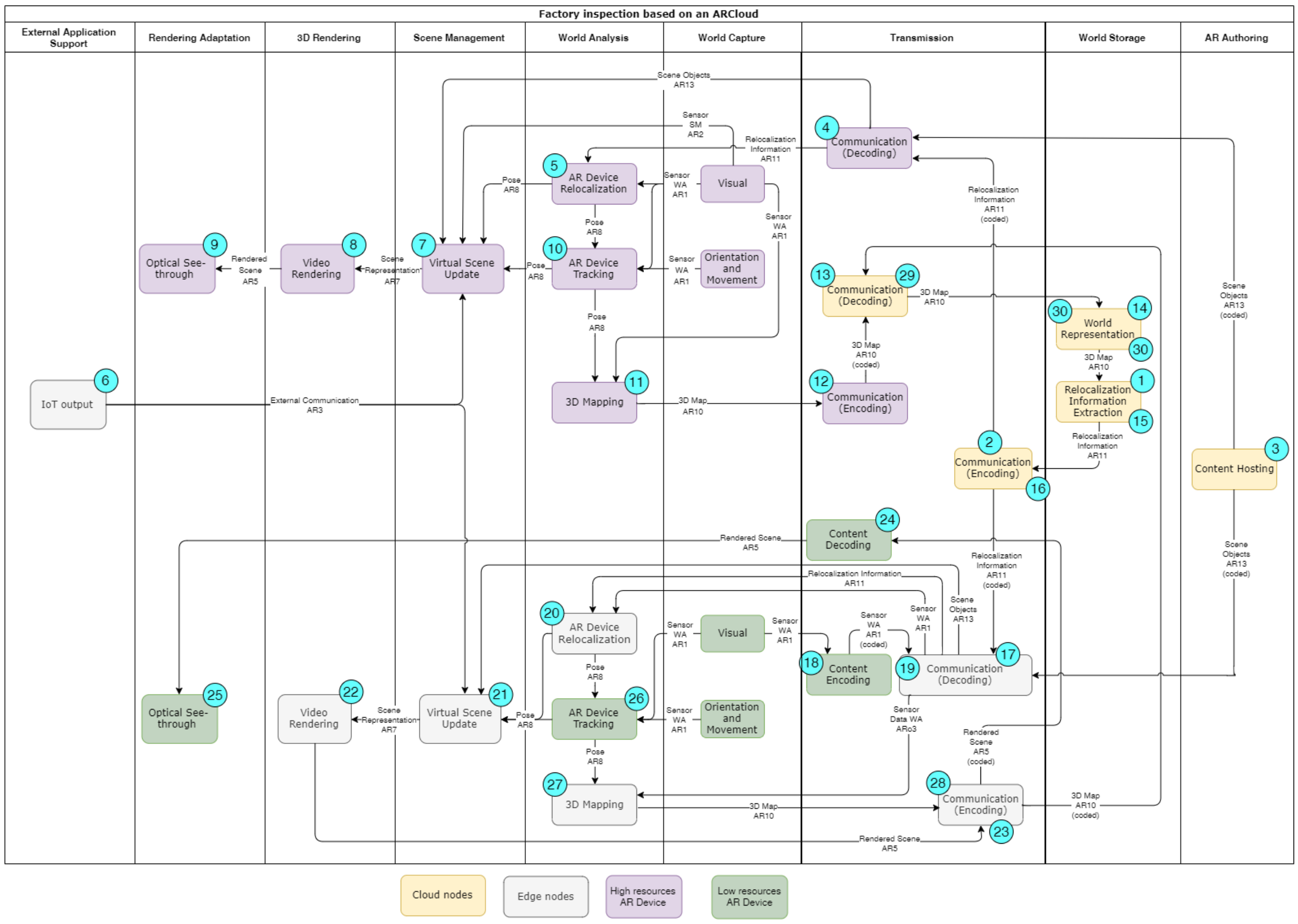


Figure 25: Workflow diagram of the subfunctions implemented for the use case "Factory inspection based on ARCloud" (copyright ETSI).

According to the ETSI specification and the ARtwin requirements, the following AR service blueprints have been identified:

- World Representation and relocalization information extraction
- Encoding
- Decoding
- Visual capture service
- AR device relocalization
- AR device tracking
- Sparse 3D mapping
- Dense 3D mapping
- Object 3D segmentation
- CAD model correction
- User interaction
- Virtual Scene Update
- User interaction
- AR Remote Rendering App

- AR content hosting

2.4.4.2 Blueprints Specifications

The following template has been proposed to describe AR services:

Function: description of the function performed by the service

Input: input object of the function

Output: output object of the function

Data Format: the format used for representation of the data

Metadata: any labels or data about the input.

Protocol: TCP, UDP, etc.

SLA: size, throughput, rate, response time, number of files/sec, all other capacity information using key: value format (ex min-throughput: 100 kBps)

Behavior: stateless/statefull, resilience, scalability, etc.

The details of these service blueprints are provided in the inventory section 3.1.1.

2.4.5 AR Platform Blueprints

2.4.5.1 AR Platforms Identification

The following inventory of AR platforms is detailed in section 3.2:

- AR/VR Service Creation Platform
  - ARCloud Service Creation Platform
  - Remote Rendering Service Creation Platform
- Telecom Platform
- BIM & PLM Platforms

2.4.5.2 Blueprints Specifications

The specification of these blueprints is an ongoing work that is carried out in WP3 and will be provided later in appropriate deliverables.

2.5 Overall Methodology

This section gives an overview on the interactions between the different provided definitions.

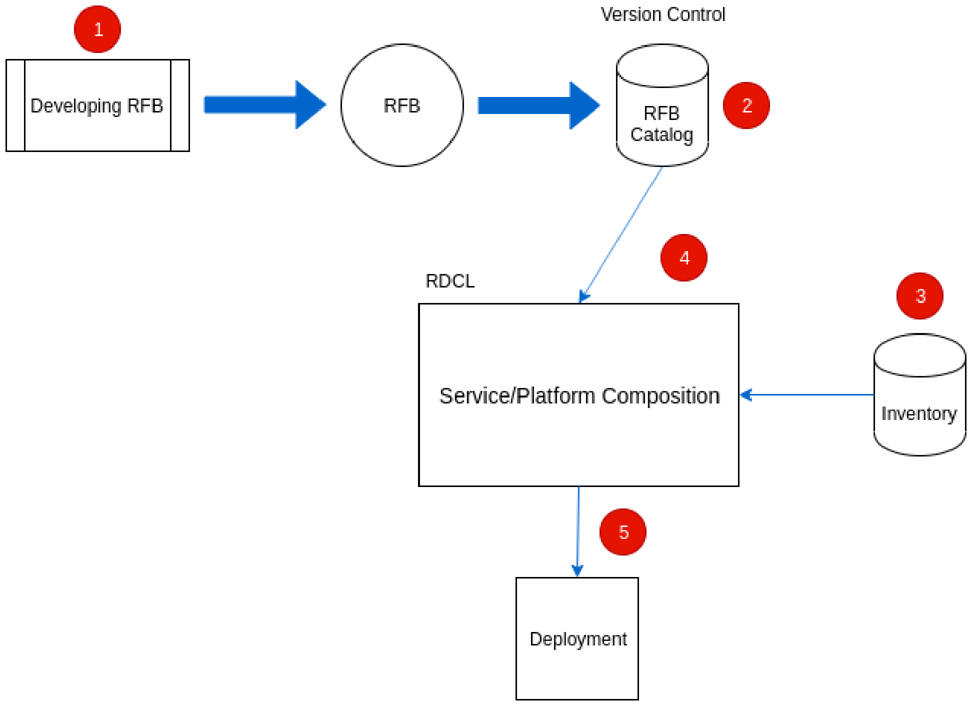


Figure 26 - Platform Composition and Deployment Workflow

Figure 26 illustrates the different steps and assets included in the process of developing, composing or editing, and deploying a platform or a service. At the very first stage is the process of developing the elementary components to be used later in service or platform composition. The output of this step is one or several RFBs that are stored in a version control system (e.g. Git). The set of RFBs stored in a repository implements a catalog of RFBs.

On the other hand, step 3 consists of building inventories of infrastructure resources that would serve as Execution Environments for deploying platforms. This is done within the RDCL tool. An inventory is a composition of infrastructure resources (hosts, VMs, etc.) in the same way a platform is composed of RFBs. Since the development of RFBs and the composition of platforms is decoupled from the environment on which they are intended to run, step 3 can be done before or after step 4.

Once the RFBs are available in the catalog, platform composition can start. The GUI of RDCL provides access to the list of available RFBs and allows dragging-and-dropping them to form a graph that describes the platform composition and the relations between the components. This step is the final one in the edition view.

The last step starts with the deployment. After the platform is composed, we can specify its Execution Environment and deploy it. The RDCL initiates the deployment and creates an event in a Message Queue system. At this stage, the PoP in charge of operating and supervising the operational states of platforms intervenes. It executes the deployment, keeps track of the operational states, and reacts in runtime to changes in those states. If there is any error in the deployment process, it is sent and displayed by RDCL. Otherwise, the RDCL shows success of deployment and the platform can be considered up and running.

3 Inventory

This section performs the most exhaustive inventory of various artefacts found in ARtwin system, from services to platforms, execution environments and deployments.

3.1 Services Inventory

3.1.1 ARCloud Services

3.1.1.1 World Representation and relocalization information extraction service

Function: The service shall maintain, update, and optimize a global 3D map of the real environment to be shared with other AR users.

Input: Local maps produced by the various AR devices connected to the ARtwin platform.

Output: The relocalization information used by the AR device requesting them to estimate its pose (position and orientation) in the real environment.

Data Format: Internal format based on the ARtwin map pivot format specification described in the deliverable D2.3 “Map Pivot Format Specification”.

Metadata: Timestamp on data.

Protocol: HTTP/2 (gRPC)

SLA: Huge storage capacities (1TB) with a response time of less than 1s.

Behavior: Run on the CPU. Using a huge database well-structured to fasten relevant data extraction, it will require synchronization in case of multiple instances and cannot restart or move easily.

3.1.1.2 Encoding services

Function: The service shall encode data to reduce its size in order to transmit it on a remote node of the cloud.

Input: Raw data (3D map, image, 3D model, localization information).

Output: Encoded data (3D map, image, 3D model, localization information).

Data Format: To be defined according to the data and the requirements in terms of compression ratio and encoding/decoding time requirements.

Metadata: Timestamp on data.

Protocol: To be defined. A first implementation could use the WebRTC framework.

SLA: For images, very-low latency (~3ms), and a throughput adapted to the targeted function and device (resolution of the tracking camera or of the AR displays).

Behavior: Can benefit of optimization provided by dedicated chip processing (ASIC, FPGA, GPU). Can be restarted on failure and scaled horizontally in load-balancing mode.

3.1.1.3 Decoding service

Function: The service shall encode data to recover original data in order to process it.

Input: Encoded data (3D map, image, 3D model, localization information).

Output: Raw data (3D map, image, 3D model, localization information).

Data Format: To be defined according to the data and the requirements in terms of compression ratio and encoding/decoding time requirements.

Metadata: Timestamp on data.

Protocol: To be defined. A first implementation could use the WebRTC framework.

SLA: For images, very-low latency (~3ms), and a throughput adapted to the targeted function and device (resolution of the tracking camera or of the AR displays).

Behavior: Can benefit of optimization provided by dedicated chip processing (ASIC, FPGA, GPU). Can be restarted on failure and scaled horizontally in load-balancing mode.

3.1.1.4 Visual capture service

Function: The service shall deliver the images (RGB, depth, RGB-D) produced by AR device sensors.

Input: Nothing

Output: Images produced by the AR device.

Data Format: An array of pixels.

Metadata: Timestamp on data.

Protocol: WebRTC.

SLA: Low-latency and response time (<5ms).

3.1.1.5 AR device relocalization service

Function: The service shall deliver the pose (position and orientation) of the device in a world coordinate system at present time without any prior knowledge of the pose of the device. To do it, the service can use data captured by the AR device sensors (e.g. images, GPS, or ultra-wide band signals). For the ARtwin project, images captured by tracking camera (RGB, depth or RGB-D) will be the first choice.

Input: Relocalization information and data produced by the AR device sensors, sent by the ARCloud tracking function (mainly images, local maps).

Output: The pose (position and orientation) of the AR device in relation to the global 3D map of the real environment.

Data Format: For images (RGB, depth, or RGB-D), an array of pixels. For local maps, the set of 3D points in local coordinate system of AR device. The relocalization information will have a format based on the deliverable D2.3 “Map Pivot Format Specification”. The pose will be realized by raw data, i.e., Matrix 4x4 or two vector3.

