PROJECT "LOCUS": LOCalization and analytics on-demand embedded in the 5G ecosystem, for Ubiquitous vertical applicationS

Grant Agreement Number: 871249
(https://www.locus-project.eu/)

DELIVERABLE D2.5

"System Architecture: final version"

Deliverable Type:	R
Dissemination Level:	Public
Contractual Date of Delivery to the EU:	31/07/2021
Actual Date of Delivery to the EU:	17/11/2021*
WP contributing to the Deliverable:	WP2 – Use cases, Requirements, Security, System Architecture
Editor(s):	Incelligent, Kostas Tsagkaris
Author(s):	CNIT, Stefania Bartoletti, Andrea Conti, Domenico Garlisi, Danilo Orlando IBM, Joseph Antony, Gabriele Ranco Incelligent, Aristotelis Margaris, Athina Ropodi NEC, Giuseppe Siracusano

	NXW, Giacomo Bernini, Elian Kraja, Mario Salinas Orange, Sana Ben Jemaa SAMS, Mythri Hunukumbure, Tomasz Mach TEI, Stefano Stracca, Marzio Puleri VIAVI, Takai Eddine Kennouche, Chris Murphy, Howard Thomas UMA, Sergio Fortes, Eduardo Baena, Antonio Tarrías, Raquel Barco VIAVI, Takai Eddine Kennouche, Chris Murphy, Howard Thomas
Internal Reviewer(s):	CNIT, Stefania Bartoletti IBM, Faisal Ghaffar NEC, Gurkan Solmaz
Short Abstract:	The goal of this deliverable is to detail the reference architecture of LOCUS, compliant with the system requirements and use-cases defined in Task 2.1 and based on the work done in Task 2.3. This final version describes the main functional and system blocks, related to localization analytics and other platform functions, as well as provides an overall view of the LOCUS Platform and its deployment options.
Keyword List:	LOCUS Reference Architecture, Localization, analytics, LOCUS System View, Deployment aspects.

** This deliverable has been intentionally delayed with respect to the due date, to consider feedbacks coming from the work of other WPs (especially WP6) and from the project review, so as to improve and refine the architecture correspondingly.*

Executive Summary

LOCUS aims at providing a unified and generalized localization analytics platform that will offer location-based analytics relying on built-in data analysis, correlation and machine learning capabilities. This deliverable elaborates on the reference architecture of LOCUS based on the LOCUS use cases defined in Deliverable 2.1 and their high-level requirements, as well as system-level, and security and privacy requirements. This deliverable presents the final architecture, which has been developed as an advancement of the preliminary version defined in Deliverable 2.4. This final document depicts the entire work done in Task 2.3 on the definition of the major functional blocks and the various functions of the LOCUS Platform, indicative applications related to the suggested use cases, as well as services to be exposed at the application layer -external to the LOCUS Platform. It is also the basis and in line with the implementation work in Work Packages (WPs) 4 and 5, as well as with the planned demonstration activities in WP6. In fact, the LOCUS Platform is designed to serve multiple layers of analytics services in an environment that is both connected to the 3GPP ecosystem, but also to other external entities, and can support production-ready deployment of Artificial Intelligence models. A comprehensive view of the LOCUS system architecture is presented, along with the deployment aspects through the LOCUS management and orchestration functions on top of a virtualized infrastructure spanning across edge and core locations.

This deliverable has been intentionally delayed with respect to the due date, to consider feedbacks coming from the work of other WPs (especially WP6) and from the project review, so as to improve and refine the architecture correspondingly.

VERSION CONTROL TABLE			
VERSION N.	**PURPOSE/CHANGES**	**AUTHOR (S)**	**DATE**
0.2	Agreed ToC	INCE	24/09/2021
0.3	Consolidation with D2.4 Functions	INCE	30/09/2021
0.3.1	Contributions (WP6 Section)	VIAVI	04/10/2021
0.4	Contributions (Sections 2,3,4)	NXW	07/10/2021
0.7	Sections 2 ,3 ,4 Finalized content	INCE	11/10/2021
0.8	LOCUS SDK and Standardization	NXW	13/10/2021
0.9	Standardization Contributions / Corrections	NXW, INCE, VIAVI	18/10/2021
1.0.0_for_review	Version for internal review	INCE	28/10/2021
1.0.0_after_review	Final Version	INCE	15/11/2021
1.1	Final Revision by Coordinator	Nicola Blefari Melazzi	17/11/2021

Table of Contents

List of Abbreviations

Abbreviation	Full Name
3GPP	3rd Generation Partnership Project
5G	Fifth generation technology standard for cellular networks
ADRF	Analytics Data Repository Function
AEF	API Exposing Function
AI	Artificial Intelligence
AMF	Access and Mobility Function
AnLF	Analytics Logical Function
AoA	Angle of arrival
API	Application Program Interface
ARIMA	AutoRegressive Integrated Moving Average
ASN.1	Abstract Syntax Notation
BSS	Business Support System
CAPIF	Common API Framework for 3GPP northbound APIs
CCF	CAPIF Core Function
CID	Cell ID
CDC	Change Data Capture
CNF	Containerized Network Function
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
DAG	Directed Acyclic Graph
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DCCF	Data Coordination and Collection Function
DL	Deep Learning
DoA	Direction of arrival
e-CID	Enhanced Cell ID
eLCS	enhanced LoCalization Services
eNA	enablers for Network Automation
GMLC	Gateway Mobile Location Centre
GNSS	Global Navigation Satellite System

GPS	Global Positioning System
HCS	Highly Connected Subgraphs
HDD	Hard Disk Drive
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
KML	Keyhole Markup Language
KPI	Key Performance Indicator
LCM	Lifecycle Manager
LDAF	Localization Data Analytics Function
LDR	Location Deferred Request
LEN	Localization Enabler
LMF	5G Core Location Management Functions
LPI	Local Polynomial Interpolation
LSTM	Long short-term memory
LTE	Long-Term Evolution
MANO	Management and Orchestration
MBFSN	Multicast-broadcast single-frequency network
MDAF	Management Data Analytics Function
MDAS	Management Data Analytics Services
MDT	Minimization of Drive Tests
MFAF	Messaging Framework Adaptor Function
ML	Machine Learning
MO-LR	Mobile Originated Location Request
MT-LR	Mobile Terminated Location Request
MTLF	Model Training Logical Function
NAS	Non-access Stratum
NWDAF	Network Data Analytic Function
NEF	Network Exposure Function
NFV	Network Function Virtualization
NG-RAN	Next Generation Radio Access Network
NI-LR	Network Induced Location Request

NN	Nearest Neighbours
NSD	Network Service Descriptor
NSE	New Services
OID	Object Identifier
OSM	Open-Source MANO
OSS	Operations Support System
PaaS	Platform as a Service
PCA	Principal Component Analysis
PLMN	Public Land Mobile Network
PoC	Proof of Concept
POI	Point of Interest
PRS	Positioning Reference Signal
QoS	Quality of Service
RBF	Radial Basis Function
REST	REpresentational State Transfer
RSRP	Reference Signal Received Power
RSRQ	Reference Signal Received Quality
RSS	Received Signal Strength
RSSI	Received Signal Strength Indicator
SaaS	Software as a Service
SBI	Service Based Interface
SDK	Software Development Kit
SDN	Software Defined Networking
SEAL	Service Enabler Architecture Layer for Verticals
SFTP	Secure File Transfer Protocol
SID	TM Forum Information Framework
SINR	Signal to interference and noise ratio
SNM	Smart Network Management
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SoTA	State of The Art
SRS	Sounding Reference Signal

SSB	Single-Sideband modulation
SSH	Secure Shell
SSO	Single-Sign On
SVM	Support Vector Machine
SQL	Structured Query Language
TDoA	Time difference of arrival
TLS	Transport layer Security
ToA	Time of arrival
TPS	thin plate spline
t-SNE	t-Distributed Stochastic Neighbour Embedding
TTC	Time To Collision
UC	Use Case
UDM	Unified Data Management
UDP	User Datagram Protocol
UE	User Equipment
UMAP	Uniform Manifold Approximation And Projection
VAL	Vertical Application Layer
VM	Virtual Machine
VNF	Virtual Network Function
VRU	Vulnerable road user
WP	Work package
XML	(e)Xtensible Markup Language
YAML	YAML Ain't Markup Language
ZSM	Zero-touch network and Service Management

Table 0.1 Abbreviation List

Table Index

1 Introduction

1.1 Introduction and Audience

Deliverable 2.5 (D2.5) “System Architecture: final version” presents the final definition of the LOCUS Platform architecture. This deliverable is based on the work done in Task 2.3 “System Architecture Specification and Implementation”. As this is a public deliverable, it is intended for the LOCUS consortium members, the EU commission, as well as general the public.

Its main objective is to elaborate on the reference architecture of LOCUS taking into account (a) the LOCUS use cases (UCs) as defined in Deliverable 2.1 [1] and their high-level requirements, so as to be supported by the LOCUS Platform derived from them, (b) the security and privacy requirements that are related to them and are part of the work done in Task 2.2, and, (c) system-level requirements, given the platform’s expected ability to provide advanced localization analytics in order to enable both Smart Network Management applications, as well as vertical sector applications through exposing services to 3rd parties.

The deliverable presents an evolved version of the architecture as compared to the submitted Deliverable 2.4 [2]- LOCUS Architecture. Specifically, the deliverable introduces the major functional blocks and then goes into more detail, defining the various functions of the LOCUS Platform and describing their use, possible processing and analytics, expected inputs and outputs. These functions refer to data collection, data management, analytics services and functions, Artificial Intelligence (AI) -including machine learning (ML) and deep learning (DL)- functions, security and privacy functions, as well as management and orchestration functions. These functional aspects of the LOCUS Platform are then translated into system blocks, interfaces, and specific technologies, while a Software Development Kit, i.e. platform tools for implementation purposes, is laid out. Furthermore, a first attempt is made to map the LOCUS Platform system blocks to the demonstration activities taking place in Work Package 6. Lastly, the overall context and expected direction for the future standardization work with respect to the LOCUS Architecture is described.

1.2 Relation with D2.4

A preliminary high-level architecture for the LOCUS Platform was presented and described in D2.4 [2]. At that stage of the project, all the functional requirements for the LOCUS Platform analytics had been collected and the project was focusing on diverse, complex and experimenting with multiple approaches in the implementation of the various functions, i.e. Localization enablers (LEN), New 3rd party Services (NSE) and Smart Network Management (SNM). To support these activities and to provide more liberty for the research and development processes, D2.4 was drafted so as to provide many capabilities and options for

the implementers, allowing them to focus on the actual results of their services. In this second phase of the project (after the 1st review), the majority of the use case implementations have matured and finalized their system and platform requirements. This has allowed the consortium partners to be more specific with respect to the actual platform details and features, narrowing them down and going into further detail at the level of function, block and specific selected technologies for the various system blocks. In addition, WP6 activities have greatly influenced the specifications of this document (D2.5), as architectural, interfacing and system aspects, as well as the actual operation of the platform were taken into account. All these inputs have resulted into a finalized document that will be able to support all the current and future use case activities as functional blocks of the platform, while simultaneously working as a deployment guideline for a realization of the LOCUS Platform on cloud or on-premises 3GPP environments.

1.3 Notes on the Document Delivery Date

The LOCUS D2.5 is a document that encloses the summary of requirements, system blocks and functionalities of all LOCUS activities (from concept, use cases to actual implementations) including the demonstration activities related to WP6. An initial draft of this document was prepared for the review of July 2021 and parts of the architecture were presented, receiving mostly positive feedback. More specifically, the reviewers pointed out the importance of having clear mappings of the LOCUS Platform system blocks to existing 3GPP standards and the possibility of issues arising regarding the latency-sensitive use cases. They further inquired whether the technology selection satisfies these needs as well as the requirements of infrastructure technologies that will allow for the deployment of part(s) of the LOCUS Platform in the “Edge”’ and “Core” domains of any assumed provider infrastructure. Therefore, and after considering a) the newly planned way forward of the WP6 activities (platform implementation) and b) the feedback received from the reviewers on all work packages, it was concluded that some parts of the LOCUS Platform architecture needed to be revised. For this reason, we requested an extension on the delivery of this document (from September to November 2021) to provide a finalized document that will capture all the changes required and the future advancements that will be added by the LOCUS partners until the final review.

1.4 Document Structure

This Deliverable is structured in 7 sections. After the Introduction (Section 1), the following sections are included:

- Section 2 describes the high-level functional architecture of LOCUS Platform;
- Section 3 maps the functional architecture into system blocks, interfaces, and technologies for the LOCUS Platform;

- Section 4 is dedicated to the LOCUS SDK (Software Development Kit), platform tools that are used for the implementation of Analytics Services;
- Section 5 correlates the finalized system architecture with the WP6 demonstration activities;
- Section 6 is comparing the existing 3GPP specification with the LOCUS platform system blocks aiming at positioning itself in a collaborative ecosystem.

The last section (Section 7) summarizes the conclusions of this deliverable.

2 LOCUS Platform Functional Analysis

The overview of the LOCUS Analytics Functions contains all the high-level categories of LOCUS functions grouped together by relevance and by an assumed co-existence in system blocks.