Metadata: Timestamp on data.

Protocol: WebRTC first for images, HTTP/2 (gRPC) for the relocalization information.

SLA: Some implementations of the service will require low-latency and response time (<50ms).

Behavior: Can benefit of optimization provided by dedicated chip processing (CPU, GPU, VPU). Can be restarted on failure and scaled horizontally in load-balancing mode.

3.1.1.6 AR device tracking service

Function: This service shall deliver the pose (position and orientation) of the device in a world coordinate system at present time based on a prior knowledge built with previous frames (e.g., the pose of the AR device corresponding to the previous frame, a 3D structure reconstructed during the previous frames). It can analyze the motion between the current data (e.g., RGB-D images) captured by the AR device and the data previously captured (e.g. based on optical flow or feature matching).

Input: The pose of the device at the previous frame or by the relocalization service, data produced by the AR device sensors (mainly images and accelerations).

Output: The pose (position and orientation) of the AR device in relation to the global 3D map of the real environment, and the images used for the tracking which can be forward to the AR device relocalization, object relocalization and 3D mapping services.

Data Format: For the pose, raw data (Matrix 4x4 or two vector3). For acceleration, raw data. For images (RGB, depth, or RGB-D), an array of pixels.

Metadata: Timestamp on data.

Protocol: WebRTC first for images, HTTP/2 (gRPC) for the AR device pose and acceleration.

SLA: Very low-latency and response time (less than 3ms).

Behavior: Can benefit of optimization provided by dedicated chip processing (CPU, GPU, VPU). Internal data can be stored meaning that the service is dedicated to an AR device instance and cannot be easily restarted on failure (requires data recovering). AR device tracking are generally provided by commercial Augmented Reality SDKs (e.g. Hololens, ARKit, or ARCore).

3.1.1.7 Sparse 3D mapping service

Function: This service shall deliver a sparse 3D representation of the real world based on a set of 3D features (e.g., points, edges). The features-based 3D representation can be constructed from data captured by one or more sensors. The pose and calibration parameters of these sensors should be known, and can be provided by tracking service.

Input: Data produced by the AR device sensors sent by the AR device tracking function (mainly images) with the associated pose of the AR device.

Output: A sparse 3D map of the real environment which can be used for relocalization.

Data Format: For the pose, raw data (Matrix 4x4 or two vector3). For images (RGB, depth, or RGB-D), an array of pixels. For the 3D map, an internal format based on the ARtwin map pivot format specification described in the deliverable D2.3 “Map Pivot Format Specification”.

Metadata: Timestamp on data.

Protocol: WebRTC first for images, HTTP/2 (gRPC) for other data.

SLA: To be defined.

Behavior: Store an internal representation of the 3D map in relation to a dedicated instance of an AR device requiring to be reloaded at each processing.

3.1.1.8 Dense 3D mapping service

Function: This service shall deliver a dense 3D representation of the real world based on a set of 3D features (e.g., points, edges). The features-based 3D representation can be constructed from data captured by one or more sensors potentially embedded in a sparse 3D map. The pose and calibration parameters of these sensors should be known, and can be provided by tracking service.

Input: RGB or RGB-D images produced by the AR device sensors sent by the AR device tracking function with the associated pose of the AR device, or a sparse 3D map including the images used to produce the 3D map with their pose and calibration parameters.

Output: A dense 3D map of the real environment.

Data Format: For the pose, raw data (Matrix 4x4 or two vector3). For images (RGB, depth or RGB-D), an array of pixels. For the 3D map, an internal format based on the ARtwin map pivot format specification described in the deliverable D2.3 “Map Pivot Format Specification”.

Metadata: Timestamp on data.

Protocol: WebRTC first for images, HTTP/2 (gRPC) for other data.

SLA: Offline processing, from several seconds to several hours depending of the scale and resolution of the dense map.

Behavior: Store an internal representation of the 3D map in relation to a dedicated instance of an AR device requiring to be recovered at each processing.

3.1.1.9 Object 3D segmentation service

Function: This service shall deliver a set of 3D objects which have been segmented from the global world representation built by the dense 3D mapping service.

Input: A dense 3D map including the images (used to produce the 3D map with their pose and calibration parameters) and CAD models of segmented objects.

Output: A dense 3D map including label on the segmented features.

Data Format: An internal format based on the ARtwin map pivot format specification described in the deliverable D2.3 “Map Pivot Format Specification”.

Metadata: Timestamp on data.

Protocol: HTTP/2 (gRPC) for other data.

SLA: To be defined.

Behavior: Can benefit of optimization provided by dedicated chip processing (ASIC, FPGA, GPU). Can be restarted on failure and scaled horizontally in load-balancing mode based on a parallelization of the 3D segmentation using a partitioning of the 3D map.

3.1.1.10 CAD model correction service

Function: This service shall compare a segmented 3D map with the CAD model extracted from a BIM or PLM system to identify which elements of the CAD model are not well-positioned, not present in the real environment but present in the CAD model, or inversely present in the real environment but not in the CAD model.

Input: A CAD model and a segmented 3D map.

Output: A list of elements which has been detected as moved, added or suppressed.

Data Format: For the input CAD model, a format widely used by industry and construction (STEP, IFC). For the input 3D map, an internal format based on the ARtwin map pivot format specification described in the deliverable D2.3 "Map Pivot Format Specification". For the list of elements, a format based on the ARtwin map pivot format specification described in the deliverable D2.3 "Map Pivot Format Specification".

Protocol: HTTP/2 (gRPC)

SLA: Not computing in real time with a high frequency, does not require low-latency.

Behavior: Can benefit of optimization provided by dedicated chip processing (CPU, GPU, VPU). Can be run once, which requires downloading both CAD models and 3D maps before processing.

3.1.1.11 CAD model registration service

Function: This service shell retrains the neural network for 3D object segmentation to be able detect new and corrected CAD models using new annotated data (e.g., images, segmented voxel grids, CAD models form BIM or PLM systems) produced by users or system administrators.

Input: A neural network for 3D object segmentation, new annotated data (e.g., images, segmented voxel grids, CAD models form BIM or PLM systems).

Output: A retrained neural network for 3D object segmentation, url of learned model (format, e.g. ONNX).

Data Format: For the input CAD model, a format widely used by industry and construction (STEP, IFC). For annotated data, multiple RGB images with camera poses and labeled object, 3D map features of the object (points, lines, mesh) based on the ARtwin map pivot format specification described in the deliverable D2.3 "Map Pivot Format Specification". For the neural network for a list of weights related of network connections.

Protocol: HTTP/2 (gRPC) for other data.

SLA: Not computing in real time with a high frequency, does not require low-latency.

Behavior: Can benefit of optimization provided by dedicated chip processing (CPU, GPU, VPU). Can be run once, which requires downloading both CAD models, annotated data and 3D maps before processing.

3.1.1.12 AR Content Management Service (HoloLayer)

Function: This service provides APIs for attaching and retrieving AR annotations (e.g. 3D primitives, voice or video messages, views into IOT backend systems, etc.) at specific spatial locations – also providing search and filter functions. The service also provides user management to allow/disallow a user access to certain API or content. In an extension the service also provides support for identifying and recognizing labels

Input: PUT or GET command at the service API, for PUT commands also authorization token

Output: For PUT commands: SUCCESS or ERROR message, for GET commands: response in the target data format of the respective annotation

Data Format: The data formats of the command's payload correspond to the data type of the annotation.

Protocol: HTTPS

SLA: Response time must be in the range of ~1 second to enable interactive applications

Behavior: Can benefit of a scaling strategy to support multiple concurrent users. Can connect to other services

3.1.2 Remote Rendering Services

3.1.2.1 AR Remote Rendering App Service

Function: Use-case dependent AR App where the Remote Render Plugin is used. This enables the App (Image Stream) to be streamed from a powerful Render Server to a SLAM capable AR Device. The Input from the SLAM capable AR Device gets streamed continuously to the Render Server. The app logic including the up-streamed Input from the AR Device is fully executed on the Render Server. So, no Data other than the Rendered Camera View is sent to the AR Device (Figure 27).

Input:

- Device Position (float32, float32, float32)
- Device Rotation (float32, float32, float32, float32)
- Camera Stream (720p, 1080p,…)
- Gesture Input (List<Vector3>, List<Vector2>, Events,…)
- Voice (open Research Task)
- Eye Gaze (open Research Task)

Output:

- Stereoscopic Image Stream (2x 1080p,…)
- Additional Output: Audio (open Research Task)

Required Components:

- Slam Capable AR Device: Remote Render Client
- Rendering Server: App incl. Remote Render Plugin
- Orchestration System: Signaling Server

Execution Environment:

  - SLAM Capable AR Device: Universal Windows Platform, Andoid, iOS
  - Rendering Server:
    - Windows 10 with Nvidia Geforce
    - VM with Windows 10 Docker Container and Nvidia Tesla with Nvidia Grid
- **Multiple Instances behavior**
  - Per SLAM Capable AR Device one Instance of the app is needed (Figure 28)
  - Multiple Apps per VM Instance is possible

- Instance can be switched by one click (change IP Address where it streams Device Information)
- load-balancing can be achieved by limiting the bandwidth by decreasing frames or resolution

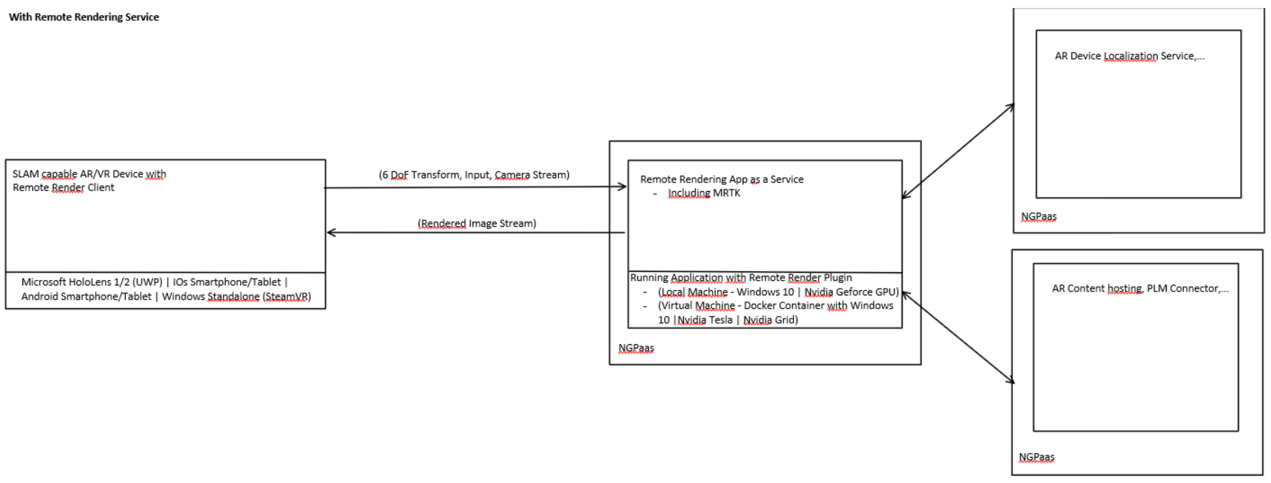

With Remote Rendering Service
SLAM capable AR/VR Device with Remote Render Client
Microsoft HoloLens 1/2 (UWP) | IOs Smartphone/Tablet | Android Smartphone/Tablet | Windows Standalone (SteamVR)
(6 DoF Transform, Input, Camera Stream)
(Rendered Image Stream)
Remote Rendering App as a Service
- Including MRTK
Running Application with Remote Render Plugin
- (Local Machine - Windows 10 | Nvidia Geforce GPU)
- (Virtual Machine - Docker Container with Windows 10 | Nvidia Tesla | Nvidia Grid)
NGPaas
AR Device Localization Service,...
NGPaas
AR Content hosting, PLM Connector,...
NGPaas

Figure 27 - Remote Rendering scheme. Remote Render client connects with Remote Render app, which is running on the edge node. This app connects to other services like localization or content hosting.