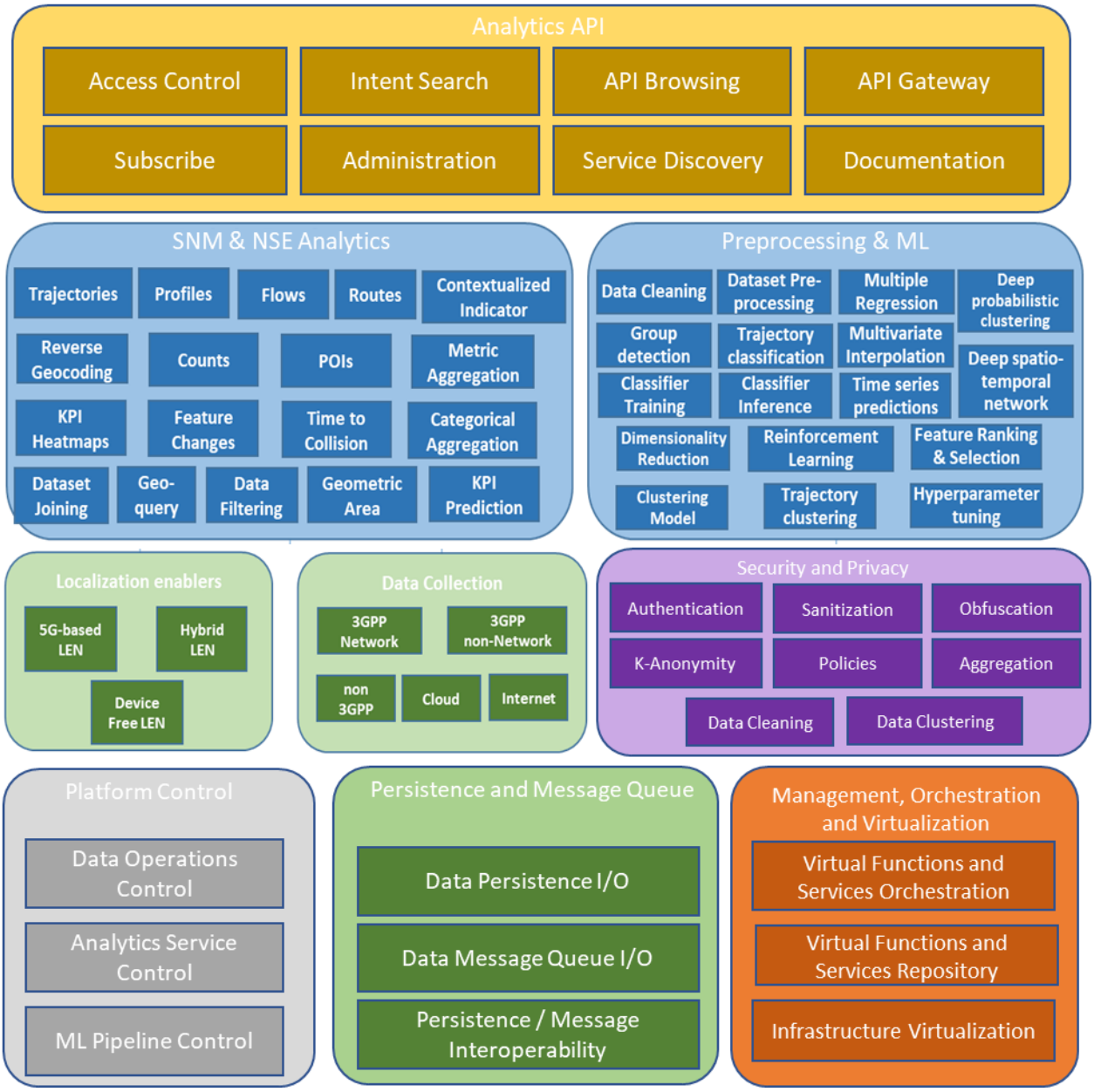

Figure 2.1 LOCUS Platform Function Diagram

A brief description of each of the high-level functional blocks is provided in the table below in the order that they will be presented in subsections 2.1-2.8.

Table 2.1 LOCUS Functional Blocks

Functional Block	Description
Data Collection	This functional block includes the summary of operations that are required for the acquisition of data from external or 3GPP ecosystem sources and deliver it inside the LOCUS platform. The different instantiations of LOCUS analytics functions, require different inputs to produce their desired output results. During the onboarding phase of an analytics function, it is important that this function is accompanied with a parametrization for the data collection function, in order to provide all the necessary information for the acquisition of such data and for the delivery to the relevant persistence modules inside the platform.
Persistence and Message Queue	The *LOCUS Persistence and Message Queue,* i.e. the LOCUS data store, is responsible for the storage of all relevant data, including data collected from sources outside of LOCUS (e.g. 3GPP-related measurements or 3rd party applications on top of 3GPP networks). It will also provide persistence capabilities for all the internal LOCUS analytics functions which will, in various stages, require to store information in a scalable and efficient centralized data lake. It enables both batch ingestion as well as data streaming functionalities for data originating outside the LOCUS platform or the various LOCUS functional blocks. All of these data are exposed within the LOCUS system through the *LOCUS APIs* (i.e. Application Program Interfaces).
Platform Control	The *LOCUS Platform Control* functions are the mechanisms, i.e. the sum of all the required tasks, to trigger the activation/ de-activation/ re-parametrization of an analytics function (including storing all the related information in a service image catalogue). The LOCUS Platform is also strongly related to applied ML/ AI; therefore it needs to support natively Machine Learning, Optimization and Pipeline Functions. The developers of the analytics functions for all use cases will be using these functions via their programmatic or other (e.g. network/ http) APIs in order to perform functions such as hyperparameter tuning, high performance native binaries binding, model persistence/ model Key Performance Indicator (KPI) persistence. Finally, the execution of the analytics functions computational backend tasks (including training/ predict/ persist lifecycle that were mentioned before) will need to be controlled by a data operations' controller set of functions, that map the demands that originate from the higher-level APIs or are triggered by events from the data collection functions.

Localization Enablers (LEN)	This functional block is responsible for providing the location data of the user device (e.g. coordinates, velocity, direction) needed by the other LOCUS blocks. It utilizes data coming from various 5G network and other sources, i.e. network data, user device, external and social data, outside the LOCUS system and provides an estimation of the user device location. This block allows for continuous localization and tracking with varying levels of granularity of the user devices with high positioning accuracy to distinguish between different users, vehicles, and targets within an area. For this reason, it employs analytics, data fusion and ML mechanisms, to achieve high accuracy, ensure data reliability for the detection and mitigation of attacks against location data and correction of the data. The information acquired from the external sources, as well as the expected positioning-related output is stored in the *LOCUS Persistence and Message Queue Block*.
Localization & Analytics for Smart Network Management (SNM)	This functional block is responsible for the use of Analytics and ML mechanisms in conjunction with the position-related information provided by the *Localization Enablers*, to offer localization analytics services related to KPI monitoring and knowledge building, network planning, management, optimization, and diagnostics purposes. The results are exposed outside of LOCUS through the *LOCUS Analytics APIs* and can be used by other external Smart Network Management applications and given availability of the appropriate APIs, can be applied towards Operations Support System (OSS)/ Business Support System (BSS). Intermediate results can be stored internally within the LOCUS Platform and re-fed to the various system blocks, enabling the internal communication of the various services.
Localization & Analytics for New Services (NSE)	Like the previous functional block, the location-related information is utilized to expose through the *LOCUS Analytics APIs*, Analytics and ML-derived results for localization analytics as a service towards vertical and 3rd party applications. Based on the identified use cases in D2.1 [1], these analytics and ML results relate to people mobility, flow monitoring and self-driving objects. Both SNM and NSE functions employ a variety of ML functions, indicatively included in Figure 2.1.
Analytics API (Consumer functions, Subscription functions)	The *LOCUS Analytics APIs* include all those functions that are required for the exposure of the resulting analytics function results coming from the Localization Enablers, Localization & Analytics for Smart Network Management and Localization & Analytics for New Services. This is split into two core categories: (i) API access functions which include function search mechanism (browsing and AI-assisted intent-based search), API Administration panel for subscription and resource management of the

	user space; and (ii) LOCUS Analytics northbound API gateway functions which include functions for the regulation of data access (linked with security functions), while providing traditional access control functionalities for enabling authorization and authentication, as well as aggregation endpoints for the multiple, internal analytics functions (gateway functions) for the exchanged data exposing LOCUS localization analytics data towards new services and smart network management applications.
LOCUS Virtualization Management and Orchestration (MANO)	All the LOCUS analytics functions are subject to automated lifecycle management through dedicated *LOCUS MANO* features. *LOCUS MANO* enables the automated instantiation, configuration, and operation of the various localization and analytics, which are deployed in the form of virtualized network functions (VNFs) on top of the virtualized infrastructure, following the ETSI NFV (Network Function Virtualization) principles for functions virtualization and their management. The LOCUS MANO also enables the packaging of the LOCUS functions as virtualized functions for their share and re-use across different localization analytics services.
Security & Privacy Functions	The location security & privacy functions relate to all the data privacy and security aspects, including authentication and advanced cryptographic techniques for all network interfaces such as homomorphic encryption, secure multiparty computation, and secure conditional sharing techniques, data management policies (e.g. anonymization, obfuscation) for ensuring privacy regulations and trigger alarm functionalities. This is done in parallel with the *LOCUS Analytics APIs* responsible for data delivery and sharing. It is also done during the data collection phase (where the core anonymization operations are performed) These functions presented in Figure 2.1 need to be applied to all data as well as both external and internal interfaces.

2.1 Data Collection Functions

This LOCUS function is responsible for collecting the data from various sources outside the LOCUS Platform to support the LOCUS proposed functionalities, therefore cater to the delivery of the appropriate “Localization Analytics as a Service” by the platform. Each analytics function should describe its data requirements which will be used as input for a new “instantiation” of a data collection function. This will ensure separation between the data-provisioning and the operation of the analytics function itself, allowing for more flexible resource allocation and

better performance to fulfil the data input requirements associated with the LOCUS Platform use case.

2.1.1 Data Collection Function High-level Requirements

A set of high-level requirements with respect to the related data collection functions is derived based on the use cases described in detail in D2.1 [1]. As mentioned previously in D2.4 [2] and described also here in order to provide an overall view of the LOCUS Platform, the general categories of data expected to be employed in the LOCUS system are defined as:

- *Network data*, i.e. positioning-related measurements (e.g. Received Signal Strength-RSS, Time of arrival-ToA, Angle of arrival-AoA, Direction of arrival-DoA, Time difference of arrival-TDoA) acquired through measurement campaigns, *3GPP location data* through 5G Core Location Management Functions (LMFs) defined in the 3GPP TS 29.572 [3], *5G network Key Performance Indicator (KPI) data* (e.g. cell-level or area-wide network KPIs such as traffic coming from network monitoring and management analytics), and *Network configuration data* (i.e. topology, configuration parameter information).
- *User device data*, referring to additional positioning data and information from user devices, such as Global Positioning System (GPS), barometer and others.
- *Environment/External data, such as environment and other external auxiliary to position-related data***,** i.e., data from Internet of Things (IoT) devices (4G, WLAN or others), e.g. cameras, sensors, as well as auxiliary data (indoor maps, building boundaries and so on). *Social data* can be considered a specific subcase of external data. They can be either directly collected from 3rd Party Apps or produced by LOCUS analytics functions based on collected external information.

Another relevant data collection aspect refers to (a) the data rate and (b) if it is acquired through:

- A Polling Process (Poll), i.e., a periodical checking for data to be fetched, or
- *A Publish-Subscribe (Pub-Sub) Mechanism*, appropriate for streaming data; where the subscriber(s) express interest in specific information and only receive messages by a publisher that are relevant to the expressed interest.
- *An Asynchronous Request-Response (Req/Resp)*, where an interaction is initiated by sending a Request message and the system matches the Response message that is sent by the provider at some unknown later time.

Other data collection aspects include the specific data transportation protocols adopted for accessing such data sources outside the platform. In LOCUS, two main protocol “channels” are envisaged to collect data:

- HTTP (Hypertext Transfer Protocol) consuming REST (REpresentational State Transfer) services: in this case, the data collection function makes use of an HTTP REST API to directly query as client the external data source. As an alternative, the data collection function itself could offer an HTTP REST API (server side) to receive the data from the external source (e.g. in the case of pre-existing datasets to be loaded in the LOCUS Platform)
- Message bus services: The message bus approach refers to TCP-based message exchange on top of a message broker. There are two operational modes of this method: (a) When an external message broker is accessible to the LOCUS Platform, it can be used as the source for the data collection; or (b) when the message bus is offered by the LOCUS Platform and the external data source is a producer to that source. In both cases, the LOCUS data collection function acts as a consumer of the continuous stream of data (while the external data source acts as a producer).

2.1.2 Data Collection Functions Analysis

This section introduces the functions that are used by the LOCUS Platform in order to access external information, either to be stored via the various Persistence functions, or to be provided/forwarded to a next processing layer (e.g. a real time service). The data collection functions are defined based on the following aspects; the Data Collection Function Target (i.e. the source of the data), the Data Collection Interaction Mechanism and the Data Type.

Data Collection Function Target: A data collection function will always involve a target entity to which the collection process will be addressed. Different sources implicitly define different transport protocols and parameterization for their interaction. To select a function target for a data collection template we must also specify the parametrization required accordingly. Table 2.2 enumerates the various targets and parameters.

Table 2.2 Data Collection targets & parameters

Function Target	Description	Parameters
Internet Sources	Various http-based (e.g. REST API) or other internet-friendly protocols	URL per endpoint, authentication-specific parameters
Google or IOS specific cloud services	Cloud services integrated with mobile phones	User Access tokens, type of service accessed (e.g. google firebase or google account analytics)

3GPP Network sources	Network-related operational information (network layers for user equipment - UE or RAT devices, system aspects, network meta-data) provided via direct or indirect interaction with the NME	3GPP compliant description of the wanted resources via JSON1 [4], XML2 [5], YAML3 [6], ASN.14 [7], graphQL5 [8] or other language
3GPP non-Network sources	Non-Network data provided via direct or indirect interaction with other 3GPP-specific entities (e.g. LMF, 5G localization entities, Smart City entities)	3GPP compliant description of the wanted resources via JSON, XML, YAML, ASN.1, graphQL or other language
Non-3GPP network sources	Other Sources of network data that can be accessed for accessing non 3GPP related information, e.g. SNMP (Simple Network Management Protocol), OpenFlow enabled routers/cloud switches)	Target Controller IP/Port and connection configuration. Definition of the wanted resource (SNMP Object Identifier-OID or OpenFlow field identifier)

Data Collection Interaction Mechanism: The interaction mechanism refers to the way that the data collection operation will be performed. In general, this can be split into two categories, active (system-initiated data collection) and passive (environment-initiated data collection).

- *Passive (environment-initiated)*
  - LOCUS external REST API; this way new data is pushed towards the LOCUS system via POST/UPDATE http methods or another http implementation.
  - LOCUS message bus; e.g. a message consumer that waits for messages from an external queue.
- *Active (LOCUS-initiated, automated or manual)*
  - Initial/one-time extraction; this is the general case for acquiring a large dataset as a non-repetitive task (e.g. extract an open-source dataset).

1 JavaScript Object Notation

2 (e)Xtensible Markup Language

3 YAML Ain't Markup Language

4 Abstract Syntax Notation

5 Graph Query Language

- Assume new state; i.e. every time a resource is accessed, a new measurement is assumed and acquired. For this case, a necessary input parameter would be the polling interval of the data collection function.
- State Difference Recognition; i.e. access to a resource for which no notion of state is available and perform updates, incremental or not on a local model. Additional input parameter that is required is the implementation of the comparison process (identification of the new state).

- *Combined approach*
  - In some cases of data collection interactions, a hybrid approach for the integration may be necessary. For example, a public REST API could be used to receive notifications for new data being available in a public server (including the file path and other configuration) and then the active mechanism will be activated in order to acquire the dataset and ingest it into the platform.

Data Collection Function Data Type: The data types the data collection function supports are presented below. These categories are then linked to the available persistence storage options, as their respective query engines will assist on the data manipulation process with the best technology available.

- *System entity meta-data*: System entities can vary, including different categories such as network elements, user equipment (UE) devices, customers, cars, i.e. single identifiable entities that are involved into the LOCUS functional architecture. Their meta-data are static, non-temporally correlated information of various data types (e.g. cell name, site name, radio configuration parameters).
- *System entity time-related data*: For the system entities, data that are correlated with time (single timestamp or duration) are denoted as 'entity time-related' data and can vary from different contextual fields (location, network KPIs, sensor input data etc.).
- *Geometry data* (e.g. areas, maps): geometry meta-data, i.e. data representation of spatial structures and definition of spaces (e.g. Keyhole Markup Language - KML, XML [9], Geospatial Data JSON – GeoJSON [10]).
- *MultiMedia data & metadata (context and time)*: Multimedia must be specially handled by the LOCUS Platform, and they usually refer to image, sound, video data or other special cases (e.g., spectrum).

In Table 2.3, the various data, derived from the Functional Requirements of the LOCUS Platform and presented in the deliverable D2.1 [1], are mapped in the Data Targets, Interaction Mechanisms and Data Types.

Table 2.3 Mapping of data types derived from the functional requirement analysis to specific Data Targets, Interaction Mechanisms and Data Types

Data Description	Data Target	Interaction Mechanism	Data Type
Position-related measurements (e.g. RSS, ToA, AoA, DoA, TDoA)	3GPP network Sources	Active	System entity time-related data
User device location information in the 5G network through 5G Core Location Management Functions (LMFs), as defined in the 3GPP TS 29.572 [3]	3GPP non-Network sources	Active	System entity time-related-data
5G network KPI data, including cell-level or area-wide network KPIs (e.g. traffic, flows, etc.) coming from network monitoring and management analytics, if required by interacting with 5G Core network functions	3GPP network Sources	Active	System entity time-related data
Network topology and configuration parameters information related to network devices	3GPP network Sources	Active - One-time	System entity meta-data, geometry data
Additional positioning data and information from user devices, such as GPS, barometer and others	-Internet Sources -Google or IOS specific cloud services	Active	System entity time-related data

Environment data, i.e. system communicate directly (online) with various IoT devices (e.g. cameras, sensors)	-Internet Sources -Non-3GPP network sources (e.g. IoT gateways, camera gateways, media servers)	Passive	System entity time-related-data, Multimedia & metadata
Auxiliary data and specifically access to open data cloud services, and metadata like indoor maps, building boundaries and so on.	-Internet Sources	Active - One-time	System entity meta-data, geometry data, Multimedia & metadata
Social data	-Internet Sources -Google or IOS specific cloud services	Active	System entity meta-data, System entity time-related data
Non-3GPP positioning-related measurements e.g. WiFi	-Non-3GPP network sources (e.g. Cisco CMX)	Active	System entity time-related data

Data Collection Function Template: Based on the data collection requirements and characteristics defined above, the Data Collection Function Template can be defined as follows in Table 2.4.

Table 2.4 Function: Data Collection

Function: Data Collection

Input:
• Data Collection Function Target (one of the available targets enumerated)
  ○ Function target sub-parameters (e.g. for internet REST API specify the URL)
• Data Collection Function Interaction Mechanism (one of the available interaction mechanisms enumerated)
  ○ Function interaction mechanism specific parameters (e.g. for Active select the sampling interval)
• Data Collection Function Data Type (one of the available high-level function data types enumerated)
• Callback for the invocation of other LOCUS Platform function
• Callback behaviour based on the various data collection status signals

• Failure handling
Description: This function's purpose is to collect the various data types from external -to the LOCUS Platform- sources with the selected interaction mechanisms (as described in the previous sections).
Output: • Batches or slices of data from the various data sources defined by the Data Collection Function Target and parametrization • Signalling data for the data collection operation (logs included)

2.2 Data Persistence and Message Queue Functions

To circulate data among the various LOCUS Platform system blocks, functions must be implemented by both the LOCUS Analytics Functions (i.e., adapters) and also dedicated systems that will provide persistence and message queue operations.

2.2.1 Data Persistence Functions

Given the heterogeneity and diversity of data sources, and data types in general, that, as described in section 2.1.1, LOCUS needs to support, a data lake approach is required to properly manage the various data collected, but also produced and processed by the localization and analytics related functions described in Section 2.

The LOCUS data store indeed can be seen as a logically centralized repository for heterogeneous structured (row and column) and unstructured non-tabular raw data in its native format. In practice, it offers the possibility to store data in open-source, standardized formats, i.e. without the need to define or impose proprietary data structures or schemas, to enable data analytics on top of it. As described in the previous sections, the LOCUS data store allows different data ingestion modes, including real-time, batch and data streams, and the various LOCUS functions can also use it to store their processed data, in either structured or unstructured formats. The various security and privacy functions listed in Section 2.8 are also highly linked to the data management functions offered by the data store.

In general, the LOCUS data store provides data manipulation, transformation and persistence functionalities that can be leveraged by all the other LOCUS system blocks and functions to access, persist and manage the data they require. A list of these data management functions is provided in the table below:

Table 2.5 LOCUS data management functions

Functionality	Type	Description
Unstructured Data Persistence	Persistence	This function is used to persist object-based and document-based data (e.g. blob/serialized/text) into the LOCUS data store. The retrievability of this object will be based on the key semi-structured (JSON) meta-data provided.
Structured/ Relational Data Persistence	Persistence	This function is used to persist multi-dimensional datasets of numerical and categorical features into the LOCUS data store. The retrievability of these data is based on SQL6-compliant methods.
Unstructured Data Query	Query	This function is used to gain access to unstructured data that are stored in the data store. It is parametrized by the query definition on the unstructured dataset's index.
Structured Data Query	Query	This function is used to gain access to structured (tabular) data that are stored in the data store. It is parametrized in an SQL-compliant language/format query.
Structured Dataset Transformation to Unstructured or Semi-Structured	Transformation	This function refers to the transformation of multidimensional tabular-based datasets into semi-structured or unstructured forms to be used by a LOCUS localization or analytics function. The implementation of the mapping logic needs to be provided as input.
Unstructured Dataset Transformation to Structured	Transformation	This function refers to the transformation of an unstructured object into a multidimensional structured dataset. The mapping function, as well as the target dataset structure/format, are to be provided by the instantiation of this function.
Unstructured Dataset Transformation to Structured	Transformation	This function refers to the transformation of an unstructured object of a given type into another unstructured object. The mapping logic is to be provided as input to this transformation function.

6 SQL – Structured Query Language

2.2.2 Data Message Queue Functions

In addition to data persistence, the LOCUS Platform is required to perform message exchange operations via a centralized (or federated) message broker. The different data delivery paradigm is required due to the streaming nature of some of the LOCUS analytics functions and it is also an alternative northbound data provisioning operation (for 3rd party applications).

Table 2.6 LOCUS data message queue functions

Functionality	Description
Message Queue Topic Generate / Destroy	The platform must provide function to create / destroy a message queue topic on demand based on the activated LOCUS analytics functions and their operational statuses
Internal Message Queue Produce / Consume Data	The platform must provide functions (adapter) for the consumption or production of data from a topic of the internal message queue
3rd Party Message Queue Data Consume	The platform must be able to provide access to the message queue (via securitized and privatized interface) to deliver streaming analytics in 3rd party application consumers

2.2.3 Persistence to Message Queue Interoperability Functions

The LOCUS Platform system is designed following mixed data source of truth principles, Lambda [11] and Kappa [12] architecture. All the data sources that exist in either the persistence domain or the message queue domain must be able to create an up-to-date copy of its information in the other data source. This means that the platform must provide functions to convert both streaming and persistent data into their respective counterpart. The functions defined in this section ensure this interoperability.

Table 2.7 LOCUS persistence to message queue interoperability functions

Functionality	Description
Persistent data change data capture function into messages	For the conversion of the batch-save and batch-update operations into LOCUS message queue events, the system must be able to support CDC (Change Data Capture) monitors on the various schemas in order to produce duplicate outputs as messages
Message Broker generic sink into landing table function	For the conversion of the various LOCUS messages into entries on existing schemas, the LOCUS Platform must provide implementation for a data sink from the message queue into a designated landing schema

2.3 Platform Control Functions

The LOCUS infrastructure is a platform for the onboarding and execution of multiple analytics serviced in an orchestrated and flexible manner. Therefore, it requires to implement some platform control functions that are the main Platform as a Service (PaaS) features supporting all these capabilities. The platform control functions are split into different categories according to their level of application: a) the data operations functions, which are the high level orchestration functions of the platform, with visibility of the platform analytics services demands based on external or internal triggers; b) the analytics function service coordinator functions, which implement the mediation between analytics access demand, availability and the virtualization layer; and c) on the lowest level, the functions that facilitate the high performance computing, data pipeline execution virtualization and coordination, hyperparameter tuning and ML model lifecycle support. All the above form a framework that can cover the platform demands of the various LOCUS analytics functions designed in this and previous deliverables [1][2].

2.3.1 Analytics Function Service Coordination functions

This section of LOCUS functions revolves around the ability to manage instantiation of analytics services according to the demand deriving from the platform operation requests. These requests can be self-triggered (i.e. from an internal pipeline) or they can be based on some external 3rd party application user event that gets translated in the Data Operations Controller. Whichever the case, appropriate functions are required for the health management of spawn analytics services and for the instantiation (on demand) of such instances. In addition, these functions require direct communication with an analytics image repository function to retrieve the necessary binary image data.

Table 2.8 Function: Analytics Function Service Instantiation / Despawn

Function: Analytics Function Service Instantiation / Despawn
Input: • The desired analytics service image name • The replication factor (i.e. number of containers requested) or despawn operation
Description: This function is the main control point for activation and replication of LOCUS analytics functions. Multiple cases exist for triggers to this function (e.g. 3rd party user access registration and cold start operations, 3rd party application user request and data operations, replication requests for scheduled jobs). Every invocation of this function results into a direct instantiation of a replica (or singleton) of an analytics function that will be used

for the execution of an analytics-related task or to serve a direct analytics via the reverse proxy mechanism.
Output: • Interfacing outputs to the virtualization layer

2.3.2 Analytics Service Catalogue

The Analytics Service Catalogue is used to collect all the analytics service dependencies and requirements in terms of deployment, execution and runtime operation. It references existing descriptors related to the implementation of the analytics service and related functions in the form of virtualized functions and services through the Management, Orchestration and Virtualization layer. It also specifies the capabilities and execution requirements of the analytics service and its other dependant functions (i.e. analytics functions), including specific input and output data requirements, execution plan (e.g. in the form of a direct acyclic graph).

Table 2.9 Function: Analytics Service Catalogue Entry Persist / Retrieve

Function: Analytics Service Catalogue Entry Persist / Retrieve
Input: • The desired analytics service meta-data to declare the service capabilities and requirements • Reference to the virtualization descriptors for virtual resource requirements and software images to be used to deploy the service
Description: This function refers to the process of adding or removing a new analytics service's capabilities and characteristics information into the Analytics Service Catalogue. It is part of the onboarding phase of LOCUS analytics services and functions (or their end-of-life) and ensures that the appropriate information for the analytics service coordinator is available and up to date to request the virtual resource allocation operations towards the Management, Orchestration and Virtualization layer of the LOCUS Platform, and then apply necessary service execution logic to properly run the analytics service.
Output: • The reference ID of the saved analytics service (for retrieval purposes)

2.3.3 Data Operations Functions

The Data Operations Functions handle the core orchestration of multi-service interoperability, heavy batch processing, background processing and batch read/ write operations. They are connected directly to all the events/ triggers of the system meaning that any request for

processing interfaces directly with this functional block which will in turn translate the request into lower-level commands (via the analytics service coordination). LOCUS analytics services have intricate dependencies in the form of data, other analytics services, message events and variable resource demands. All this high-level pipeline coordination must be encapsulated in the functions of this block in a centralized and stable scheduler that will ensure the timely outcomes for all the analytics' (end-to-end) results.

Table 2.10 Function: Analytics Service Configuration Processing

Function: Analytics Service Configuration Processing
Input: • The desired analytics service configuration input • The internal dependencies of the analytics service function (to generate children)
Description: This function is the initializer of LOCUS analytics service pipeline. Reads the input configuration request of the specific analytics service and the internal dependencies of the service (e.g. LEN function plus SNM function) and requests their instantiation from the lower-level functions (analytics service coordination). Afterwards, it passes the configuration parameters to each children service instance. Given all the necessary requirements are met, the pipeline is being executed via the designated communication interfaces.
Output: • Appropriate message request for the analytics service coordinator functional block to instantiate/ de-instantiate a service instance • Pipeline triggering request to the initial function sub-task

Table 2.11 Function: Analytics Service Configuration Processing

Function: Analytics Service Health Check
Input: • A specific data operations pipeline to check • A specific analytics service instance to perform remote status analysis
Description: The Data Operations Controller is using parallel and sequential analytics service subtask execution in the form of a directed acyclic graph. To do so, it requires to have knowledge of the status for the instantiated analytics service sub-task nodes. After the configuration request (which propagates to the instantiation layer), the spawned services must provide health information to the Data Operations Controller to start the chain execution. The same

endpoint is used to check if the specific task is finished or if it has failed, with appropriate actions in each case.
Output: • Status of availability (waiting, running, error) of the analytics service instance to begin the next phases of the data processing

2.3.4 Machine Learning, Optimization, Pipeline Functions

In the following subsections, the elementary ML /DL-related functions are presented. These functions -appropriately parametrized- will be chained in ML pipelines so as to be exposed through programmatic APIs LOCUS to higher level analytics.

The elementary ML/ DL-related functionalities offered by the LOCUS Platform are presented in this section. The following tables provide details for these LOCUS functions, in terms of data requirements and main features provided in the context of the LOCUS UCs.