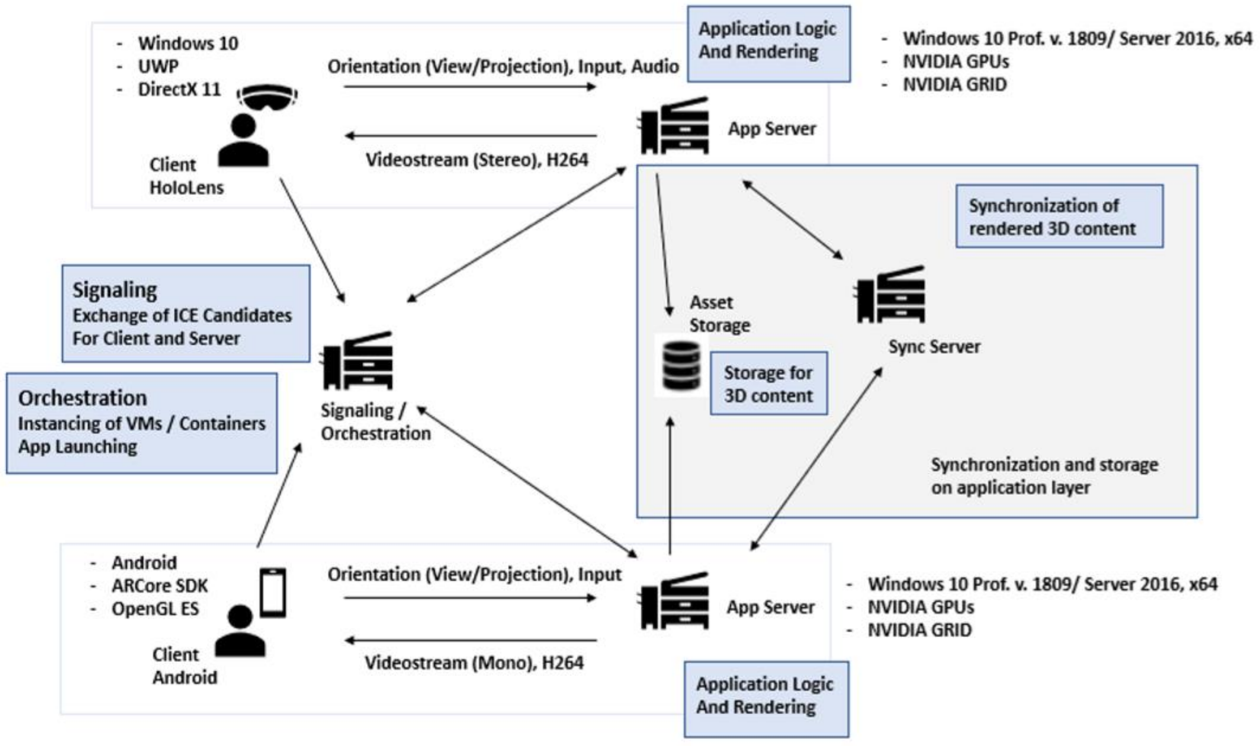

- Windows 10
- UWP
- DirectX 11
Client HoloLens
Orientation (View/Projection), Input, Audio
Videostream (Stereo), H264
Application Logic And Rendering
App Server
- Windows 10 Prof. v. 1809/ Server 2016, x64
- NVIDIA GPUs
- NVIDIA GRID
Signaling
Exchange of ICE Candidates For Client and Server
Orchestration
Instancing of VMs / Containers
App Launching
Signaling / Orchestration
Asset Storage
Storage for 3D content
Synchronization of rendered 3D content
Sync Server
Synchronization and storage on application layer
- Android
- ARCore SDK
- OpenGL ES
Client Android
Orientation (View/Projection), Input
Videostream (Mono), H264
App Server
- Windows 10 Prof. v. 1809/ Server 2016, x64
- NVIDIA GPUs
- NVIDIA GRID
Application Logic And Rendering

Figure 28 - Advanced Edge Cloud system having multiple edge server with combined communication.

3.1.2.2 User interaction service

MRTK-Unity is a service-oriented opensource-driven library that provides a set of UI/UX components and features, used to accelerate cross-platform MR app development in Unity. Here are some of its functions:

- Provides the basic building blocks for Unity development on HoloLens, Windows Mixed Reality, and OpenVR.
- Enables rapid prototyping via in-editor simulation that allows you to see changes immediately.
- Operates as an extensible framework that provides developers the ability to swap out core components.
- Supports a wide range of platforms, including:
    - Microsoft HoloLens 1 & 2
    - Microsoft HoloLens 2
    - Windows Mixed Reality headsets
    - OpenVR headsets (HTC Vive / Oculus Rift)

Function: Mapping Device Inputs for cross-platform app development for AR/VR devices. This library needs to run in the app. (local at the device or on the remote render server)

Input: All kind of device inputs (specified for each device individually in the input service of MRTK)

Output: Interfaced Input Events. These interfaces will be called by the MRTK input system to handle custom app logic based on user input interactions.

(list see in https://microsoft.github.io/MixedRealityToolkit-Unity/Documentation/Input/InputEvents.html)

3.1.2.3 AR content hosting

Function: 3D Content Storage

Input: Request for a specific File (JSON)

Output: 3D CAD/BIM File

Data Format: JT

Behavior: A user needs for his Use-case a specific 3D CAD/BIM file. This can be done manually by selecting a file from File browser which is looking on a specific “network path” or automatically in the background, because different service is requesting a specific CAD/BIM file.

Remark: Content storage can also be Siemens Teamcenter for CAD or Autodesk Vault/Forge for BIM.

3.1.2.4 Remote Render Orchestration Service

Function: Provide proper resources to a user, which requests Remote Render App Service

Input: Request for a specific app

Output: IP address / Port of the Remote Render Server running the specific app

Data Format: JSON

Behavior: A user needs for his Use-case a Remote Render App, because data visualization requirements exceed the available performance of the mobile AR device. Therefore, a user is asking the orchestration service to connect him with an available Remote Render Server.

3.1.2.5 Shared Assets (Multiuser) Meta-Data Service

Function: Service which hosts and distribute non-permanent user generated data.

Input: Change Request for specific meta data

Output: Changed Data

Data Format: JSON

Behavior: For example, two users want to review a CAD object in more detail and therefor want to disassemble the object. Therefore, one user is starting to grab and place objects from its origin to another position. This information has to be streamed from one instance to another instance. This can be done by a separate service, which takes care about data synchronization.

3.1.3 Telecom Services

We will provide in ARtwin to a 5G AR/VR Service Operator ability to deploy a private 4/5G network using PoP services and Telco platforms to provide communication between several AR/VR platforms and remote sites using AR/VR devices. The service operator may own the platforms on which the 4/5G service will run or may make use of platforms provided by a Platform Operator. As the 4/5G service is composed of RAN and CORE services, each of these services can be deployed on different Platform, which in turn is running on different Execution environments and IaaS. The 5G services implements a pre 5G service on top of two customized platforms: One localized in a central cloud hosting the 5G Core and one localized in the edge and hosting the Radio Access Network (RAN). Both platforms are deployed and managed by the PoP and customized with features allowing them to run the service on top.

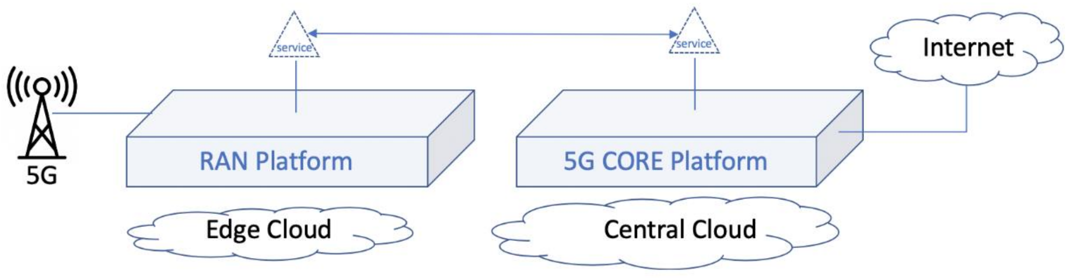


Figure 29 - Telecom and 5G Platforms

3.1.3.1 5G RAN service

For the RAN platform and services, one option for the platform is compose RFBs fom Open Air Interface open-source project with split into RCC and RRU. These radio components are deployed on the Edge platform relatively close to the antenna to provide good latency performances. In addition, RFBs and platform provide ability to locally accelerate some radio sub-part in stringent conditions by offloading to FPGA some RFBs so that the RAN service runs on an accelerated cloud.

Commercial options from NOKIA are also considered (to be completed at next release of platform specification).

3.1.3.2 5G core service

The core part of the 5G service is built from Wireless Edge Factory (WEF), developed by b<>com. This solution is a virtualized network element in charge to aggregate different network accesses (5G, LTE, Wi-Fi, fixed networks). Thanks to virtualization, it can be deployed at different locations in Operator's data center, in distributed Point of Presences (PoP) or even closer to the users, at enterprise premises for instance. The WEF is modularly designed, allowing its users to deploy the necessary network functions (NF) only. As part of the ARtwin project, only standalone 5G NF will be deployed. The WEF architecture aims to be compliant with 3GPP 5GC release 15 standards.

Following the 3GPP (System Architecture for the 5G System) the key principles to reach the goals of 5G system architecture include:

- Separating the User Plane (UP) functions from the Control Plane (CP) functions, and allowing independent distribution schemes for UP and CP functions (centralized, distributed);
- Adopting a modular function design to enable flexible network slicing;
- Defining reusable procedures for interaction between Network Functions (NFs);
- Minimizing dependencies between access and core networks: a common interface integrates different 3GPP and non-3GPP access types;
- Supporting stateless NFs, where the used resources are decoupled e.g., decoupling the compute resource from the storage resource.

Figure 30 represents the 5G service which is planned to be deployed for the Use Cases.

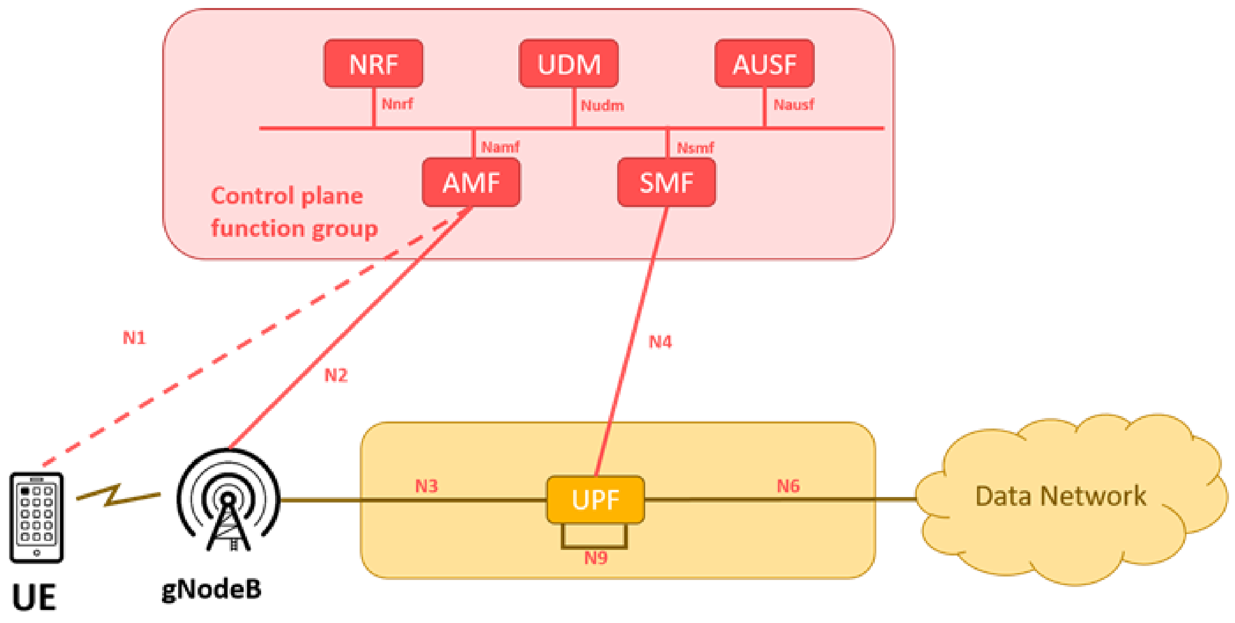


Figure 30 - 5G Core Functions

The blocks in the figure (CNFs) are:

- **AMF:** entry point in the Core Control Plane for the UE and the (NG) RAN. It provides the termination of RAN Control Plane signaling interface (N2), as well as the termination of UE signaling interface (N1), also named the NAS. The AMF supports registration, connection, reachability and mobility management, lawful intercept, access authentication. It is also used as a proxy for session management messages and their transport toward the SMF (see below). The AMF deployed in the context of ARtwin is compliant to the release 15 of the 3GPP Core Network. Its software is developed by b<>com. This component is split compliant with a microservice architecture (finer-grained loosely services architecture). It is composed of different types of containers that are managed by a Kubernetes PaaS;
- **SMF:** handles session management (establishment, modification, release), encompassing tunnel maintenance between the UPF and the AN, UE IP address allocation, selection and control of UPF, traffic steering configuration, interface toward Policy control, charging data collection, … It uses the N4 reference point to control tunnel establishment and traffic steering rules in the UPF. Session management signaling with the UE in the NAS are processed by the SMF which is not directly exposed to the N2 interface thanks to the proxy function from the AMF The SMF deployed in the context of ARtwin is compliant to the release 15 of the 3GPP Core Network. Its software is developed by b<>com. This component is split compliant with a microservice architecture (finer-grained loosely services architecture). It is composed of different types of containers that are managed by a Kubernetes PaaS;
- **NRF:** supports service registering to maintain the NF profile of available NF instances and their supported services, as well as service discovery function to allow any NF to discover what instances of others NF are available and their related information. NRF also allows other NF instances to subscribe to, and get notified about, the registration in NRF of new NF instances of a given type. The NRF deployed in the context of ARtwin is compliant to the release 15 of the 3GPP Core Network. Its

software is developed by b<>com. This component is split compliant with a microservice architecture (finer-grained loosely services architecture). It is composed of different types of containers that are managed by a Kubernetes PaaS;

- **UDM:** uses subscription data to generate user's authentication credentials, identification handling, access authorization, lawful intercept, subscription management…. The UDM deployed in the context of ARtwin is compliant to the release 15 of the 3GPP Core Network. Its software is based on free5GC. It is composed of two different types of containers: one containing the database and another one containing the software handling the logic of the service. These containers are managed by a Kubernetes PaaS;
- **AUSF:** supports user's authentication. The AUSF deployed in the context of ARtwin is compliant to the release 15 of the 3GPP Core Network. Its software is based on free5GC. It is composed of one container containing the software handling the logic of the service. This container is managed by a Kubernetes PaaS.

The Core Network (CN) user plane is supported by the **UPF**, which is connected to the NG RAN (Next Generation Radio Access Network), the Control Plane and the external data network through N3, N4 and N5 reference points.

In a 5G core network, the UPF includes the following main functionalities (extracted from the 3GPP TS 23.501):

- External PDU (Protocol Data Unit) session point of interconnect to Data Network;
- Packet routing & forwarding.

Currently, the UPF NF has its software based on OpenvSwitch (OVS) installed in a virtual machine (VM). For the initial prototype the UPF is not part of any PaaS, it will be deployed on a hypervisor, such as Openstack or KVM. In the enhancements brought in the final prototype, the target PaaS will be analysed and deployment files adapted. The UPF provides a switching service for the data path which is a wider service than provided by other Core components and which is more complex to adapt in a PaaS.

3.1.4 Infrastructure Services

The infrastructure presents several services:

Identities management. This service offers a HTTP(S) API for controlling access to Infrastructure resources. It enforces policies to those resources and returns audit logs about their usage as well. As several use cases developed by ARTwin project members should be hosted by the common Infrastructure platform, this service should support multi-tenancy feature. Finally, the ARTwin solutions will be deployed on edge and public clouds so this service should provide a central way to configure control accesses to those distributed Infrastructure platforms.

Artefacts repository. This service offers a HTTP(S) API for storing, into a unique place, all ARTwin built artefacts such as ISO image files, Docker images, Helm charts, CLI tools, binaries and so on. Different versions of an artefact can be stored and fetched later. This service can be requested from a CI/CD pipeline.

Secret management. This service provides a HTTP(S) API for creating, storing and managing any secrets used by any ARTwin services, applications, platforms or users. A secret can be a user password, an API key, a token, a certificate an encryption key and so on. This service offers a central store and deploy secrets across the distributed infrastructure platforms. A user, an application or a machine can access to a secret once the authentication is successful and the access to this secret is granted by the system.

Networking. This service allows users and/or applications to create virtual network connectivity between compute and/or storage machines, in an infrastructure platform. It provides a HTTP(S) API that allows the building of a complete virtual network infrastructure in a cloud, including L2/L3 networks, subnets, routers and so on. IPv4 address(es) of machines deployed in a such virtual network are configured dynamically via DHCP. This service offers public access to these machines as well, if required.

VPNaaS service. This cloud-based service offers the possibility of building virtual private networks (VPN) on demand. It is useful for creating site-to-site secured communication channels between machines deployed in the embedded, edge or central clouds. Thus, its HTTP(S) API gives the possibility to users and/or applications to create quickly IPSEC tunnels.

FWaaS service. This cloud-based service offers a HTTP(S) API to cloud platform administrator to control incoming and outgoing network traffic by defining firewall policies which allow or block application network traffic. FWaaS eliminates the appliance form factor and offers the simplicity and rapidity for deploying firewall policies into the embedded, edge and central clouds.

Computing service. This service provides a HTTP(S) API to provision computing machines and manage their life cycle. These machines used for running AR pipelines can be virtualized or not, according to application constraints. To operate virtual machines, this service pilots dedicated type-1 or Type-2 hypervisors which can manage the execution of several instances of a variety of guest operating systems (Linux, Windows, …) on a same host physical machine. To operate bare metal machines, this service can provision, on demand, real servers with the appropriate operating system and turn on/off theses ones. Such bare metal machines can provide dedicated computing resources for very demanding applications such as AR pipelines (virtual machines are less efficient than bare metal ones because an application running in a virtual machine has not a direct access to the physical hardware resources).

Storage service. This service offers access to data storage capabilities such as objects, blocks or files, through its HTTP(S) API. Data are stored and replicated on distributed storage nodes.

Object data allows you to manipulate application data as distinct units, called objects. An object contains the data itself, some metadata and a unique identifier. Objects are stored into a flat address space (this is not hierarchy between objects). Objects are recommended for data analytics (based on metadata), and are efficient for data retrieval and storage cost. Another way to store huge amount of data is the block storage: file data are so split into evenly sized chunks, the blocks, that are distributed amongst the storage machines. A block has no metadata but has a unique address. From computing machine, block data are accessible from a mounted volume. Block data are recommended for high level IO constraints and for data frequently modified (such as database). Finally, file storage manages data as a single piece of information in a hierarchical storage (i.e. a file into folder). The path to these data as to be known by the application. This last solution is more dedicated to legacy applications.

Observability services. To meet AR pipelines requirements, we have to focus and improve stability and performance of the distributed infrastructure platforms. These services offers to users and applications monitor metrics and dataset related to the performance of infrastructure resources. Several services can be listed here to observe the global deployed system:

- A log aggregator service aggregates, stores and analyses logs or events data from any machine and application sources, by using open protocols. It offers data visualization features to user in a central place as well.
  A time-series metrics collector service aggregates monitoring metrics at regular interval and stores them to dedicated storage system, customized specifically for time-series data. The regular interval is important for getting mathematical results consistently. Metrics comes from machines and applications as well. Additionally, this service can generate alert notifications according to user defined policies. It offers data visualization features to user in a central place as well.

3.1.5 Additional Industry Services

- Service to convert BIM Data in AR usable Data (Would fit in a construction 4.0 service)
  - Functionalities: Mesh Decimation, Mesh Optimization, Object Filter, Meta Data
- Service to convert CAD Data in AR usable Data
  - Functionalities: Mesh Decimation, Mesh Optimization, Object Filter, Meta Data
- Standard Service for “simple” reviewing of BIM/CAD Data on AR Devices
- Proxy service to connect to external (e.g. cloud) services for additional functions offered by 3rd parties

NB: More services may be added during the project, depending on the context.

3.2 Platform Inventory

This section introduces various types of platforms present in the overall architecture. It is important to remind that the complete system is a composition of several platforms, name of the “platform of platforms” is ARtwin platform. The various sub-platforms types are:

- ARcloud sevice creation
- Remote rendering service creation
- Telecom
- BIM & PLM
- IOT

In this version of the platform specification, each sub-platform provides a self-sustainable component to other sub-platforms. This component can be:

- managed by ARtwin platform operator (therefore it will be deployed and executed by ARtwin),
- or can be managed by an external platform operator (therefor it will be access via a service).

In both cases, internally or externally managed component, all components shall be operated by ARtwin platforms as one unique domain. The latter is therefore in charge of interconnecting all components.

In a second iteration, we shall introduce the notion of domains within ARtwin in order to augment ARtwin platform flexibility. In this case, functions specified in each sub-platform appear as individual components identified and operated by ARtwin platform. Therefore, the granularity of operations is smaller.