Table 2.12 Function: Data cleaning

Function: Data cleaning
Input: • Multi-dimensional raw dataset (structured or semi-structured) • Data cleaning specification that includes the order of operations and parameters of each operation
Description: This function refers to the data cleaning pipeline that will take as input a raw, unprocessed data as found in the persistence context of the LOCUS Platform. The function will include several data cleaning operations such as impute/interpolate/predict the missing data values, smoothen the data using regression or other methods, remove outliers using simple clustering or other methods, create attributes from the existing attributes, if required.
Output: • Multi-dimensional numerical structured dataset as it results from the data cleaning function

Table 2.13 Function: Dataset Pre-processing

Function: Dataset Pre-processing
Input: • Multi-dimensional raw dataset (structured or semi-structured)

• Pre-processing pipeline specification including order of operations and parametrization per operation
Description: This function refers to the processing pipeline that will take as input a raw, unprocessed dataset as found in the persistence context of the LOCUS Platform and prepare it to be used into various analytical and ML functions that require numerical representations. This function can include several operations like data transformation, data encoding to other formats like binary, ordinal or one-hot, categorical or label-encoding, feature scaling (data standardization and normalization), creating training and test splits for a supervised learning.
Output: • Multi-dimensional numerical dataset as it results from the pre-processing function

Table 2.14 Function: Multiple Regression

Function: Multiple Regression
Input: • Multi-dimensional numerical dataset • Selected hyperparameters (linear/polynomial, degree, etc.)
Description: This function refers to training a regression model based on two or more independent variables (or regressors or predictor variables) to predict a dependent forecast variable.
Output: • A regression model from the training • Predictions/forecasts from the regression model • (Optional) Performance evaluation of the training

Table 2.15 Function: Multivariate or Spatial Interpolation

Function: Multivariate or Spatial Interpolation
Input: • Multi-dimensional numerical dataset • Selected method and parameters

Description:
This function refers to a task to estimate the variables at unobserved geo-spatial locations based on the data from the observed locations. Existing methods like inverse weighted distance (IDW), kriging, nearest neighbours (NN), thin plate spline (TPS), radial basis function (RBF), local polynomial interpolation (LPI) can be used for spatial interpolation.
Output: • Spatially interpolated values for the missing input locations/observations

Table 2.16 Function: ML-based Dimensionality Reduction

Function: ML-based Dimensionality Reduction
Input: • Multi-dimensional numerical dataset to be reduced by the vector quantization method • Selected algorithm/method • Selected hyperparameters with respect to the algorithm
Description: This function refers to the transformation of a multi-dimensional numerical dataset into a reduced representation of itself by the means of dimensionality reduction schemes. This function will contain both the training and the reduction phase. This function can include existing ML-based dimensionality reduction techniques such as Principal Component Analysis (PCA), random forest, t-Distributed Stochastic Neighbour Embedding (t-SNE), uniform manifold approximation and projection (UMAP).
Output: • Multi-dimensional numerical dataset that is the quantized representation of the input dataset

Table 2.17 Function: Feature Ranking and Selection

Function: Feature Ranking and Selection
Input: • Multi-dimensional numerical datasets (features) for feature selection • Selected algorithm/technique • Selected hyperparameters for a technique

Description: This function refers to the process of selecting highly informative variables (features) that are useful to build an efficient predictive model. This function can include ML algorithms based on random forests like Boruta and/or RRF variable importance; LASSO regression, recursive feature elimination, stepwise forward and backward selection, simulated annealing etc.
Output: • Multi-dimensional numerical dataset that is the filtered and reduced from the input dataset

Table 2.18 Function: Hyperparameter tuning of a ML task

Function: Hyperparameter tuning of a ML task
Input: • Reference to specific instantiation of ML Task • Hyperparameter Space related to the selected ML Task • Methodology of tuning and hyperparameters of the search operation
Description: This function refers to a methodology in which an internal ML task is being repetitively executed with a goal to either minimize or maximize a specific ML KPI (e.g. minimization of mean square error)
Output: • Optimum Execution Result (ML Model) for the specific input hyperparameter space and referenced ML task

Table 2.19 Function: Classifier - Training

Function: Classifier - Training
Input: • Multi-Dimensional Dataset • Selected Classification Algorithm • Selected hyperparameters based on the Selected Algorithm

Description: This function refers to the training process of a ML model that will be used as a classification model for an input dataset. Exiting ML techniques like Support Vector Machine (SVM), k-nearest neighbours, decision trees can be used for this function.
Output: • ML model as it resulted from the training function • (Optional) Performance evaluation of the training function • (Optional) Multi-dimensional dataset including the predicted classes of the classification

Table 2.20 Function: Classifier - Prediction

Function: Classifier - Prediction
Input: • Classifier model that will be used for the predictions • Multi-dimensional dataset that fits the specifications of the classifier model • (Optional) Additional multi-dimensional dataset that will be used for validation purpose
Description: This function refers to the prediction process of a classification ML Model. It takes as input a multi-dimensional numerical dataset that fits its input specifications and executes the prediction step, generating predicted labels for each data point.
Output: • Multi-dimensional dataset with the predicted class(es) per data point based on the selected model • (Optional) Evaluation of error of the KPIs based on the validation dataset

Table 2.21 Function: Clustering Model

Function: Clustering Model – Training and Inference
Input: • Multi-Dimensional numerical dataset • Selected clustering algorithm • Selected hyperparameters with respect to the selected algorithm

Description: This function refers to the execution of ML training and predict (merged) task that will generate information for underlying data clusters (based on the input dataset), and it will also produce a per-data-point cluster classification. The existing approaches like K-means clustering, k-medoids clustering, k-paths large scale trajectory clustering and/or density-based clustering (DBSCAN, HDBSCAN) will be used for this function.
Output: • Description of the identified clusters in the form of meta-data (e.g. centroids) • Per data point classification with a cluster from the result clusters • (Optional) Performance evaluation of the clustering execution

Table 2.22 Function: Space-time trajectory clustering

Function: Space-time trajectory clustering
Input: • Multi-dimensional numerical dataset • Selected hyperparameters
Description: This function refers to the execution of a ML algorithm to identify clusters in spatio-temporal datasets using agglomerative hierarchical clustering. Traditional clustering based on statistical techniques often fail when applied to moving objects and big data. This function based on space-time hierarchical clustering incorporates location, time, and attribute information to identify the groups across a nested structure reflective of a hierarchical interpretation of scale.
Output: • Identified clusters or hotspots and shape of the clusters that indicate the movement patterns for tracking • (Optional) Performance evaluation of the clustering using metrics like homogeneity score and average random index score

Table 2.23 Function: Time series predictions/forecasting

Function: Time series predictions/forecasting
Input: • Time-series input data, intervals (if any used), and target variable for prediction

• Constituent components of time-series: level, trend, seasonality, noise (if the components are available) • Selected ML/DL approach and related hyperparameters
Description: This function refers to the training of a ML/DL model for time series forecasting/predictions. This function can include baseline methods like simple naïve or persistence forecasting and averaging methods, autoregressive forecasting method (ARIMA), or DL methods that using LSTMs, Convolutional Neural Nets (CNNs), or hybrids of CNN and Long short-term memory (LSTM) models, such as CNN-LSTMs, ConvLSTMs.
Output: • Predictions for the target variables • Trained ML/DL models from this approach • Optional performance evaluation of the ML/DL model

Table 2.24 Function: Static or mobile group detection

Function: Static or mobile group detection
Input: • Multidimensional numerical data set that contains geolocations generated from localization functions, coordinates of areas under investigation, time window under investigation • Selected approach and related hyperparameters
Description: This function refers to the execution of a ML-based method to detect crowd groups in short-time intervals and long-time linkages using multimodal data (like bluetooth, wifi etc.) in an indoor or outdoor environment. This function includes graph-based clustering algorithms such as density-based scanning approach on graph models (DenGraph), clustering highly connected subgraphs (HCS) or fully connected subgraphs (MaxClique).
Output: • Description of the identified static or mobile groups and group size • (Optional) Performance evaluation of the selected approach

Table 2.25 Function: Classifying moving object trajectories using representation learning

Function: Classifying moving object trajectories using representation learning
Input: • Multidimensional numerical data set that contains geolocations generated from localization functions with time stamps (trajectories of moving objects) • Annotations (labels) for the mode of transportation • Model hyperparameters
Description: This function refers to a DL-based method to estimate users' transportation modes from their movement trajectories. The steps involved in this method: generating informative trajectory images from raw trajectories, feature extraction from trajectory images using a fully-connected deep neural network with stacked denoising autoencoder, train a classifier (SVM, decision tree, or logistic regression) using the extracted features and annotations, and estimating the transportation modes using this classifier.
Output: • Estimates of the transportation mode • Trained ML/DL models from this approach

Table 2.26 Function: Deep probabilistic clustering with self-organizing maps

Function: Deep probabilistic clustering with self-organizing maps
Input: • Multidimensional numerical dataset (preferably time series data/trajectories) • Model hyperparameters
Description: This function refers to a DL-based method that combines clustering and representation learning to generate interpretable visualizations. This function uses self-organizing maps with probabilistic clustering assignments and includes a variational auto encoder extended to time-series probabilistic clustering.
Output: • Time series clustering and forecasting (predictions of target variables) • Generated interpretable visualizations of the clusters • Trained DL model (T-DPSOM) from this function

Table 2.27 Function: Deep spatio-temporal network with data fusion

Function: Deep spatio-temporal network with data fusion
Input: • Multidimensional numerical data set that contains geolocations generated from localization functions with user ids, maximum uplink and downlink throughputs • Model hyperparameters
Description: This function refers to a DL-based method to estimate the network demand in terms of maximum uplink and downlink throughputs. This DL model employs convolutional-based dense networks to model both nearby and distant spatial dependencies between regions of the cities and includes several branches for fusing external data sources of different dimensionality. This DL model uses three different dense networks to model various temporal properties consisting of closeness, period, and trend.
Output: • Estimates of network demand (maximum uplink and downlink throughputs) • Trained DL model from this approach

Table 2.28 Function: Reinforcement Learning/Multi-armed Bandit

Function: Reinforcement Learning/Multi-armed Bandit
Input: • Multidimensional numerical data set that contains geolocations generated from localization functions with user ids and radio environment maps • Model hyperparameters
Description: This function refers to a ML-based method to provide improved scheduling decisions, to enhance user throughput, and a better bandwidth utilization. This function includes epsilon greedy, decayed epsilon greedy, softmax exploration, and upper confidence bound as solution strategies.
Output: • Scheduling decisions • Trained DL model from this approach

Training and Deployment Specific Pipelines: This section presents different training and deployment pipelines based on the ML functions described above. The ML/DL modelling options for the LOCUS Platform remain open, as the various deployment pipelines will have to cater to the needs of the UCs described in D2.1 [1], which are currently under development. Indicative pipelines allowing for an offline stream -possibly in a parallel to the LOCUS Platform inference/prediction stream and even outside of the LOCUS Platform will be considered. Online model training may also be within the LOCUS project scope, if it is deemed necessary for the purposes of the UCs deployed. Currently, some indicative -but subject to change- examples of ML pipelines are presented in the following subsections.

A) Training ML/DL models One-off (Offline/batch training) and Inference (Predictions) on the fly: In this approach a pre-trained ML model (i.e. a model both trained and evaluated offline using a pre-determined dataset) is used.

Example A1: Network demand forecasting (SNM-UC1: Knowledge building for network management, described in detail in [1])

Figure 2.2 exemplifies the case of offline training and inference on the fly for this UC.

Offline Training: Existing datasets after cleaning and pre-processing are used to train a DL model to estimate the network demand in terms of maximum uplink and downlink throughput. The outcome of this pipeline is a trained DL model for network demand prediction.

Inference on the fly: The new data after cleaning and pre-processing is used as test data for predictions (inference) by the pre-trained DL model. The outcomes of this pipeline are the predictions for network demand (max uplink and downlink throughput).

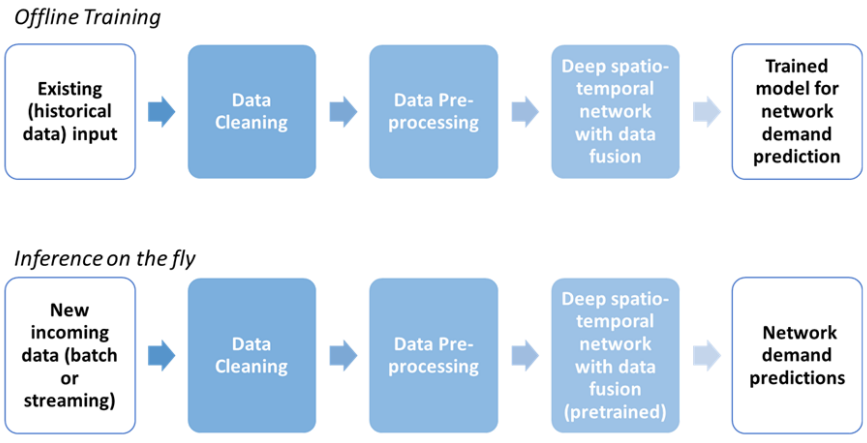

Figure 2.2 Network demand forecasting ML pipeline

Example A2: Learning group mobility characteristics using wireless fingerprints (NSE-UC2: Crowd mobility analytics using mobile sensing and auxiliary sensors, as described in [1])

Similarly, for example 2, the ML pipelines are presented in Figure 2.3.

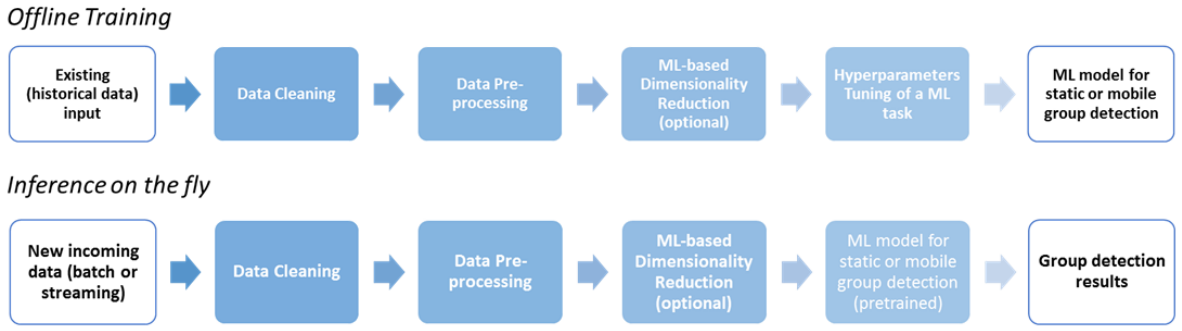

Figure 2.3 ML pipeline for learning group mobility characteristics using wireless fingerprints

Offline Training: Existing datasets after cleaning and pre-processing are used to train ML models to detect groups in the crowd movements. ML-based dimensionality reduction and hyperparameters tuning of a ML task can be included as optional stages before training. The outcome of this pipeline is a pretrained ML model for the group detection in crowds.

Inference on the fly: The inference pipeline uses a similar set of functions like the training pipeline except that the dimensionality reduction process (if applicable) is predetermined and in the final stage the pre-trained ML model is applied instead of training a new ML model.

B) Real-time/Online Training and Inference: In the online learning (also called as incremental learning) the ML models are trained in real-time and tested (inference) on the fly with the new data. A related example follows.

Example B1: Identifying crowd mobility patterns (NSE-UC1: Flow monitoring and management in large venues and dense urban environment)

Figure 2.4 ML pipeline for Identifying crowd mobility patterns UC

The new data after cleaning and pre-processing is split into training and test data. This pipeline can include optional stages like ML-based dimensionality reduction and hyperparameters tuning of a ML task after selecting a specific ML model/algorithm. The training split is used for training a selected ML model and the test split is used for predictions, both happening in real time. The ML functions in this pipeline include a provision for online training/incremental learning from the new data. In this functionality, the outcomes are identified clusters from the crowd movements. The described pipeline is presented in Figure 2.4

2.4 Localization Enabler Functions (LEN)

The Localization enablers provide the localization mechanisms by utilizing the properties of radio signals associated with a given UE (or the user). These enablers studied in the LOCUS project are divided into 3 categories. These are the following:

- *3GPP Technologies:* The localization of a UE is performed using the collected radio signals that have been transmitted from 5G NR, LTE, UMTS or GSM antennas.
- *Non-3GPP Technologies:* The localization of a UE is performed using the collected radio signals that have been transmitted from e.g. WiFi Access Points etc.
- *Device-free Technologies:* The localization of a UE/object is performed using a passive monitoring system that can locate the object position without the subject's participation, where the subject does not need to carry any radio device.