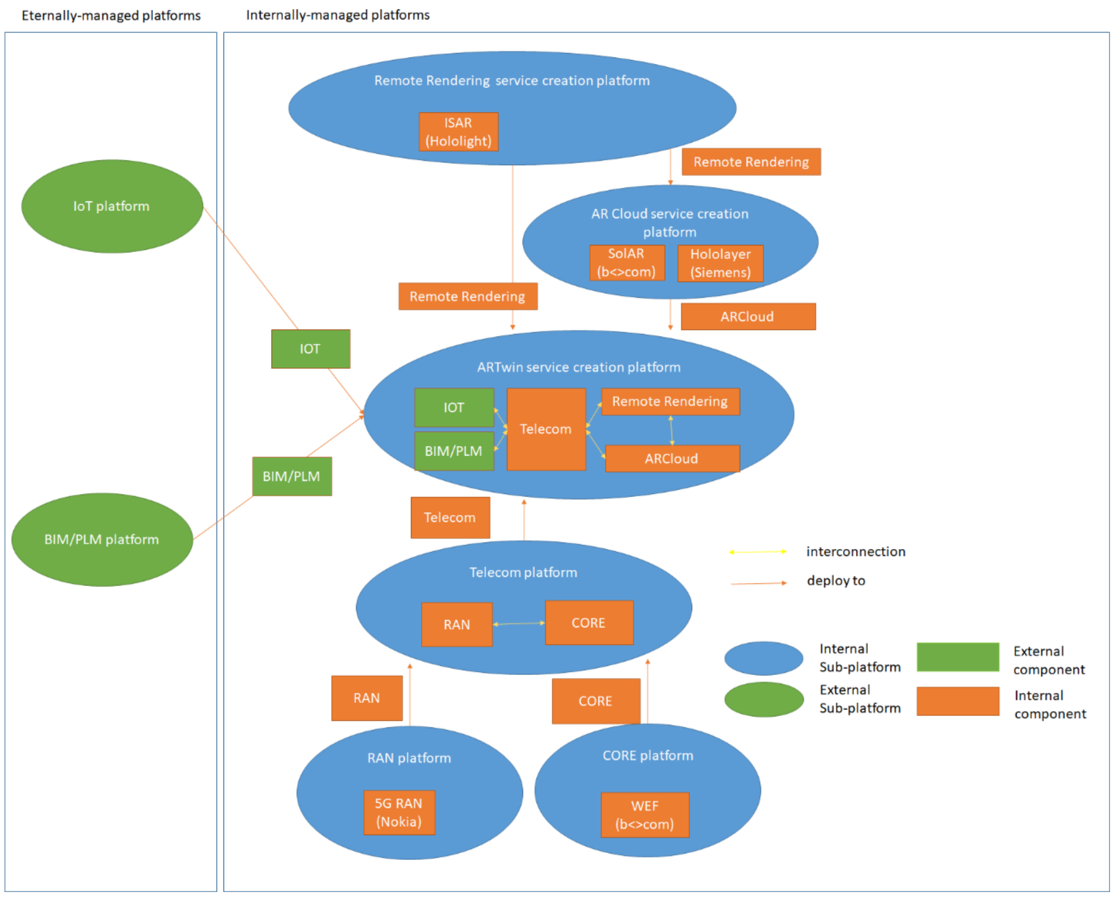


Figure 31 - ARtwin Platform and its Sub-Platforms

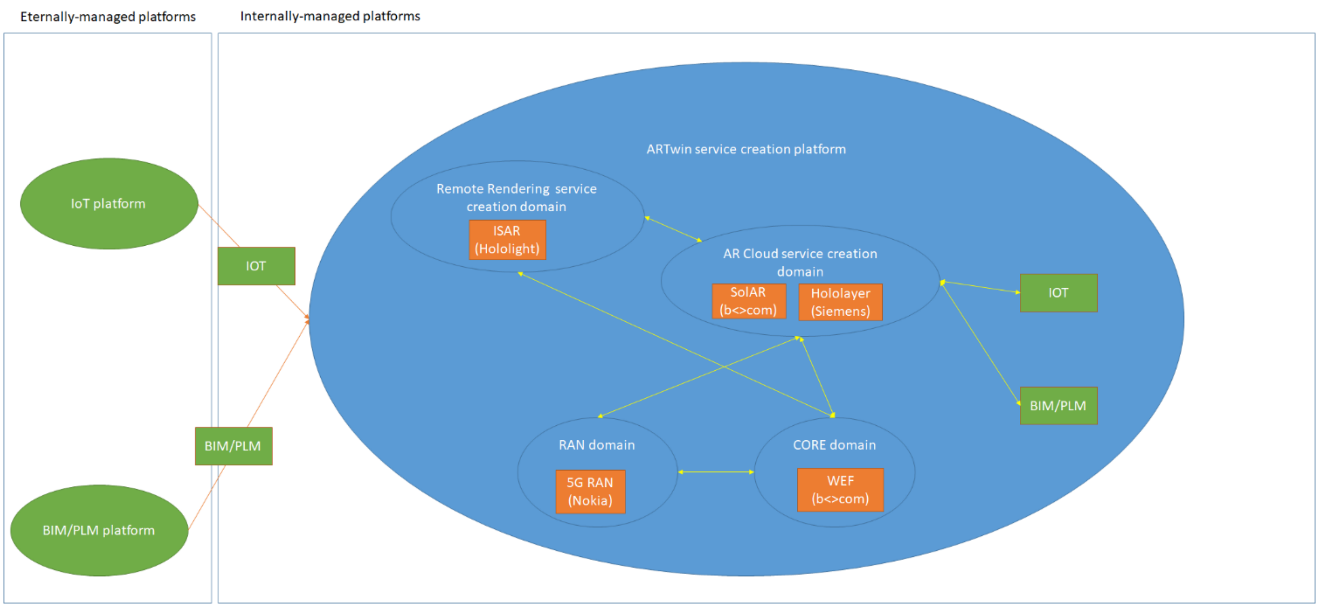

Eternally-managed platforms
Internally-managed platforms
IoT platform
IOT
BIM/PLM
BIM/PLM platform
ARTwin service creation platform
Remote Rendering service creation domain
ISAR (Hololight)
AR Cloud service creation domain
SolAR (b<>com)
Hololayer (Siemens)
IOT
BIM/PLM
RAN domain
5G RAN (Nokia)
CORE domain
WEF (b<>com)

Figure 32: ARtwin domains (for next iteration)

3.2.1 AR/VR Service Creation Platform

3.2.1.1 ARCloud Service Creation Platform

Many services rely on vision (and machine learning) pipelines that consist of a chain of low-level processing components:

- World Representation and relocalization information extraction service
- Visual capture service
- AR device relocalization service
- AR device tracking service
- Sparse 3D mapping service
- Dense 3D mapping service
- Object 3D segmentation service
- CAD model correction service

To build these vision pipeline-based services, the ARtwin plaform will benefit from the open-source SolAR framework. Indeed, SolAR is an open source framework dedicated to augmented reality (AR), and can also be used for many pipelines targeting VR applications. It offers a C++ and C# SDK to easily and quickly develop and use custom vision pipelines addressing mixed reality applications. It provides developers with a full chain from low-level vision components development to high-level computer vision pipelines and XR service development. Currently focusing on camera pose estimation, it will be extended in the context of the ARtwin project to support also dense 3D reconstruction and scene understanding.

The SolAR framework is open-source and released under Apache license 2.0, and this allows it to be used for research as well as for commercial purposes addressing various domains (smart factory, smart home, real estate, health, etc.).

SolAR aims at stimulating the interaction between all XR actors for the benefit of end users.

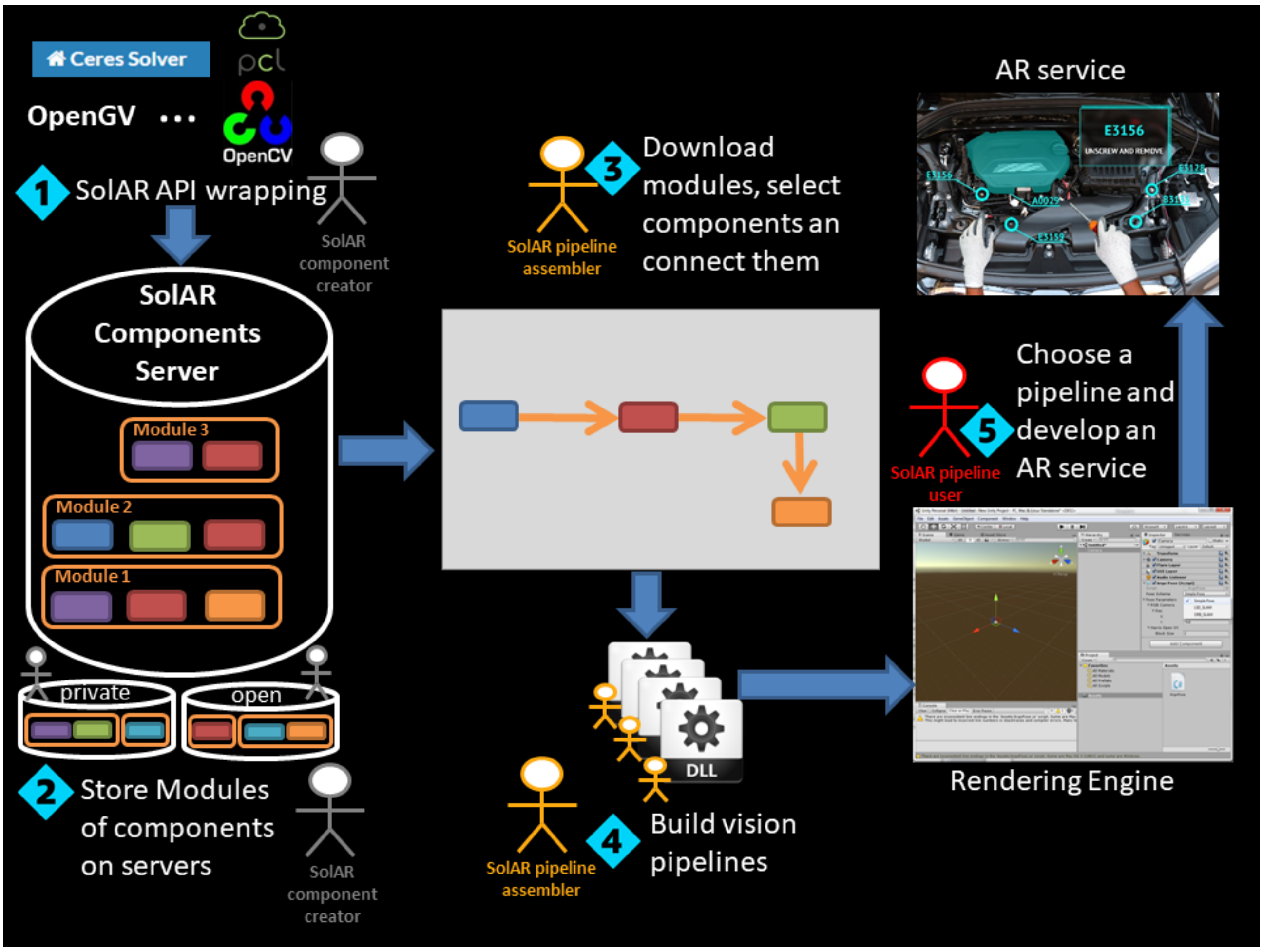


Figure 33 - Different steps and associated user roles to create an AR solutions based on the SolAR framework.

The SolAR Framework addresses the full chain of XR applications development related to computer vision (see Figure 33):

1. **Components creation:** SolAR defines a unique API for basic components required for computer vision pipeline (features extractor, descriptors calculation, matching, Perspective N Points, homography, image filters, bundle adjustment, etc.). The SolAR community can implement new components compliant with the SolAR API.
2. **Component publication:** A set of components are packaged into modules to ease their publication. SolAR modules, whether royalty free or under a commercial license, are stored on an artefact repository to be available to the SolAR community of pipeline assemblers.
3. **Vision pipeline assembling:** Modules of components published by the SolAR community can be downloaded from the module repositories. Then, SolAR provides developers with a pipeline mechanism allowing them to assemble SolAR components to define their own vision pipeline such as a camera pose estimation solution.
4. **Vision pipeline publication:** When a vision pipeline has been assembled and configured, it can be published in a repository of artefacts dedicated to SolAR vision pipelines to make it accessible to the SolAR pipeline users.

5. **XR service development:** Thanks to a SolAR plugin for Unity, XR service developers can download SolAR pipelines stored on the dedicated artefact repository and integrate them in few clicks to their applications. They can simply develop XR applications as with any AR SDK and roll them out. Since SolAR is based on a unified interface, XR application developers will be able to easily make their applications evolve with the new solutions developed by the SolAR community.

To better illustrate a vision pipeline developed with the SolAR framework, Figure 34 shows the different components involved in a SLAM solution. Each small box describes a low-level component compliant with the unified SolAR component API. Large boxes correspond to high-level pipelines which can be deployed on different nodes of the cloud (Initialization, Relocalization, Tracking, Mapping, and Optimization).

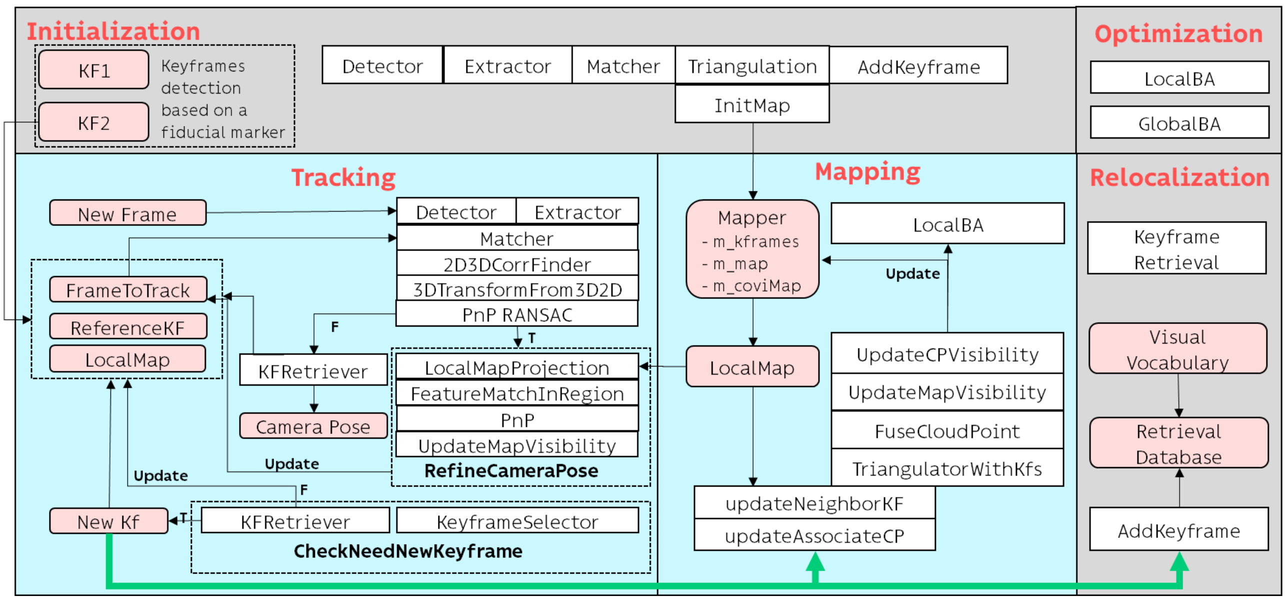


Figure 34 - A sample of a SLAM pipeline based on the SolAR framework.

The objective during the ARtwin project will be to extend the functionalities of the SolAR framework to add a new step which will consist, after the publication of the pipelines, to orchestrate them and to deploy them on the cloud.

Indeed, the SolAR framework already provides a set of data structures defining data formats used to exchange information between the low-level components and the pipelines (a description of component unified API and data structures is available on the SolAR API web page). These data structures will be extended with the map pivot format described in the deliverable D2.3 “Map Pivot Format Specification”.

XPCF (the cross-platform component framework used by SolAR) will be extended to provide additional communication features:

- distributed component ability (distribution through their interfaces)
- distributed message queues (for raw data exchange)

Those features will rely on gRPC with several message formats support (protobuf and flatbuffers to start).

This XPCF extension will allow data exchange over the network between components/pipelines.

Moreover, a tool will be provided to developers to easily create Docker images that will integrate the vision pipelines needed to develop cloud-based applications. These pipelines, ready for deployment, will be orchestrated by the AR platform based on NGPaaS.

For the moment, every part of a SolAR pipeline runs on the same device inside one process as described in Figure 35.

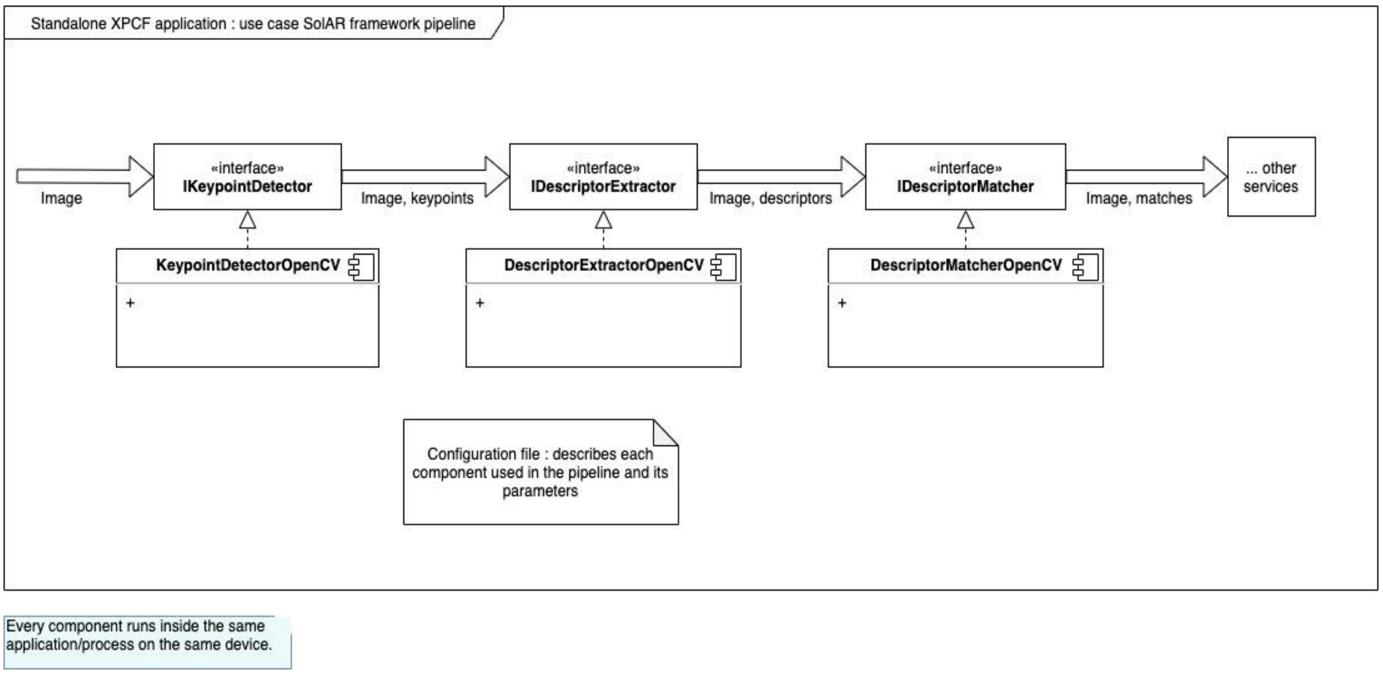

Standalone XPCF application : use case SolAR framework pipeline
Image
«interface» IKeypointDetector
Image, keypoints
«interface» IDescriptorExtractor
Image, descriptors
«interface» IDescriptorMatcher
Image, matches
... other services
KeypointDetectorOpenCV
+
DescriptorExtractorOpenCV
+
DescriptorMatcherOpenCV
+
Configuration file : describes each component used in the pipeline and its parameters
Every component runs inside the same application/process on the same device.

Figure 35 – SolAR simple vision pipeline sample

The XPCF extension aims to provide a distributed cross-platform solution. To achieve this goal, the following architecture is proposed:

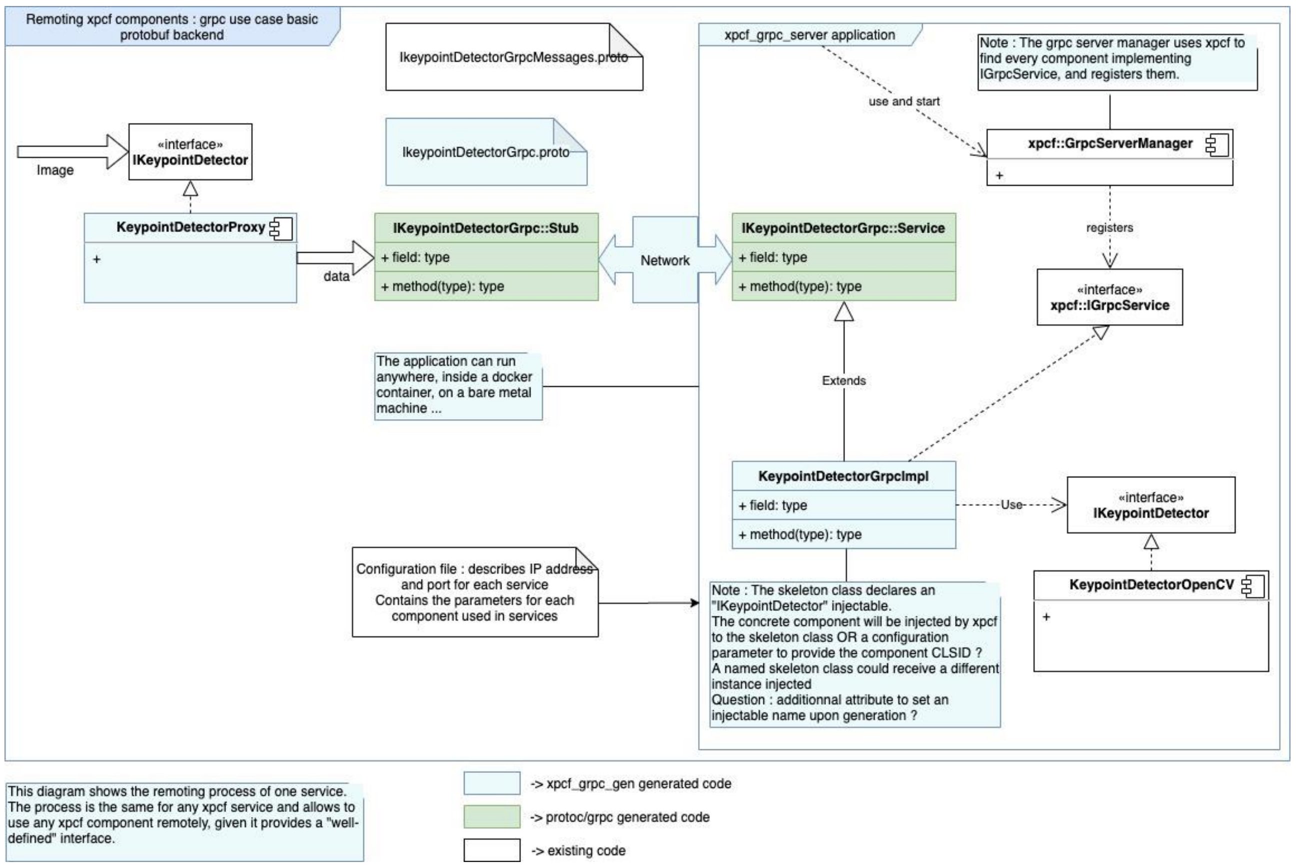

Remoting xpcf components : grpc use case basic protobuf backend
IkeypointDetectorGrpcMessages.proto
IkeypointDetectorGrpc.proto
Image
«interface» IKeypointDetector
KeypointDetectorProxy
+
data
IKeypointDetectorGrpc::Stub
+ field: type
+ method(type): type
Network
xpcf_grpc_server application
Note : The grpc server manager uses xpcf to find every component implementing IGrpcService, and registers them.
use and start
xpcf::GrpcServerManager
+
registers
«interface» xpcf::IGrpcService
IKeypointDetectorGrpc::Service
+ field: type
+ method(type): type
Extends
The application can run anywhere, inside a docker container, on a bare metal machine ...
KeypointDetectorGrpcImpl
+ field: type
+ method(type): type
Use
«interface» IKeypointDetector
KeypointDetectorOpenCV
+
Configuration file : describes IP address and port for each service Contains the parameters for each component used in services
Note : The skeleton class declares an "IKeypointDetector" injectable. The concrete component will be injected by xpcf to the skeleton class OR a configuration parameter to provide the component CLSID ? A named skeleton class could receive a different instance injected Question : additionnal attribute to set an injectable name upon generation ?
This diagram shows the remoting process of one service. The process is the same for any xpcf service and allows to use any xpcf component remotely, given it provides a "well-defined" interface.
-> xpcf_grpc_gen generated code
-> protoc/grpc generated code
-> existing code

Figure 36 – Communication layer based on GRPC and protobuf backend

Most, if not all, communication proxies and server-side helpers will be generated from xpcf interfaces definitions with a dedicated tool (xpcf_grpc_gen).

Every custom data structure will need to be serializable.
To ensure this, data structures used in interfaces will have either to:
- provide messaging definitions
- implement specific xpcf accessors

3.2.1.2 Remote Rendering Services Creation Platform

Holo-Light's Remote Rendering Service has the objective to develop a framework to process typical CAD (Computer Aided Design) models on head mounted devices, which has very limited computing capability. Such a framework enables it to collaborate during production or prototyping over large distances in real time.

Typical XR devices (e.g. Microsoft HoloLens or mobile phones) have very limited computing resources available for applications. On the other hand, the visualization of CAD models has a tremendous demand of computing power and storage.

To make it possible bringing large content onto limited devices, Holo-Light creates a framework for outsourcing the rendering process of XR devices to high-end computers. To enable such a tool, the Remote Rendering the concept is to run XR applications on high-end computers instead of limited head mounted devices and just send the application's rendered images back to the devices where the image stream then only needs to be displayed.

Remote Rendering Concepts

The basic concepts to enable Remote Rendering can be summed up as follows:

- Code Library for Remote Rendering: Holo-Light provides a code library which can easily be used as SDK/Plugin for other developers to write their own applications with remote rendering and streaming capabilities.
- Plugin for Unity game engine: The integration of remote rendering capabilities should be simple for application developers. For this we provide a Plugin for the Unity game engine with an interface written in C# language. Application developers can easily use this to develop their own XR applications with remote rendering capabilities.
- Device specific clients: Due to the variety of XR Devices it is required to write for each device a specific client which are optimized for the respective hardware platforms
- ARTwin integration: Remote rendering should be integrated into ARTwins Service oriented Architecture. The Remote Render Unity Plugin cannot be a Service. It always needs to compiled together with an application logic, where a user can interact on a meaningful way.
- Cloud/Edge infrastructure: Remote Rendering needs to run on a low latency edge/cloud infrastructure, due to the fact, that up and down stream of data need to be synchronized and the RTT should not be higher than 16ms.

Figure 37 illustrates the modules and working concepts of Remote Rendering in greater details:

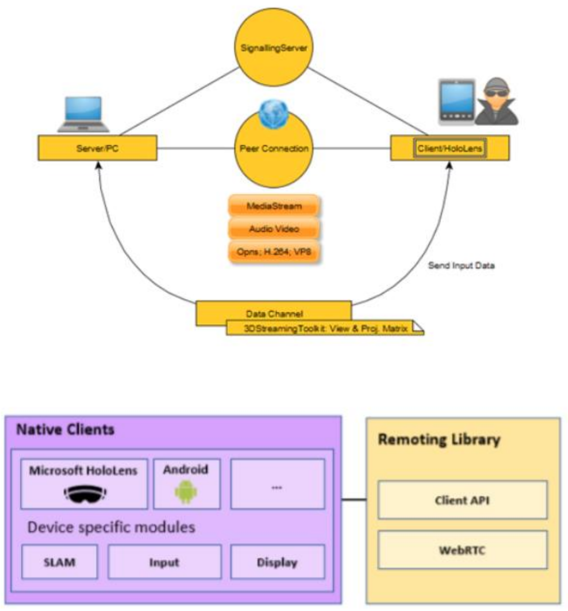


Figure 37 - Deployment setup for Remote Rendering

For each supported client platform (e.g. Microsoft HoloLens, Android, iOS, HTC Vive, ...) a platform-specific client application is built to support the platforms rendering, display and user input characteristics.

Holo-Light provides clients for the most commonly used head mounted displays and mobile platforms to ensure the best experience on XR devices with fluent frame rates and low input delays.