In all of the above types of localization, the user consent for this in the privacy settings is taken into account, except for emergency and regulatory cases (where the laws allow for). A dedicated section related to location data privacy and security follows.

The following tables define the three main localization enabler functions templates.

Table 2.29 Function: 5G-based LEN

Function: 5G-based LEN
Input: • 3GPP defined radio measurements to compute time delay of arrival (Positioning Reference Signal-PRS and Sounding Reference Signal-SRS), angle of arrival/departure (Single-Sideband modulation-SSB components), received signal strength (Reference Signal Received Power-RSRP, Reference Signal Received Quality-RSRQ, Received Signal Strength Indicator-RSSI) and cell ID indications (CID and e-CID). • Locations of fixed and/or ad-hoc access points (these through GPS or Global Navigation Satellite System - GNSS) • Physical and radio geometrical data of the area of interest. • Accuracy, reliability and update rate criteria
Description: This function's purpose is to localize any type of 5G mobile device, by leveraging multi-RAT, multi-carrier and mmWave technologies.
Output: • The derived location of the device, which should meet the accuracy, reliability and update rate criteria as set by the service KPIs

Table 2.30 Function: Hybrid (5G & non-3GPP) LEN

Function: Hybrid (5G & non-3GPP) LEN
Input: • 3GPP based localization inputs as noted above. • Non-3GPP localization inputs related to received signal power, time delay, angle of arrival/departure and access point IDs. • Area of interest geometrical data (definition of physical sections, streets, areas and definition of radio coverage geometry, location of 3GPP and non-3GPP access points. and coverage zones/polylines) • Accuracy, reliability and update rate criteria
Description: This function's purpose is to fuse 5G network data and data/information derived from non-3GPP technologies with the explicit purpose of enabling device localization.
Output: • The derived location of the device, which should meet the accuracy, reliability and update rate criteria as set by the service KPIs

Table 2.31 Function: Device-free LEN

Function: Device-free LEN
Input: • Cellular and/or non-cellular signals for localization of device-free targets. • Locations of the cellular and/or non-cellular signal emitters. • Accuracy, reliability and update rate criteria
Description: This function's purpose is to enable the detection and tracking for single and multiple device-free targets.
Output: • The derived location of the target, which should meet the accuracy, reliability and update rate criteria as set by the service KPIs

2.5 SNM and NSE (Core) Analytics Functions

LOCUS can provide various analytics that leverage geolocation data, either in real-time or non-real-time. These analytics functions can be integrated and chained within wider scope localization analytics services (e.g. including cascades of analytics and ML functions), or be consumed and directly exposed to Smart Network Management and vertical/3rd party applications, so as to enable use cases related to the following categories:

- People Mobility & Flow Monitoring.
- Network-Assisted Self-Driving Objects.
- Smart Network Management based on Localization.

Coupled with the ability of the LOCUS Platform to deploy ML/ DL models and expose the Localization Analytics as a Service for the Application layer, these functions enable advanced analytics capabilities leveraging position information and encompass the following:

- *Descriptive Analytics*: Perform statistical analysis on collected data (What happened?).
- *Predictive Analytics*: Assess future possibilities, given collected data (What can happen?).
- *Prescriptive Analytics*: Search for actions to be taken, given data (What to do to make it happen?).
- *Diagnostic Analytics*: Determine the causes for specific outcomes from data (Why did it happen?).

The following tables present the Statistics/ML/DL-based functions in terms of data requirements and main functionalities provided.

Table 2.32 Function: Trajectories Identification

Function: Trajectories Identification
Input: • Dataset of historical geolocation data as generated from localization functions • Area coordinates under investigation • Time window in which trajectories will be extracted
Description: This function refers to the identification of mobility trajectories extracted from specific areas of interests, and within a specific time window of interest. This functions output is agnostic to the localization source, identified trajectories could come from the positioning source considered the most accurate at the moment of execution, or could come from the fusion of

different sources information. This function could trigger a predictive model to forecast trajectories in case the request time window includes future timesteps.
Output: • List of identified distinct trajectories present in the designated area and the designated time window

Table 2.33 Function: POI Identification

Function: POI Identification
Input: • Dataset of historical geolocation data as generated from localization functions. • Area coordinates under investigation
Description: This function refers to the identification of geometric areas of interest (Points of Interest - POIs) that result from high population density and concentration. In general, we expect that marketplaces, shopping malls, popular streets and venues will be identified by this function.
Output: • Geometric Representation of the POIs persisted in the LOCUS persistence context

Table 2.34 Function: Routes identification

Function: Routes identification
Input: • Dataset of historical geolocation data as generated from localization functions • Coordinates of Point of Origin • Coordinates of Point of Destination • (Optional) Coordinates of Intermediate Points • Time window/constraint
Description: This function refers to the identification of one or more possible path(s) that lead from a Point of Origin to a Point of Destination under the time constraint requested. In general, we expect that this function identifies the best route, but in case of multiple paths are possible, multiple options will be provided. In case no option is possible, no path will be provided, instead, estimation of the minimum time constraints could be provided. This function can trigger a predictive model in case the request time is in the future.

Output:
• List of identified routes (timed trajectories) that LOCUS estimates will lead from Point of Origin to Point of Destination in the specified time window • An estimation of minimum time window, in case the request cannot be satisfied

Table 2.35 Function: Flows identification

Function: Flows identification
Input: • Dataset of historical geolocation data as generated from localization functions • Coordinates of areas under investigation • Flow tiles configuration • Time window under investigation
Description: This function refers to the identification of the Origin Destination Matrix, of velocity fields/vectors over the designated area, computed within the designated time window. This function can trigger a predictive model in case the request time spans future timesteps.
Output: • An Origin Destination Matrix of the mobility flow (e.g. as represented by velocity fields)

Table 2.36 Function: Counts identification

Function: Counts identification
Input: • Dataset of historical geolocation data as generated from localization functions • Coordinates of areas under investigation • Measurement Time
Description: This function refers to the identification of how many separate individuals are present in a designated area at a specific time. By means of sensor radar networks, counting can also be performed without the active participation of any individuals' devices (device-free approach). This function can trigger a predictive model in case the requested time is in the future.

Output:
• A count of individuals identified in the designated area at the requested time

Table 2.37 Function: Feature Changes Tracking

Function: Feature Changes Tracking
Input: • Continuous stream of historical geolocation data as generated from localization functions • Coordinates of areas under investigation • Feature of interest, with a specific change condition to be tracked • Optional Time window
Description: This function refers to a subscription request to track a designated feature (velocity, position, acceleration, direction, proximity, grouping, etc.) of an individual/ trajectory or a group/ trajectory in a designated area within (optionally) a designated time window.
Output: • A notification with a list of elements (groups, individuals, trajectories...) when the requested condition is satisfied

Table 2.38 Function: Time to Collision

Function: Time to Collision
Input: • Continuous stream of historical geolocation data as generated from localization functions • Coordinates of areas under investigation
Description: Time-to-Collision (TTC) estimates the likelihood of a collision of moving objects. See TTC calculation example for Vulnerable Road user (VRU) in [13], section 6.5.10.5.
Output: • Estimation of Time to Collision and Likelihood of Collision

Table 2.39 Function: Profiles Identification

Function: Profiles Identification
Input: • Dataset of historical geolocation data as generated from localization functions • Coordinates of areas under investigation • Optional list of user/trajectory IDs
Description: Within the designated area, this function identifies the provided trajectory/ user's mobility profile, from a set of possible profiles. If no IDs are requested, identify the profiles of all available trajectories in the designated area.
Output: • A mapping between Trajectory/User IDs and their profiles

Table 2.40 Function: KPI heatmaps

Function: KPI heatmaps
Input: • Dataset of historical geolocation data as generated from localization functions • A list of KPIs of interest, with their parameters (geographic scope, time window, etc.)
Description: This function refers, to the generation of a heatmap view, from the historical geolocation data. This function might trigger a predictive model to generate information if the requested time window spans future timesteps. The KPIs may refer to historical and/ or near future KPIs, such as network-related measurements (e.g. Signal to interference and noise ratio - SINR) and may refer to selected subareas that can be ML-derived, such as mapping KPIs in detected trajectories.
Output: • Visualization ready, list of heatmap data structures

Table 2.41 Function: Contextualized Indicator Generation

Function: Contextualized Indicator Generation
Input:

• Dataset of structured historical data of interest, that include UEs' time-series position and network measurement data • Base station positions • (Optional) Coverage maps or propagation models results • Information on the areas of interest to be considered. Optionally this function may rely on an automatic generation of areas of interest • Parameters related to geographic scope and time window under consideration
Description: This function refers to the fusion of diverse time-dependent data in a selected area and time-frame from the historical geolocation data in order to create new indicators of time-series format as a result of this data fusion. These new indicators can be directly used as input for inference mechanisms (i.e. fed to other analytics services referring to different functional (sub)blocks as an alternative to full data input) and improve their results in comparison to employing purely network measurement-based inputs. To reduce the generated possible huge number of indicators (due to the huge number of possible areas of interest to be considered) ML algorithms for dimensionality reduction and feature selection can be applied after the generation of the indicators.
Output: • New contextualized indicator values

Table 2.42 Function: KPI Prediction

Function: KPI Prediction
Input: • Targeted network KPI(s) to be predicted • Set of data to be included in the prediction process, including network, position/mobility and other external data • Modelling (Analytics/ ML/ DL) approach and related parameters employed • Area coordinates and time-frame under investigation
Description: This function uses ML and/ or DL methods to predict network-related KPIs, such as network demand and Quality of Experience metrics, in the near future.
Output: • Time-stamped predicted values for selected KPI(s)

Additional LOCUS basic analytics functions, based mostly on metric/location data manipulation and statistics are described in the following tables.

Table 2.43 Function: Geometric Area Correlation

Function: Geometric Area Correlation
Input: • A set of geometrical areas to be correlated • Parametrization of the correlation process (incl. correlation method)
Description: This function refers to the computation of correlation between a set of selected input geometrical areas.
Output: • Correlation results (value per distinct area pair)

Table 2.44 Function: Data Filtering

Function: Data Filtering
Input: • Multidimensional dataset (structured or unstructured) • Filtering criteria (metric, time, or entity context) • Desired columns or sections of the original dataset
Description: This function refers to the transformation of a dataset to a subset of itself by applying computational logic.
Output: • New multidimensional dataset (filtered) based on the filtering criteria and the selected columns

Table 2.45 Function: Metric aggregation

Function: Metric aggregation
Input: • Multidimensional Dataset (Structured) • Aggregation Criteria (selected KPIs, Aggregation Columns, Metric Aggregation Functions – average, sum, maximum etc.)

Description: This function refers to the transformation of a dataset into a new dataset by applying the aggregation function and reducing its dimension to the selected aggregation parametrization.
Output: • New dataset (with aggregated metric values) based on the aggregation criteria

Table 2.46 Function: Categorical Aggregation

Function: Categorical Aggregation
Input: • Multidimensional Dataset (Structured) • Aggregation Criteria (Selected Categorical Features, Aggregation Columns, Categorical Aggregation Functions – Majority, Distribution etc.)
Description: This function refers to the transformation of a dataset into a new dataset by applying a categorical-specific aggregation function that will reduce its dimension to the selected aggregation parametrization.
Output: • New dataset (with aggregated categorical values) based on the aggregation criteria

Table 2.47 Function: Structured Dataset Joining

Function: Structured Dataset Joining
Input: • More than one, multi-dimensional datasets • Criteria to be used for the joining operation (e.g. join on column1 = column2)
Description: This function refers to the transformation of a set of structured multi-dimensional datasets into a single, joined dataset, by the means of a structured joining operation.
Output: • The new generated joined structured dataset

Table 2.48 Function: Geo-query

Function: Geo-query
Input: • List of geometric data points (latitude, longitude) • List of geometric areas
Description: This function refers to the process of identifying the data points (latitude, longitude) pairs that belong to or are relevant to a specific geometric area.
Output: • Report of correlation between the points and the areas

Table 2.49 Function: Reverse Geocoding

Function: Reverse Geocoding
Input: • Input geolocation data (single, multi data points) • Geocoding specification (contains, adjacency parameters)
Description: This function refers to the transformation of input geolocation data into geometric areas that are correlated or contain these location data.
Output: • List of Geometric Areas that are correlated with this location • (Optional) Adjacency, relevance, accuracy

Table 2.50 Function: Geometric Area Manipulation

Function: Geometric Area Manipulation
Input: • Geometric areas that will be involved in the manipulation operation • Type of operation (intersection, union, split)
Description: This function refers to the transformation of new geometric areas based on other geometric areas, manipulation operations and parametrization. The result will be a new, edited geometric area.

Output:

- New geometric areas generated by the manipulation operation

2.6 Analytics API Functions

2.6.1 Analytics Service Discovery Function

The internal analytics function instances are designed to be generated in a demand-based manner allowing for flexibility and scalability. Each of these instances require to register their respectful internet and connectivity information for an easier access (via a designated naming convention).

2.6.2 User Access Control Manipulation

This function is related to the linking between LOCUS Platform users and their respective activated analytics functions. It is basically an interface between the security functions (Authorization) and then service subscription function, synchronizing their status.

2.6.3 Analytics Service Metadata Manipulation (Onboarding/ Edit)

This function refers to the changing (creation, deletion, update) of meta-data in the analytics API catalogue which acts as the meta-data persistence for the LOCUS Platform API control panel.

2.6.4 Analytics Service Subscription

The LOCUS localization services that are offered by the platform to the LOCUS applications can be seen as the composition of one or more LOCUS functions that need to be wired and chained together according to their input/ output data requirements to achieve a specific analytics goal.

Indeed, a specific LOCUS application localization related requirement or need can be translated into the deployment of one or more LOCUS functions for collecting data, providing localization of users and devices, analysing it and applying ML to predict some specific behaviour or localization characteristic. In this case of pre-defined localization analytics services, such combinations of LOCUS functions can be defined in advance and properly modelled to instruct the LOCUS MANO for their automated deployment, when a given service is request by a LOCUS application.

Therefore, such service descriptors are stored in the LOCUS MANO Service Catalogue, where multiple combinations of LOCUS functions (modelled as individual VNFs) are exposed through the LOCUS Analytics APIs in the form of Network Service Descriptors (NSDs).