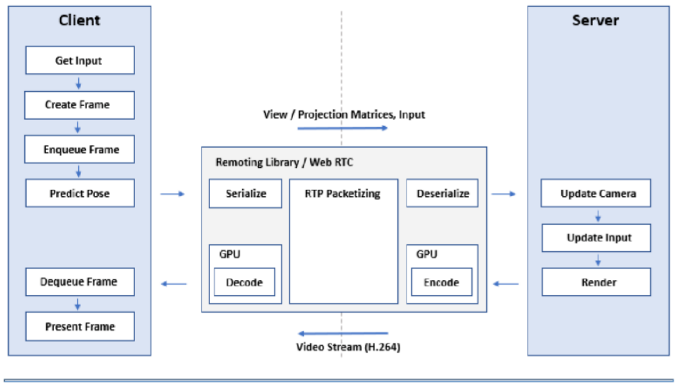


Figure 38 - Remote Render information processing High Level

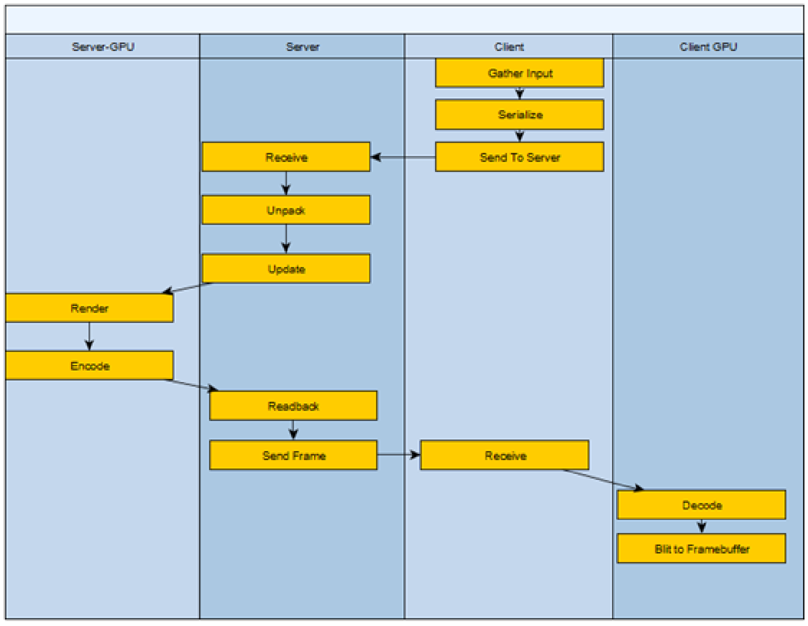


Figure 39 - Remote Render information processing Low Level

The application logic of ARtwin integration is hosted on cloud/Edge servers. When a user starts the application on an XR device, it starts the requested server-side application (Figure 38 and Figure 39). The application is executed within a virtual machine (VM) instance or a Docker container on a physical cloud server through several automated mechanisms. For this purpose, an orchestrator selects an appropriate server and starts a new VM instance or Docker container with the server application. Client application and server application use a signaling server to exchange ICE candidates in order to establish a direct Peer-to-Peer connection between client and server application.

Web RTC

Web RTC is a library for Web-based Real Time Communication (Video, Voice, Data) and builds the base technology for streaming of holographic content over Holo-Light's Remoting Library.

ICE (Interactive Connectivity Establishment) Protocol for Connection Establishment and as standard method of NAT traversal. SDP is used by Web RTC to negotiate the session’s parameters RTP (Real-time Transport Protocol).

3.2.2 Telecom Platform

The Telecom platform is implemented using 5G mobile connectivity platform. This platform itself is composed of two platforms: Radio Access Network (RAN) and Core Network. The two platforms rely on cloud native container based mobile service components deployed on two different flavors of Telco Grade Kubernetes PaaS: A Core Kubernetes PaaS and an Edge Kubernetes PaaS, tailored respectively for the Core and RAN service components. The Core Kubernetes is an almost standard Kubernetes PaaS except for the fact that it includes a Telco Grade extension to support the Stream Control Transmission Protocol (SCTP) protocol that is required to interconnect with the RAN.

From its side, the Edge Kubernetes PaaS includes acceleration capabilities that enables container deployment on bare-metal and accelerated hardware (FPGA) and that, in addition to SCTP support, embeds a first Telco Grade performance extension. Other Telco Grade extensions, such as Data Plane Development Kit (DPDK), Single Root Input/Output Virtualization (SR-IOV) and CPU pinning, can be added if needed.

Figure 17 presented earlier shows how a Kubernetes Platform looks like when it is composed with RFBs.

3.2.3 BIM & PLM platforms and external IOT platforms

- App interface/service to Siemens Teamcenter in order to extract and display meta data from PLM System: (informative) At Siemens factory sites, the CAD models are produced from existing Siemens PLM tools such as PlantSimulator™, ProcessSimulate™, Tecnomatix™ etc. The 3D scans are usually POI (points of interests) or Autodesk Recap formats. There will be sandbox environments set up for the purpose of the ARtwin project which serves as a collection of 3D artefacts for the "test area" for the project. For the measurement and calibration use case for the factory site test area in Karlsruhe, information from all the field devices are collated into an IOT system called SIEMENS SITRANS SAM IQ. This data can be for example measurement data, temperature of electronics, versions of hardware/software used, maintenance, failure and so on. The data assists the field technicians to perform computational or analysis tasks accordingly. In the context of ARtwin, we can have this app data connect to our AR application to help the field operators/technicians perform measurement and calibration tasks to begin with. The app itself is running on Siemens Mindsphere (cloud based, Open IoT OS) for which an account and login will be required.
- App interface/service to Autodesk Fusion in order to extract and display meta data from BIM System.

3.2.4 ARtwin platform

The ARtwin platform is a *Platform of Platforms*, for it is composed of the different platforms listed earlier (AR remote rendering platform, ARCloud platform and telecom platform). It encapsulates those sub-platforms to provide an end-to-end view on the capabilities it has and the services it offers. This PoP will have the characteristics of any platform in terms of views, blueprints, and so on. i.e. It will be editable, deployable, and orchestrated. If the use-case doesn't require full capabilities of ARtwin platform, one can tailor the blueprint, and therefore the deployed platform, to their need. Several versions of this platform can exist.

3.3 Execution Environment Inventory

3.3.1 AR/VR devices

Android Smartphones/Tablets with ARCore

ARCore is a software development kit developed by Google that allows for augmented reality applications to be built. ARCore uses three key technologies to integrate virtual content with the real world as seen through your phone's camera. Six degrees of freedom allows the phone to understand and track its position relative to the world. Environmental understanding allows the phone to detect the size and location of flat horizontal surfaces like the ground or a coffee table. Light estimation allows the phone to estimate the environment's current lighting conditions. ARCore has been integrated into a multitude of phone manufacturers. (reference of Wikipedia article)

iOS Smartphones/Tablets with ARKit

In June 2017 Apple released the ARKit API tool for developers working on virtual reality and augmented reality applications. The ARKit Tool is designed to accurately map the surrounding using SLAM (Simultaneous Localization and Mapping). Moreover, to create augmented reality experiences users don’t need any external equipment. (reference of Wikipedia article)

Universal Windows Platform (UWP) – HoloLens 1 | HoloLens 2

Microsoft HoloLens, is a pair of smartglasses developed and manufactured by Microsoft. HoloLens was the first head-mounted display running the Windows Mixed Reality platform under the Windows 10 computer operating system. It is a full 6 Degree of Freedom SLAM capable mobile device. The tracking technology used in HoloLens can trace its lineage to Kinect, an add-on for Microsoft's Xbox gaming console that was introduced in 2010. It uses Universal Windows Platform (UWP) is an Computing platform. UWP is an extension of the Windows Runtime. Universal Windows apps that are created using the UWP no longer indicate having been written for a specific OS in their manifest build; instead, they target one or more device families, such as a PC, smartphone, tablet, or Xbox One, using Universal Windows Platform Bridges. These extensions allow the app to automatically utilize the capabilities that are available to the particular device it is currently running on. (reference of Wikipedia article)

OpenVR Devices (Oculus, HTC, Varjo,…)

OpenVR is a software development kit (SDK) and application programming interface developed by Valve for supporting the SteamVR (HTC Vive) and other virtual reality headset (VR) devices. The SteamVR platform uses it as the default application programming interface (API) and runtime. It serves as the interface between the VR hardware and software and is implemented by SteamVR. Although OpenVR is the default SDK for HTC Vive, it was developed to have multiple vendor support. (reference of Wikipedia article)

Referring to D2.1, the preference of devices to realize different use cases is shown below:

Table 2 - AR/VR Device Preference

Use case	Primary device(s)	Alternative device(s)
Workstation Planning use cases in GWE, Erlangen Planning new assembly line/production lines in factory site in Fürth	Tablets (iOS and Android)	HoloLens (1 & 2) Some parts of visualization/planning could also be in VR(HTC Vive)
Measurement and calibration use case in SMA, Karlsruhe	HoloLens (1 & 2)	Tablets (iOS and Android)
Walkthrough plant inspections in SMA Karlsruhe	HoloLens (1 & 2)	Tablets (iOS and Android)

In addition, the AR devices shall provide an access to the frames captured by the built-in vision sensors, with a timestamp corresponding to the capture time (accurate to the millisecond).

The AR device should provide an access to the data captured by the inertial measurement unit with a timestamp corresponding to the capture time. Also, the AR devices should provide an access to the calibration parameters for each vision sensor (intrinsic, extrinsic and distortion parameters).

The AR device or its built-in AR library should provide an access to the 3D sparse point cloud with associated descriptors built by the SLAM implementation, and ideally, a way to compute descriptors used by the AR SDK for a set of key points extracted within an image.

Development Pipeline for End Users

If an end user wants to create a new service which covers their use-case, they start in Unity 3D. Unity is a 3D environment developing platform. Here a user can create his own App/Service logic. Considering AR capable AR devices, a user has the benefits of spatial computing, spatial understanding, and some way for input (e.g. HoloLens has Hand Gestures). Based on these components the app behaves in a certain conditional way. Depending on the requirements of the use-case the user can also use already existing AR Services from ARTwin like Remote Rendering if large 3D Data needs to be processed, Object Recognition if real-world object identification is needed or localization.

3.3.2 EDGE Cloud

Edge computing [JSD18] refers to locating applications relatively close to end users and/or IoT endpoints. This greatly benefits applications performance and associated QoE, and it can also improve efficiency and thus the economics depending on the nature of the specific application. Distributed edge computing is analogous to, and can be regarded as an extension of, the evolution of content distribution (CDN) over the last few decades.

Enabling applications to be localized in edge compute close to end users first and foremost improves network transit latency. Latency is a significant driver in improved performance as is high reliability, e.g. through setting up radio bearers that allow low block error rate tolerance. Edge compute combined with the optimized latency performance of 5G air interface and 5G Core processing can reduce round-trip-time by up to two orders of magnitude in situations where there is tight control over all parts of the communication chain. This will enable new classes of cloud applications, in such areas as industrial robotic/drone automation, V2X, and AR/VR infotainment, and associated innovative business models.

One of the functional requirements specified in Deliverable D2.1 [D2.1] is to leverage a distributed computation based on Edge and Cloud computing and optimize workload over nodes available in factory and construction site’s internal network. This can help to re/place dynamically the service (Relocalization, mapping, rendering) on a distributed cloud (device, local case, central cloud) within the second including function and data.

Many use-cases promised by 5G require ultra-high connectivity with low latency and the need for a scalable infrastructure. For that, the solution is to bring performance optimized, virtualized compute-and-store capabilities very close to users. This development will enable even latency-sensitive and high-throughput applications to be efficiently hosted in the cloud.

The Edge Cloud can contain:

- Compute services consisting of baremetal servers or/and hypervisors, virtual machines, or/and container engines.
- Networking services consisting of physical and/or virtual network switches and routers.
- Storage services.

Additionally, the EDGE Cloud may be equipped with powerful processing (ARM, x86), acceleration capabilities (FPGA, SoC), large memory capacity, etc.

This is aligned with the trend observed in the market where the big cloud players (Amazon, Microsoft) offers a local deployment of their cloud. Amazon opened 53 distributed data centers closer to end users, of which 20 opened in the last three years alone. And recently, they offer the possibility of a local deployment, on premises, like the snowball EDGE solution proposed by Amazon.

The EDGE Cloud will host many workloads type like Cloud RAN, Mobile EDGE Computing, analytics, 3D rendering, etc.

The different EDGE Clouds are connected to a Central cloud. The rest of the workload without a latency constraint like the application servers will be deployed in the central cloud. It is a typical IT datacenter, having much more memory storage capacity. According to D2.1, for the Remote rendering framework, a Wi-Fi connection with a large bandwidth range is desired for both uploads and downloads. Around 5MB with HD stereoscopic is desired for bandwidth. The latency shall be 10ms roundtrip time, in the event of without rendering. Furthermore, every device needs its own cloud instance. As per the resources for one instance, a high-end GPU is requested (for example NVIDIA GEFORCE, TESLA DRID GPU, …). The range from the Wi-Fi router shall be within 20-50m with no obstacles in between. For the Relocalization frequency, 30-60 frames per second is desired for HMDs.

For the Remote rendering framework, every edge node needs at least one Virtual Machine with Windows 10 Professional OS including Nvidia Grid.

3.3.3 Central cloud

Central Cloud can be either public or private. A centralized private cloud will comprise dedicated resources (compute, storage, etc) for the organization consuming those resources. The organization may build its own data center infrastructure for this purpose or consume cloud compute services from a service provider. An example of a service provider’s cloud compute “Infrastructure-as-a-Service” (IaaS) hosting capability is *Nokia Innovation Platform* (NIP) which is a live development and trial environment for start-ups, industries and other partners to accelerate innovation of IoT solutions through an open, collaborative model. NIP provides wide range of services that can meet any of your demand:

- *Openshift*: a complete container application platform to quickly develop, host, and scale applications in a cloud environment.
- *Node-RED*: an open source visual tool for wiring together hardware devices, APIs and online services in new and interesting ways.

- *GPU Computing*: If you are scientists, engineers or a lab research and you require high processing capability and access to massively parallel computational power. Nokia now offers virtual machines with GPUs NVIDIA® Tesla® P100. GPU-based instances provide access to VM with NVIDIA GPUs with thousands of compute cores. Tesla P100 is the world's most powerful data center GPU ever built, engineered to deliver giant leaps in performance for deep learning, video processing and other graphics workloads.
- *World-Wide Streams*: World-Wide Streams is a distributed stream processing platform that offers easy onboarding of streams and devices, a rich transformation language to process streams and "serverless" deployment across central and edge clouds. The platform unifies the creation, deployment and execution of data and media processing pipelines.
- *ENGIDI New Generation Devices*: a small electronic device, fixed on the inside of the protective helmet, allows to monitor the worker in real time.
- *Mobagel Decanter™ AI*: MoBagel is a cloud service that uses its Decanter™ Predictive AI engine to help IoT owners easily convert IoT data into business value and insights.
- *Craft AI*: an artificial Intelligence, as a service. Hosted API that enables your services to learn every day: provide a personalized experience to each user and automate complex tasks.
- *Apizee*: Apizee SaaS platform enables easy roll-out of real-time customer engagement, visual assistance, enterprise collaboration as well as video conferencing services.
- *OpenDataSoft* Smart Cities Solution: OpenDataSoft enables corporations to aggregate data from IoT sources, transform it to relevant format enabling exploration and share it accordingly to foster new services.
- *Cloud Compute*: Compute Cloud Instance is a virtual computing environment with a variety of operating systems and a firewall to control the access permissions.

In the scope of ARtwin project, we will use and consider NIP as an IaaS provider. Thus, the service that we are interested in is *Cloud Compute*. In this service, NIP will provide VMs which can be specified and accessed by partners using VPN. On this resource, we can build our PaaS for each use case.

The central cloud can also be public. Public cloud is based on shared physical hardware which is owned and operated by a third-party provider. The primary benefits of the public cloud include, almost infinite scaling, elasticity, the speed of deployment, and the ability to pay only of the server resources you use.

Some major public cloud services include:

- **Amazon Web Service (AWS):** One such products "Elastic Compute Cloud (EC2)", is described as follows:
    - Amazon Elastic Compute Cloud (Amazon EC2) is flexible service that provides resizable cloud-based compute capacity in the form of EC2 instances, which are equivalent to virtual servers. You can commission one or thousands of instances simultaneously, and pay only for what you use, making web-scale cloud computing easy. We provide a broad range of instance types optimized to fit different use cases, and choice in how you pay for instances, so you can optimize your application based on business needs while minimizing costs.
- **Google Cloud:** One such product "Compute Engine", is described as follows:

    - Google Compute Engine delivers virtual machines running in Google's innovative data centers and worldwide fiber network. Compute Engine's tooling and workflow support enable scaling from single instances to global, load-balanced cloud computing.
- **Microsoft Azure:** One such product "Virtual Machines", is described as follows:
    - With support for Linux, Windows Server, SQL Server, Oracle, IBM and SAP, Azure Virtual Machines gives you the flexibility of virtualization for a wide range of computing solutions – development and testing, running applications and extending your data center.

The major public cloud providers all have a very diverse and rich range of products, beyond the relatively simple "IaaS" compute offerings listed above.

3.4 Deployment inventory

3.4.1 All-in-one 5G AR edge case

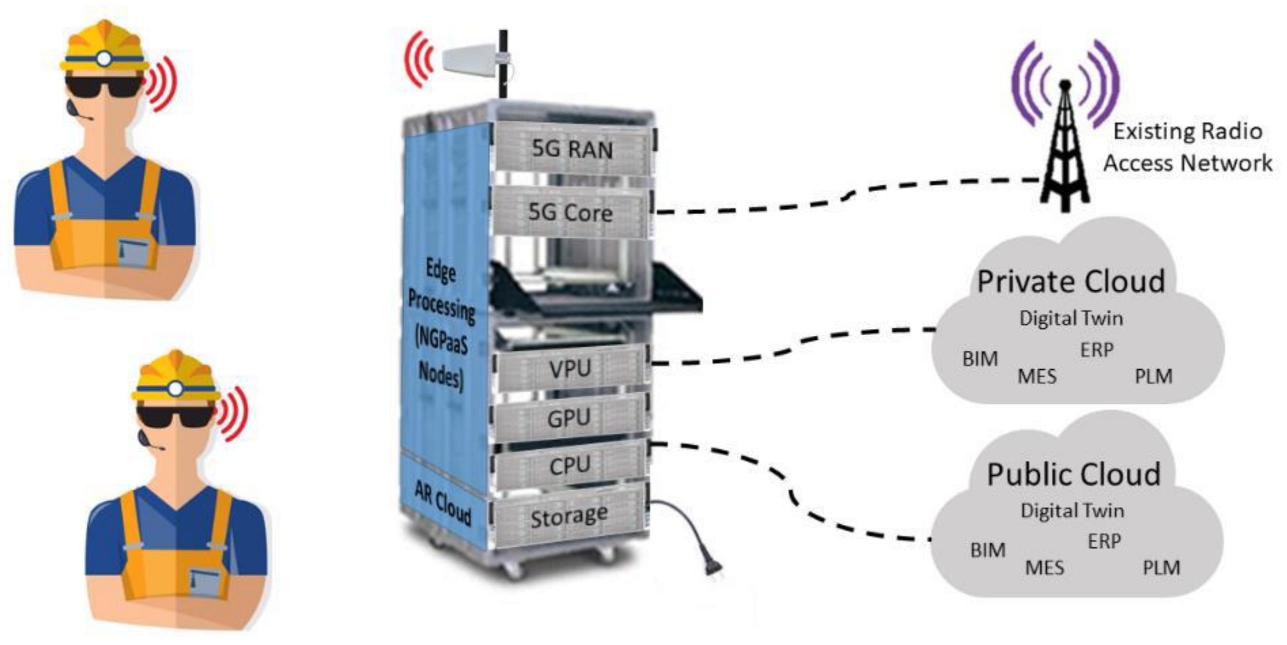


Figure 40: All-in-one 5G AR edge case

The All-in-one 5G AR edge case is a stand-alone solution offering a complete AR cloud solution with 5G connectivity. The ARtwin platform of platforms consisting of the remote rendering, the ARCloud and the telecom platforms can be fully deployed on this autonomous solution. It will consist of the following elements:

- A 5G multi-user mimo base station,
- A Telecom platform including 5G RAN (Nokia) and 5G Core (b<>com WEF),
- An ARCloud platform,
- A Remote Rendering platform,
- A server of CPU,
- A server of GPU,
- Storage capacity.

Two instances of the All-in-one 5G AR edge case will be implemented, one travelling between the Siemens factories and the selected construction site to evaluate the pilot use cases, and the other one used in laboratory for the integration and testing of the services.

This All-in-one 5G AR edge case, thanks to its autonomy, will not require any connection to a private production cloud, avoiding any disruption on the test site. However, a connection to a remote cloud, public or private, could be considered.

3.4.1 Lab + Remote Compute

This deployment anticipates low resource edge case that is not able to host the all-in-one scenario. In this case, some platforms, or parts of platforms that are not latency-sensitive can be offloaded to a remote cloud and connected through the telecom platform to the components running on the edge case.

4 Conclusions

TM Forum’s *Digital Platform* approach offer great advantages for building, editing, and operating complex platforms such as ARtwin, thanks to multi-sides, multi-views, RBAC, composition, recursion and other features. After the introduction to TM Forum’s Digital Platform approach, this document provides a description of the Platform of Platforms implementation of this concept and a preliminary assessment of its relevance to ARtwin. Next, a detailed inventory of the planned services, platforms, execution environments, and deployments is presented. This document’s preliminary specifications and architecture will be used as a starting point for the platforms development taking place in WP3.

5 References

[IG1157] TMForum 2020, IG1157 Digital Platform Reference Architecture Concepts and Principles R19.0.1. Online, accessed on April 21st, 2020.

[PAC16] Parker, Geoffrey G., Marshall W. Van Alstyne, and Sangeet Paul Choudary. Platform Revolution: How Networked Markets Are Transforming the Economy? and How to Make Them Work for You. WW Norton & Company, 2016.

[Mit17] Zach Church 2017. Platform Strategy Explained. Online, accessed on April 21st,2020. <https://mitsloan.mit.edu/ideas-made-to-matter/platform-strategy-explained>

[IG1167] TMForum 2020, IG1167 ODA Functional Architecture v5.0.1. Online, accessed on April 21st, 2020.

[LF14] James Lewis, Martin Flower 2014. Microservices. Online, accessed on April 21st, 2020. <https://martinfowler.com/articles/microservices.html>

[ETSI20] ETSI 2020, Augmented Reality Framework (ARF); AR framework architecture. Online, accessed on April 21st, 2020.
<https://www.etsi.org/deliver/etsi_gs/ARF/001_099/003/01.01.01_60/gs_ARF003v010101p.pdf>

[JSD18] FCC 5G IoT Working Group 2018. 5G Edge Computing Whitepaper. Online, access on April 21st, 2020. <https://transition.fcc.gov/bureaus/oet/tac/tacdocs/reports/2018/5G-Edge-Computing-Whitepaper-v6-Final.pdf>

[D2.1] ARtwin Project 2020. ARtwin Deliverable D2.1 Artwin Requirements Specification.

6 Appendix A: Glossary

3D Tracking is the process of estimating the pose (position and orientation) at time t of a real-world object or of an AR device knowing its pose at time t-1.

AR (Augmented Reality). MR (mixed reality) is the ability to mix in real-time spatially registered digital content with the real world.

ARCloud (Augmented Reality Cloud) aims to compute and store in the cloud a complete cartography of the real-world used both for visualization and for accurate localization.

BIM (Building Information Modeling) is a digital representation of physical and functional characteristics of a facility. A BIM is a shared knowledge resource for information about a facility forming a reliable basis for decisions during its life cycle; defined as existing from earliest conception to demolition[2].

Digital Twin is a virtual representation of a physical product or process, used to understand and predict the physical counterpart's performance characteristics.

Mapping is the process of building a 3D map of the real environment generally used for localizing one or a set of vision sensors.

Relocalization is the process of estimating the pose of a device at time t based on a knowledge of the real environment, but without any knowledge of its pose at time t-1.

AGVs: Automated Guidance Vehicles – is a portable robot that follows along marked long lines or wires on the floor, or uses radio waves, vision cameras, magnets, or lasers for navigation.

PoP: Platform of Platforms

RFB: Reusable Functional Block

RDCL: RFB Description and Composition Language

IaaS: Infrastructure as a Service

PaaS: Platform as a Service

SaaS: Software as a Service

IoT: Internet of Things

OSS/BSS: Operations Support System / Business Support System

NaaS: Network as a Service

RFS: Resource Facing Services

CFS: Customer Facing Services

CSP: Communication Service Provider

CI/CD: Continuous Integration / Continuous Delivery

RBAC: Role Based Access Control

API: Application Programming Interfacing

SDK: Software Development Kit

FPGA: Field Programmable Gate Array