In practice, the exposure of these pre-defined service offers is made through a set of HTTP REST APIs that provide access to the service descriptions in the Service Catalogue. Therefore, the pre-defined localization analytics services management workflow, as supported by the LOCUS Platform and enabled by the LOCUS Analytics APIs (in accordance with the asynchronous service request/ response transactions defined above) can be summarized in the following steps:

1. *Service exposure*: a LOCUS application queries the MANO Service Catalogue and retrieves the list of localization analytics services that can be activated on-demand.
2. *Service request*: after having selected the localization analytics service of interest, the LOCUS application issue a request of service activation, specifying its constraints (at least in terms of desired service response access format, i.e. HTTP REST service vs. message bus continuous data stream)
3. *Service preparation*: the LOCUS MANO Service LCM (Lifecycle Manager) validates the service request, performs some feasibility check (in terms of available resources and required localization related data and information), identifies the LOCUS functions to be either instantiated or re-used, prepares the service consumption endpoint (in the form of HTTP REST or message bus service according to the request), that is configured and managed on the Localization Analytics Gateway, and trigger the activation and on-demand instantiation of the various LOCUS functions
4. *Service response*: in the synchronous service response, the LOCUS MANO Service LCM provides to the given LOCUS application a service identifier (to be used for service status queries and subsequent operations, e.g. termination) and the details of the service consumption endpoint
5. *Service activation*: in this step, the Service LCM takes care of the service activation, creating all of the required LOCUS functions instances and configuring their interconnections in the virtualized infrastructure to properly realize the requested service according to the requirements in terms of input/output data for each function.
6. *Service consumption*: once the service is ready to be consumed, the LOCUS application can access the service consumption endpoint and start accessing the localization analytics data of interest, through the mediation of the Localization Analytics Gateway

This catalogue based way of exposing services towards the LOCUS application layer follows a traditional telco-oriented approach, also part of well-defined standard approaches in TM Forum (with the Information Framework – SID [14]) and ETSI NFV itself (with the Network Service and VNF Descriptors models, see also section 2.7.3) for example, and make the LOCUS Platform aligned with related telco service management and orchestration solutions. In particular, this can be practically pursued by adopting de-facto standard existing network and service management open-source tools as a baseline for the LOCUS MANO.

2.6.5 *Intent-based Analytics Services Subscription*

Complexity of network related services is continuously growing due to increasing number of technologies, network layers and dynamics, practically driven by virtualization. The request for fulfilment of a given service can become very complex and pose many different constraints depending on multiple factors and parameters.

Intent is a way to abstract service complexity at API level. An intent can be seen as a simplified service goal, expressed in natural language and linked to a declarative policy, and intent-based APIs allow the request for service intents. In turn, an intent-based system follows two main principles:

- Translation and validation: the system translates a service intent into concrete actions and operations that the system itself can perform to achieve the service intent goal. In parallel, it verifies that the requested intent can be satisfied according to the system capabilities.
- Automated implementation: when the service intent is translated and decomposed into actions and operations, these are automatically implemented and applied to deploy the actual service instance

In practice, following these principles, the LOCUS Platform can provide an alternative mechanism (with respect to pre-defined services described above) to define and customize localization analytics services through the LOCUS Analytics APIs, adopting an approach based on service intents expressed in natural language. This is done by allowing LOCUS applications to request localization analytics service through a sentence in plain English that describe the service constraints and requirements. Such request can be issued through the LOCUS Analytics APIs, and then processed by the Intent Engine embedded in the API layer to apply translation and validation of the requested service intent.

In this case, the intent-based localization analytics services management workflow does not include an explicit service exposure phase, and can be summarized in the following steps:

1. *Service intent request*: the LOCUS application requests for a service intent issuing an HTTP REST request containing the intent sentence through the LOCUS Analytics APIs, and specifying the desired service response access format (i.e. HTTP REST service vs. message bus continuous data stream)
2. *Service intent translation*: the Intent Engine translates the service intent into localization analytics service constraints, looking for a service descriptor in the MANO Service Catalogue that could fulfil them. If not found, the Intent Engine can further investigate available LOCUS functions in the Functions Catalogue and create a new ad-hoc service descriptor to serve the intent
3. *Service preparation:* the Intent Engine requests to the LOCUS MANO Service LCM to activate the given selected service. Similar to the pre-defined service case, the Service

LCM identifies the LOCUS functions involved, prepares the service consumption endpoint and trigger the activation and on-demand instantiation of the various LOCUS functions

4. *Service response*: as in the pre-defined service case, the MANO Service LCM provides to the given LOCUS application a service identifier and the details of the service consumption endpoint
5. *Service activation*: as in the pre-defined service case, the Service LCM takes care to follow the service activation, creating all the required LOCUS functions instances and wiring them in the virtualized infrastructure
6. *Service consumption*: once the service is ready to be consumed, the LOCUS application can access the service consumption endpoint and start accessing the localization analytics data

In summary, the LOCUS intent-based approach for the LOCUS Analytics APIs can simplify the process of service definition and customization, enabling an easy access to the localization analytics services offered by the LOCUS Platform.

2.6.6 Analytics 3rd party Consumer API Gateway

The Analytics 3rd party Consumer API Gateway is the main access point for the LOCUS applications to consume the localization analytics data generated by the services they request. As part of the asynchronous localization analytics service consumption offered by LOCUS to the application layer through the Analytics Services as a Service API, the Analytics API Gateway guarantees that applications are given access only to analytics data generated in the context of their services, thus providing data isolation across multiple services and applications. For this reason, it leverages the LOCUS security and privacy functions to provide LOCUS applications secure, authorized and authenticated access to localization analytics data and store in the persistency entities part of the data store.

Through the Analytics API Gateway, the LOCUS applications can also inject additional analytics data into the LOCUS Platform that can potentially be exploited by other applications, or by the internal LOCUS functions themselves. This option is subject to be validated by the LOCUS Platform in the phase of the service request, as a service consumption constraint posed by the application itself.

Different deployment models are supported in the LOCUS Platform with respect to the Analytics API Gateway. As a special localization analytics and data exposure management function, it can be either deployed in single or multiple instances (e.g. one for each service consumed by a given LOCUS application as part of the on-demand service creation) within the LOCUS Platform.

In any case, its configuration and behaviour are under the control of the MANO Service LCM, which -according to the service constraints and characteristics expressed by LOCUS applications in the service request- instructs the Analytics API Gateway to apply a suitable analytics data exposure model. The Analytics API Gateway supports both HTTP REST and message bus-based models, and in practice is responsible for the exposure services implementation for the LOCUS APIs This service delivery model is in line with ETSI ZSM (Zero-touch network and Service Management) principles for data and analytics services exposure [15].

The LOCUS services exposed by the APIs can be grouped in three main categories, according to the type of interaction implemented with the LOCUS applications:

- HTTP REST based access to localization related data stored in the LOCUS Platform in the persistency entities;
- HTTP REST based access to localization analytics and data as produced by the LOCUS localization, analytics and ML functions;
- Message bus base access to localization analytics and data as produced by the LOCUS localization, analytics and ML functions.

Therefore, the APIs provide two means through which to access and consume the localization analytics and data generated within the LOCUS Platform:

- HTTP REST services, with traditional Create, Read, Update, Delete (CRUD) operations exposed toward the LOCUS applications;
- Message Bus services, to subscribe and consume streams of localization analytics data.

However, to maintain control on authorization and authentication of LOCUS applications accessing the LOCUS Platform, each of the service request/ response transaction occurring through the APIs is asynchronous and split into three separate phases:

- *Service request*: a LOCUS application issues a service request through a HTTP REST operation, requesting for the given localization analytics or data.
- *Synchronous service response*: a service consumption endpoint is provided by the LOCUS Platform, with two options: (i) an HTTP REST service endpoint for the LOCUS application to access and retrieve the required localization analytics information, or (ii) a message bus endpoint and topic to subscribe and start consuming continuous streams of data.
- *Asynchronous service consumption*: the LOCUS application accesses the service consumption endpoint provided in the synchronous service response and consume localization analytics or data service requested.

2.7 Management, Orchestration and Virtualization Functions

LOCUS provides a unified platform for the deployment of localization analytics functions and services on top of 5G virtualized infrastructures, aiming at leveraging functions virtualization for reuse of common localization and analytics functions in several applications and use case scopes. This enables a flexible collection, manipulation, analysis and exposure of localization related data.

As detailed in deliverable D4.3 [16], this translates into two main technical aspects and functionalities for the LOCUS platform. First, a virtualization platform provides an abstraction of edge and core clouds in the 5G infrastructure, while exposing a unified virtualized infrastructure for the deployment and execution of the various LOCUS analytics and localization functions as virtualized functions. Second, a Management and Orchestration (MANO) framework provides full automation in the provisioning of localization analytics services, including deployment, configuration and operation of LOCUS virtual functions on top of the virtualization platform.

The following subsections provide a description of the main functions related to the virtualization platform (in terms of infrastructure virtualization) and the LOCUS MANO (in terms of virtual functions and services orchestration, virtual functions and services repository). These are mapped to the architectural system blocks defined in deliverable D4.3.

2.7.1 Infrastructure Virtualization

As said, the LOCUS virtualization platform provides the means to run the localization analytics services as combination and interconnection of virtual functions. Therefore, it leverages existing virtualization technologies and offers hybrid environment to support distributed deployments in line with the high degree of infrastructure virtualization required by 5G. It integrates Infrastructure as a Service and cloud-native virtualization functionalities to realize a distributed edge/ core platform where the localization and analytics virtual functions can be executed as either containerized or Virtual Machine applications.

In summary, the following tables describe the main infrastructure virtualization functions required to be supported by the LOCUS virtualization platform to fulfil the requirements and capabilities detailed in deliverable D4.3 [16], namely: virtual resource management, container management and virtual resource monitoring.

Table 2.51 Function: Virtual Resource Management

Function: Virtual Resource Management
Input: • Amount of virtual compute resources to be created, updated, deleted • Amount of virtual storage resources to be created, updated, deleted • Characteristics of virtual network resources (e.g. virtual interfaces, ports, bridges, routers, etc.) to be created, updated, deleted • If needed, requirements for interconnecting virtual network resources with containerized virtual functions
Description: The Virtual Resource Management function allows to create, update (if needed) and delete virtual compute, virtual network and virtual storage resources on top of the virtualized infrastructure to properly execute the LOCUS localization and analytics functions as Virtual Machines.
Output: • Virtual Machines are created, updated or deleted in the virtualized infrastructure with the characteristics expressed in the inputs

Table 2.52 Function: Container Management

Function: Container Management
Input: • Reference to the container image or running instance to be used for the virtual function to be created, updated, deleted • Characteristics of virtual network (e.g., virtual interfaces, ports) to be created, updated, deleted • If needed, requirements for interconnecting virtual network resources with Virtual Machine-based virtual functions
Description: The Container Management function allows to create, update (if needed) and delete containers (or PODs) in the virtualized infrastructure to properly execute the LOCUS localization and analytics functions as containerized functions.
Output: • Containers and PODs are created, updated or deleted in the virtualized infrastructure with the characteristics expressed in the inputs

Table 2.53 Function: Virtual Resource Monitoring

Function: Virtual Resource Monitoring
Input: • Subscription to virtual compute, virtual network, virtual storage and container resource monitoring information
Description: The LOCUS virtualization platform allows the LOCUS MANO to consume monitoring information concerning the use and status of the virtualized resources in use to execute the LOCUS localization and analytics virtual functions. This translates into monitoring metrics that can be retrieved and consumed to implement smart management functionalities, such as virtual function for service optimization and fault management.
Output: • Exposure of virtual compute, virtual network, virtual storage and container resource related metrics and status

2.7.2 Virtual Functions and Services Orchestration

LOCUS embeds dedicated management and orchestration functions to flexibly deploy and operate the localization analytics virtual functions and services following the ETSI NFV principles. This allows to reach full automation in the lifecycle management of the LOCUS functions and services, modelled as VNFs and NFV Network Services, on top of the distributed edge/ core virtualization platform. In addition, the consumption of virtual resource monitoring information exposed by the virtualization platform (as described above) allows to implement smart NFV functions (such as service optimization and fault management). This way, the LOCUS MANO becomes a key enabler for the alignment of the LOCUS platform with current trends in managing and orchestrating virtualized infrastructures in 5G and beyond. Indeed, 5G network management and orchestration requires a high level of automation and integration with NFV principles to dynamically and flexibly operate 5G services for verticals.

The following tables describe the main virtual functions ad service orchestration functionalities required to be supported by the LOCUS MANO to address the requirements and required capabilities presented in deliverable D4.3 [16], namely: resource orchestration, service orchestration service optimization, fault management.

Table 2.54 Function: Service Orchestration

Function: Service Orchestration
Input: • Request to coordinate the execution of a lifecycle management action (e.g. instantiation, runtime update, scaling, termination) for a NFV Network Service representing a LOCUS localization analytics service
Description: The Service Orchestration function coordinates a set of lifecycle management steps for deploying in the virtualization platform the individual localization and analytics functions modelled as VNFs in the given Network Service. It is the overarching service coordination entity in the LOCUS MANO, and it leverages the Resource Orchestration function to actually manage the various VNFs. It allows service modifications to be requested by the Smart NFV Functions for automated service optimization and fault management.
Output: • NFV Network Service lifecycle management action automatically executed through proper interaction with the Resource Orchestration function

Table 2.55 Function: Resource Orchestration

Function: Resource Orchestration
Input: • Request to coordinate the execution of a lifecycle management action (e.g. instantiation, runtime update, scaling, termination) for a VNF representing a LOCUS localization analytics function and part of a NFV Network Service
Description: The Resource Orchestration function takes care to allocate, configure and maintain the virtualized resources required to run and operate the individual LOCUS functions VNFs that compose a given localization analytics service. Therefore, it directly interfaces with the virtualization platform and the virtual resource management functions.
Output: • VNF lifecycle management action automatically executed through proper interaction with the virtual resource management functions in the virtualization platform

Table 2.56 Function: Service Optimization

Function: Service Optimization
Input: • Virtual resource (performance) monitoring information collected and consumed from the virtualization platform
Description: The Service Optimization function takes care to automatically adjust NFV Network Services at runtime, with the aim of optimizing the service topology, size and deployment (in terms of which VNF runs where). This is done considering the actual or predicted performance of the service itself, given the consumption and load of virtualized resources used for the various VNFs and NFV Network Services.
Output: • Request to the Service Orchestration function to trigger an NFV Network Service automated scaling (e.g. to add or remove entire VNF instances) or automated service update (e.g. to change the deployment flavour of VNF instances)

Table 2.57 Function: Fault management

Function: Fault Management
Input: • Virtual resource (status) monitoring information collected and consumed from the virtualization platform
Description: The Fault Management Smart NFV function provides the means to automatically manage failure conditions that occurs either in the virtualized platform or in the VNF instances themselves (e.g. at application level). This is done considering the actual or predicted status of VNFs, to detect and automatically react to faulty conditions affecting the virtual resources they use, as well as VNF application reachability or manageability.
Output: • Request to the Service Orchestration function to trigger lifecycle management operations (e.g. scale, update) to react to detected (or anticipate predicted) faulty conditions

2.7.3 Virtual Functions and Services Repository

All of the functions described above for the LOCUS virtualization and MANO require a proper modelling and abstraction of the localization analytics virtual functions to ease and enable automation in their management. This includes various aspects, from the packaging of the LOCUS functions applications into virtualization-ready software images, to their modelling as VNFs through MANO standard descriptors (including NFV Network Services). This allows to specify their requirements in terms of VNF virtualized resources (computing, storage, networking) and NFV Network Service topologies.

As comprehensively described in deliverable D4.3 [16], LOCUS relies on the ETSI NFV standard information and data models for VNF and NFV Network Service Descriptors, based on SOL006 specifications [17], as well as for packages according to SOL004 [18] and SOL007 [19] specifications. These, in turn, as descriptive templates of VNF ad Network Service requirements, need to refer to localization and analytics functions software images and packages (i.e. container or Virtual Machine images according to how they are implemented). For each of them, dedicated catalogue and repository functions are considered in LOCUS to assist the virtualization platform and MANO procedures.

The following tables describe the main functionalities offered by the virtual functions and services repositories, as needed to address the LOCUS MANO and virtualization platform requirements presented in deliverable D4.3 [16] in terms of functions packaging and modelling, namely: VNF and NFV Network Service Catalogue, Container Image Repository, Virtual Machine Image Repository.

Table 2.58 Function: VNF and NFV Network Service Catalogue

Function: VNF and NFV Network Service Catalogue
Input: • Onboard operation: Request to store a new VNF Descriptor or Network Service Descriptor • Query operation: request to query information of a VNF Descriptor or Network Service Descriptor
Description: The VNF and NFV Network Service Catalogue is the centralized MANO repository that hosts the description of the VNFs and Network Services, in terms of requirements of virtual resources and service topology for their instantiation in the virtualization platform. It represents the database of the available localization and analytics functions and services, modelled following the ETSI NFV standard data models. This catalogue is maintained by the Service Orchestration function described above, in terms of access and content verification.

Output: • Onboard operation: VNF Descriptors and Network Service Descriptors are stored and available in the catalogue • Query operation: information VNF Descriptors and Network Service Descriptors is provided

Table 2.59 Container image repository

Function: Container image repository
Input: • Onboard operation: request to store a new container image • Query operation: request to query information of a container image
Description: The container image repository allows to maintain the container images of LOCUS localization and analytics virtual functions implemented and deployed as containers. This repository, in the form of a container registry, is accessible by the virtualization platform to create new container and PODs instances, as well as from the LOCUS MANO functions for linking VNF Descriptors and packages to existing container images.
Output: • Onboard operation: The container imagine is stored and available in the container registry • Query operation: information about the container image is provided (e.g., size, format, checksum, etc.)

Table 2.60 Virtual Machine image repository

Function: Virtual Machine image repository
Input: • Onboard operation: request to store a new Virtual Machine image • Query operation: request to query information of a Virtual Machine image
Description: The Virtual Machine image repository allows to maintain the software images of LOCUS localization and analytics virtual functions implemented and deployed as Virtual Machines. This repository has to be accessible by the virtualization platform to create new related virtual functions instances from the proper software images, as well as from the LOCUS MANO functions for linking VNF Descriptors and packages to existing software images.

Output:

- Onboard operation: The Virtual Machine imagine is stored and available in the repository
- Query operation: information about the Virtual Machine image is provided (e.g., size, format, checksum, etc.)

2.8 Security & Privacy Functions

This subsection presents the LOCUS architecture functions related to the security and privacy aspect. As presented in Figure 2.1, the LOCUS architecture includes a functional block specifically dedicated to location security and privacy functions. In the designed architecture, security and privacy requirements have strong dependencies with other LOCUS functions. Location security and privacy functions are applied to different parts of the LOCUS architecture through the specific functions interface, enabled by the LOCUS APIs. The main properties required for location base service in 5G networks can be summarized as follows:

- Security Requirements
    - **Authentication**: is one of the building blocks of security measures and it protects private and confidential user information. Authentication is the process of identifying entities (i.e. users) accurately.
    - **Confidentiality**: is necessary to protect messages contents from passive eavesdroppers. Confidentiality guarantees the delivery of messages to designated recipients or authorized parties.
    - **Integrity**: Integrity can ensure that the data cannot be tampered by intended or un-intended interference during the data delivery in communication networks, ultimately providing the accurate data for authorized users.
- Privacy Requirements
    - **Location Privacy**: the exact location information of the user must be protected from unauthorized entities. The user's trajectory which contains location data of the user's present and past locations including near POIs must not be revealed to unauthorized entities.
    - **Unlinkability**: is the process related to the user tracking protection that must be implemented to protect the users from linkability of two or more successive positions. In mobile networks, the Service Provider should be unable to link two or more successive positions of the user.
    - **Anonymity**: this means that the subject may perform an action without disclosing its user identity to 3rd parties.

The main goal for the LOCUS Platform definition is to separate security and privacy requirements from the location-based application functions. This allows the LOCUS Platform functional blocks to focus on specific localization features using a common approach to retrieve security and privacy functions. In order to achieve this goal, LOCUS supports all the security requirements related to mobile devices, network, and 3rd party entities. In this context, all transactions between these three actors go through the functions interface. The interface is in fact acting as a connection between the Security & Privacy Layer, and the appropriate APIs exposing functionalities, and 3rd party applications for network management and other vertical applications that shall exploit localization analytics as a service by the LOCUS Platform. The main advantage of this approach is that the security and privacy functions have the ability to monitor and modify all the content, maintain different policies for the device and evaluate possible attacks. The Location Security & Privacy Functions shall ensure all data privacy and security aspects through the LOCUS APIs which handle data acquisition and sharing among the system blocks.

2.8.1 Security Functions

The following tables detail the security functions supported and provided by the LOCUS Platform.

Table 2.61 Function: Authentication

Function: Authentication
Input: • User profile metadata (e.g. username, password hash) • External metadata related to venue/area maps and buildings/shops • Authentication type
Description: This function performs the user/device authentication and authorization process. It implements advanced cryptographic techniques for all network interfaces (homomorphic encryption, secure multiparty computation, and secure conditional sharing techniques), it will be in line with Lawful Interception Architecture defined in 3GPP [20].
Output: • User privilege and permission • User defined policy

Table 2.62 Function: Security Data Clustering

Function: Security Data Clustering

Input:
• Multi-dimensional numerical dataset
  ○ Location measurements provided by LOCUS location enablers
  ○ Location measurements provided by 5G network
  ○ I/Q samples by 5G network (raw data)
  ○ Data from other sensors
• Selected clustering algorithm
• Selected hyperparameters with respect to the Selected Algorithm

Description: This function implements ML models for Security purposes (e.g. Anomaly Detection). Specifically, it uses clustering algorithms for anomaly detection. It works in the original domain or in a transformed domain of the received waveform (Wavelet Transform, Wigner-Ville transform, etc.). The latter may lead to an increase of the separability between the useful signal component features/parameters and those of the interference signal. The candidate algorithms shall process data in order to detect possible anomalies by clustering data in either the original or a transformed domain and to somehow mitigate the erroneous measurements.
Output: • Ongoing attack flag • Measurement labels • Mitigated measurements

Table 2.63 Function: Security Data Cleaning

Function: Security Data Cleaning

Input:
• Multi-dimensional numerical dataset
  ○ Location measurements provided by LOCUS location enablers
  ○ Location measurements provided by 5G network
  ○ I/Q samples by 5G network (raw data)
  ○ Data from other sensors
• Selected clustering algorithm
• Selected hyperparameters with respect to the Selected Algorithm

Description: The LOCUS Platform needs to ensure that the provided location estimates are reliable. This function monitoring estimation degradation can be effective to detect the presence of a malicious attack. Specifically, the goal of this function is to use summary statistics (e.g. mean, standard deviation, kurtosis, skewness) to feed algorithms (e.g. Kalman filters and particle filters) and detect malicious attacks. This function is aimed at the detection and (possibly) mitigation of malicious attacks. Specifically, the algorithms contained in this function shall face with both (noise) jamming and spoofing/meaconing attacks. The exploited techniques shall rely on the tools provided by statistical signal processing ranging from compressed sensing approach, tracking algorithms up to machine learning strategies.
Output: • Ongoing Attack Flag • Measurement labels • Mitigated Measurements

2.8.2 Privacy Functions

The following tables detail the privacy functions supported and provided by the LOCUS Platform.

Table 2.64 Function: Sanitization

Function: Sanitization
Input: • Continuous stream of historical geolocation data as generated from localization functions
Description: This function implements the process of removing user sensitive information from stored location data, so that the data may be distributed to a 3rd party entity. Sanitization attempts to reduce the data classification level. The goal of this function is also related to data anonymization. According to the user/device policy, the function can also apply encrypting on location data.
Output: • Continuous stream of geolocation data as it results from the pre-processing function

Table 2.65 Function: k-anonymity

Function: k-anonymity
Input: • Dataset of historical geolocation data, also sanitized (e.g. from LOCUS persistence entity) • Coordinates of areas under investigation
Description: This function implements data process to produce that any individual entity in the database is indistinguishable, with respect to the quasi-identifiers, from k - 1 other individuals. There are various methods to achieve k-anonymity: generalization of an attribute (for example postal addresses can be generalized to the street or to the city, depending on how much we need to generalize), suppression of an attribute, or addition of dummy records.
Output: • Dataset of historical geolocation data as it results from the pre-processing function

Table 2.66 Function: Obfuscation

Function: Obfuscation
Input: • Dataset of historical geolocation data, also sanitized (e.g. from LOCUS persistence entity) • Coordinates of areas under investigation
Description: This function implements techniques that aim at blurring or perturbing the location information contained in LOCUS persistence entity because of its potential sensitivity. As an example, the precision of location information can be decreased by translating precise point coordinates to geographic regions; Analogously, the precision of the temporal information usually associated with location can be decreased by converting precise timestamps into time intervals.
Output: • Dataset of historical geolocation data as it results from the pre-processing function

Table 2.67 Function: Policy definition

Function: Policy definition
Input: • Set list of restriction specifications that list location data fields that must have K unique values in the result set and a percent level of obfuscation • Set list of operation restriction specifications that deny access to a list of user location data fields and attributes except via the list of utility functions
Description: In order to manage different levels of privacy and security, the LOCUS Platform provides the possibility to define user/device policy. This function implements the setting of the privacy policy. It describes a fine-grained sanitization policy developed for private data, to facilitate their release to public, and a sanitization tool that applies the policy. This policy works in the following way: i) query result restriction specifications that list fields that must have K unique values in the result set; ii) operation restriction specifications that deny access to a list of fields and variables except via the list of utility functions. This function has implications on filtering, aggregation, anonymization, and obfuscation procedures.
Output: • Policy definition result

Table 2.68 Function: Result Aggregation

Function: Result Aggregation (query)
Input: • Query description to be applied on the unstructured dataset's index • Persistence Entity context
Description: LOCUS Platform protects data location through a secure query framework which only releases aggregate and prevalent results based on work done by [21] and [22]. Thus, the privacy of a user/device is further protected by using a "hiding in a crowd" approach 3rd party can query on any features which interest them, but they only receive aggregate responses (counts, histograms, etc.) to address data query correlations. These responses are synthesized from user/device individual responses, after applying their query policies to ensure.

Output:

- Structured Dataset as it resulted from the query operation

3 LOCUS Platform System Architecture

LOCUS provides a platform for location-based analytics that are offered and exposed to Smart Network Management and New Services/3rd party applications and makes use of 3GPP technologies combined with analytics and Machine learning/ Artificial Intelligence (ML/ AI) techniques. This chapter aims at providing an updated, final view of the system's architectural blocks that will contain all the functional requirements (specified in previous chapter, chapter 2), as well as their interfacing mechanisms and the selected technologies for such implementations. It will also differentiate between system blocks that are pre-existing (in open-source implementations) and those that are novel and/or specific to the LOCUS Platform.

3.1 Architecture Principles

The LOCUS Platform's responsibilities is to provide the optimum framework for the successful implementation and execution of the LEN, SNM and NSE use cases and function as described within the LOCUS project deliverables and mainly in [1],[2],[23]. The diverse requirements of all these use cases require the State of The Art (SoTA) software platform technologies. These technologies and the relevant design principles are validated in commercial and research projects worldwide in multiple domains and have therefore formed a so-called de-facto standard.

3.1.1 Software Containerization

Software containerization is a modern technology that allows isolation of a software system in its own dedicated runtime environment. The complexity of some system dependencies in installation requirements (various file system changes, environmental variable, registry, and network changes) may cause serious problems in bare metal installations due to multiple conflicts occurring for co-existing systems. Therefore, the software containers are an ideal method to solve this issue while simultaneously transforming 3rd party system dependencies into simple commands or meta-data instead of requiring actual bare metal installation of supporting system blocks (like databases, REST services and other environment entities).

3.1.2 Microservice Architecture

Monolithic, microservice and hybrid deployments are various design principles for systems that are large scale remote service providers. Based on the work presented in D2.4 [2], the use of a monolithic architectural approach in the LOCUS Platform would generate a lack of flexibility in the development and onboarding of its diverse internal function ecosystem (including privacy/security, LEN, SNM and NSE standalone systems). Ad-hoc generation,

instantiation and de-activation of services is one of the core requirements of the LOCUS platform, however while it can be implemented via the monolithic approach, it would require a very demanding system with multiple resources that would not utilize the virtualization techniques that are already part of this platform. The system microservices are also complemented by the existence of the API access gateway system block which acts as a microservice aggregator, enforcing authorization and user access control as well as combining all the underlying services in a single endpoint for ease of access.

3.1.3 AI-aware Design

It is expected by the LOCUS Platform system to incorporate Analytics Services that will require a set of elementary ML model operations that can be performed for various AI tasks such as classification, regression, clustering, etc. Therefore, model lifecycle management, model training and performance optimization and embedded hyper-parameter tuning capabilities with flexible infrastructure (virtual or actual) resources should be built into the system. The developers of the various LOCUS analytics functions will be able to build prototypes of their algorithms in native, local environments and also deploy them in the LOCUS Platform in the form of analytics service blocks to see the actual performance improvements from the platform AI-related services.

3.1.4 Computation Optimization Abstraction

The LOCUS analytics functions act on large amounts of batch and streaming data. However, the adaptation of best practices for these operations can be greatly abstracted by the UC developer using modern libraries and frameworks. The separation between hardware performance optimization and use case implementation is a method that greatly speeds up the development of analytics services and AI applications. This can be applied to either data processing/ pre-processing tasks (in memory batch processing) and/ or ML- specific tasks (like hyperparameter tuning and model fitting), but also at the prediction phase (where the trained ML models are computing the outcome of new data).

3.1.5 Automated Dependency Resolution and Linking

The LOCUS analytics functions aim at re-using core building blocks to compose higher level services of advanced, localization-based analytics. This means that every high-level service must inform the system (via meta-data) about its internal dependencies and allow for an orchestrator entity to be able to resolve all these service dependencies. This will also allow for better resource utilization as the orchestrator entity will be able to re-use already active analytics services (serving other requests of the same analytic) making the computation blocks of the LOCUS Platform efficient and intelligent. An example of how this approach can be

applied is the reuse of the data collection function, e.g. specifically from the 3GPP network sources, and a localization enabler for a specific area/ venue (e.g. the fingerprinting model) as input for two different NSE applications, trajectory detection and path identification service. Dependency linking must also be an inherit capability of the platform, services can be either directly linked via their intra-service interfaces and also, they can communicate via data exchange in the Persistence and Message Queue system blocks.

3.1.6 Mixed Kappa and Lambda Data Lake Approach

The amount of data that is expected to be part of the processing layer of the LOCUS Platform is expected to be very high in volume and velocity. In addition to this, most of the LOCUS functions will require high performance read, write and transformation operations on these data in order to enable the higher-level analytics that will be served on the system's API gateway. During the use case design phase, there have also been identified requirements for low latency and streaming data processing via message buses and highly available data queues. In order to mix the requirements for both streaming, Pub-Sub and batch/mini-batch processing, we are using a two-sources-of-truth approach which can also be found in literature as mixed Kappa [12] and Lambda [11] data architecture. To ensure the interoperability between the two approaches, appropriate system blocks exist that convert flat SQL persistence into messages for the message queue and vice-versa. Therefore, simple configuration can allow the system to switch from batch to stream processing and to be able to reuse data sources for multiple high-level analytics services and use cases.

3.1.7 End-to-End Low Latency communication

In order to satisfy the low latency requirements that have been defined in real-time analytics use cases of the LOCUS Platform, the involved ingestion, computation and information exchange blocks must communicate via high-throughput low-latency streaming channels. These specifications must be taken into consideration for the selected technologies in: a) the data ingestion blocks (including the cleansing/ anonymization phase); b) the internal message exchange queue; c) the data processing framework (e.g. stream processors); and, d) the analytics delivery (API) mechanisms, such as push notifications (http) external message queue producers. With these capabilities, LOCUS applications can be developed entirely in a streaming context minimizing the end-to-end delay of the analytics service delivery.

3.1.8 Processing and API Access Decoupling

Analytics systems that directly translate analytics requests into heavy computing loads are proven to lead to unstable operation and extreme resource usage scenarios. In order to avoid this, we have designed a specific system block pipeline that will propagate the analytics service

request to an appropriate entity, namely the Data Operations Controller, which will then orchestrate the execution of each task ensuring resource availability, execution order/ schedule and will deliver the outcomes of the analytics results from a separate API, interfacing the data repository designated to this service. The data delivery API will include contextual information about the availability of the analytics results related to the user access request (e.g. ready, pending and so on).

3.1.9 Dynamic Resource Allocation for Computation

During the operation of the LOCUS platform, the demand for analytics results from the various 3rd party application users can have high variation profiles based on the type of service and application it feeds. Intelligent matching of resource requirements and analytics demand allows for a dynamic increase/ decrease of allocated resources which is also leveraged by the virtualization layer and the dynamic service discovery engines the system incorporates.

3.1.10 Security and Services Decoupling

The adaptation of the microservice architecture is introducing a complexity on the enforcement of security, both for encryption and access control. Therefore, the system is designed to have dedicated system blocks for the enforcement of the security requirements with respect to the external access. The internal services will then be easier to develop and maintain since their security features will be limited (allowing for the developers to focus on the actual service). Internal microservices will use the internal SSO interface to replicate the user information and utilize it internally according to the implementation

3.2 System Block Analysis

In this section, various system block categories that are related to the LOCUS Platform are analysed. Each category is essentially a grouping of several internal system blocks that perform a specific set of functions, differentiating itself from the rest of the blocks. It must be noted that each block of the system block analysis is a separate system that exist within its own container on top of the virtualization layer. The separation between blocks has been made according to best practices for such systems to ensure minimum overhead and maximum flexibility. A special category of blocks is dedicated to the external environment of the LOCUS Platform (coloured in grey). These blocks indicate the ecosystem/ stakeholders of LOCUS Platform which directly interface with its “surface” system blocks to perform action on various flow or scenarios (will be covered in next chapter).

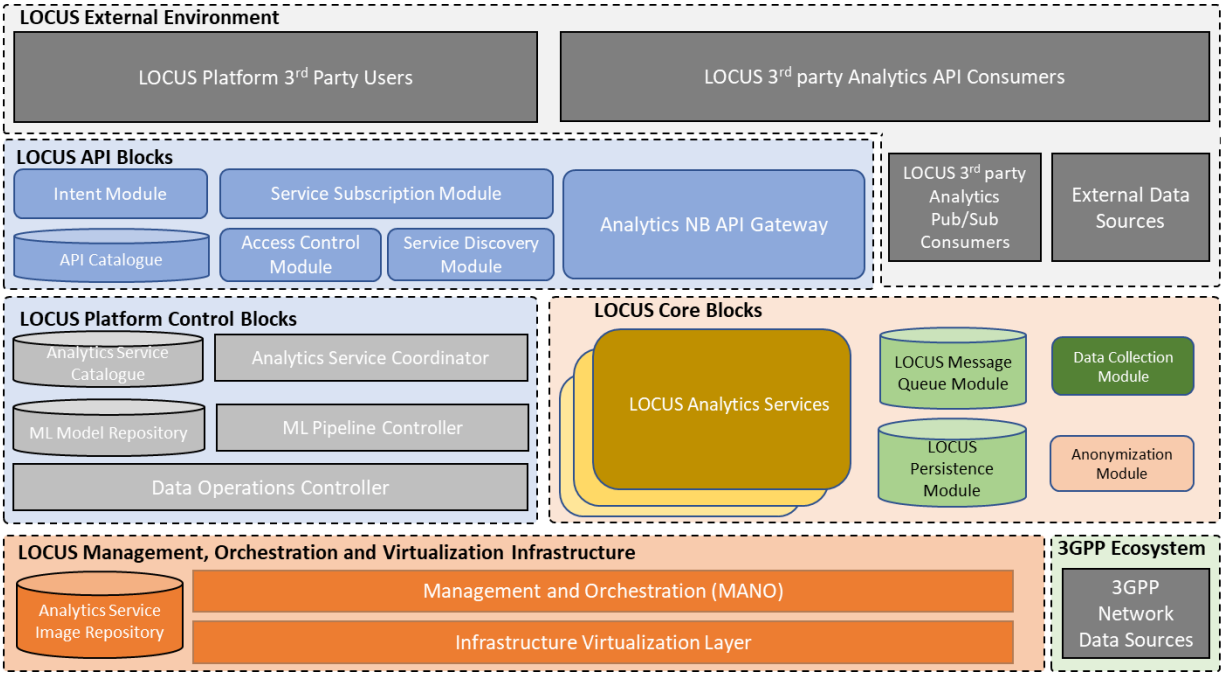

Figure 3.1 LOCUS Platform system block categories and their respective system blocks

3.2.1 3GPP Ecosystem Blocks

The LOCUS Platform is designed to be a system that is either tightly integrated or part of the 3GPP infrastructure. From a southbound perspective, the LEN and SNM use cases that have been designed require access to network information that is either in the form of static batch file exports or -in some cases- direct data feeds from network measurement jobs that occur in the various network management entities systems of the vendor. From a northbound perspective, the outputs of the LOCUS analytics functions can also be an internal service of the 3GPP ecosystem, assisting in the operations of the network, as well new services that are introduced with the 5th Generation release including support for IoT, self-driving cars, smart cities, etc. Therefore, the 3GPP ecosystem (and its various exposable blocks) is expected to be an available resource for the LOCUS Platform.

3.2.2 External Environment Blocks

With the term “External Environment”, we are referring to all the external systems and stakeholders that interact with the LOCUS Platform either actively or passively. This platform is designed to resemble a cloud API portal, i.e. there is a system for the subscription of services, and a different system for the provisioning of this system. In this sense, there is a separation of use case scenarios for the subscription and consumption of the actual analytics. In addition to the consumers of the LOCUS Platform, in this section we have also added the internet-based external resources that can be a useful input for ingestion and inclusion inside some analytics function via the data collection module.

Table 3.1 External Environment Blocks

Block	Description
LOCUS Platform 3rd Party Users	This block refers to all the users that are accessing the LOCUS Platform user management and administration APIs. These users can be normal users, trying to activate their access into the platform by browsing/ finding the appropriate API for their needs and triggering the mechanism for the analytics instantiation or admin users that are monitoring the operation of the system, accepting/ rejecting API access requests and performing general operational analytics.
LOCUS Platform 3rd Party Service Consumers (Applications/ Users)	Any application that is a direct consumer of the LOCUS Platform REST API gateway is included into this functional block. This system (or user in development mode) will be accessing an internal LOCUS analytics function, passing through the gateway reverse proxy system, the authentication and access control mechanism all the way to the internal analytics function and its respective persistence zone (or queue).
LOCUS Platform 3rd Party Pub/ Sub Consumer (Applications)	In the specifications of the LOCUS Platform, under conditions that allow the data exchange of mixed sensitive and non-sensitive information and via VPN access, an application can subscribe directly to the LOCUS Message queue to consume the generated analytics information with the highest available rate and lowest delay. Therefore, this block represents the access to the system for the use of the aforementioned type of analytics.
External Data Sources	This block refers to any number of resources available to the system via internet access. These resources can be file repositories, databases, or other message queues -if appropriate networking and security measures are applicable. These sources have a designated access point to the LOCUS platform via instantiations of the data collection function, which is related to the data collection module.

3.2.3 API Blocks

The API blocks are the primary interface point of external users to the LOCUS Platform. They can be sub-categorized into the Platform API Access blocks (including all the systems that handle user interaction, profile, activation/ deactivation of API capabilities, enable/ disable security authentication/ authorization) and the Core API Analytics blocks (including the API gateway and the various security blocks) that control all the 3rd party user access into the platform. The API Blocks can also be referred to as “REST API modules” due to the capability

of either being deployed as standalone system blocks or part of a multi-module system block architecture.

Table 3.2 API Blocks

Block	Description
Service Subscription Module	This system block is dedicated to the user plane operations of the LOCUS Platform. It provides information about the platform analytics services in the form of meta-data included in the API catalogue. It allows for user registration with the available analytics services and implicitly triggers all the necessary preparation and background processing that will be required from a requested service. It is directly linked with the access control module, which dictates the permissions of each platform user for the system. It is also linked internally with the service discovery module in order to propagate the readiness of a freshly instantiated analytics function.
API Catalogue	The listing of all the available LOCUS service capabilities is an important source of metadata for any user accessing the LOCUS Platform. It is directly connected with the service subscription module and other blocks that share this information. It is also necessary during the onboarding operation of a new LOCUS service that this meta-data is updated with the relevant information. Special details are also used in the content of the meta-data, allowing for intelligent services (such as the intent-based discovery module) to automatically map high-level requirements into the existing API functionalities.
Intent-based Service Mapping Module	The Intent-based Service Mapping Module is an external assistant for the mapping between high-level user requirements and internal Analytics Services. It helps to bridge the API catalogue with a more human-friendly interface, such as natural language in programmatic or a web interface. This system ultimately interfaces with the Service Subscription Module in order to activate a specific identified service to the requesting 3rd party user.
Access Control Module	The LOCUS Platform security requirements indicate that the access to the various analytics functions must be allowed according to their access rights. This block provided user generation, administration and association with such rights and will also enforce the credential verification or API key-based access on all the internal, sensitive resources.
Service Discovery Module	The LOCUS internal ecosystem has the flexibility to dynamically activate/ de-activate the various service capabilities based on demand. One of the engines that supports this is the internal Service Discovery

	Module, which acts as the link between the external API and the status of the internal services. It is responsible for keeping track of the internal systems and it communicates with the Service Subscription Module to provide information to the user about the status of his/ her requested API access.
Analytics NB API Gateway	The system block that is exposed to the consumer of the various LOCUS 3rd party services is based on the API gateway for microservices principle. It uses the reverse-proxy communication to link the external LOCUS environment with the analytics results that lie in the core of the LOCUS Platform. This block enforces user access control (via its communication with the Access Control Module) and also it acts as a "combined" system of all the available analytics resources of the platform. Additionally, the gateway system includes load balancing capabilities, as well as https-based (Transport layer Security - TLS) transport protocols to ensure standard internet security.

3.2.4 Platform Control Blocks

The internal LOCUS Platform blocks are crucial for the smooth operation of the LOCUS machinery that supports the complex analytics functions described in the various use cases. The analytics functions can differ greatly with respect to their lifecycles, based on their dependencies on other analytics functions, on external data source ingestion or built-in specific ML operations, such as training, testing, re-training, etc. In addition, the LOCUS Platform is a system that links API access requests with virtualization technologies, making it an on-demand Software as a Service (SaaS) system. Dedicated blocks are also related to this operation, performing the replication/ instantiation of the requested resources and also re-parametrization of batch computations to serve multiple simultaneous requests.

Table 3.3 Platform Control Blocks

Block	Description
Analytics Service Coordinator	This system block is responsible for receiving instantiation requests from a higher layer of the LOCUS Platform and transforming them into appropriate signalling for the virtualization platform, resulting in the resource allocation and start-up of all the necessary software for any analytics service. It is directly linked with the Analytics Service Catalogue, which allows for access to the dependency and execution requirement descriptors of each Analytics Service.
Analytics Service Catalogue	This system block is a catalogue for all the LOCUS Platform analytics service descriptors, which specify the service capabilities and its execution requirements (including reference to virtual resource

	requirements descriptors). The analytics service coordinator requires access to this information in order to trigger the instantiation of the analytics service and its related functions as virtual service and virtual functions (e.g., containers) by the Management, Orchestration and Virtualization Infrastructure blocks, and then to orchestrate the service execution and runtime configuration.
Data Operation Controller	The Data Operations Controller is a high-level orchestrator and scheduler of executions for the various LOCUS Analytics Services. It receives configuration from the higher levels (API Access) or internal triggers to perform computations that are required for the result of the analytics services. It communicates with the existing analytics services using a control interface of lightweight, non-blocking signalling and it also works closely with the Analytics Service Coordinator, in order to request more replication of the same analytics service to serve incoming load. In general, the Data Operations Controller is a system block that fits the Lambda data lake architecture [11], where a central dispatcher is orchestrating the high data volume operations that manipulate data from the various location of the lake.
ML Model Repository	The LOCUS Platform is an AI-ready infrastructure that encapsulates many of the functional requirements of AI applications. One very important aspect of these is the ML model continuous training, evaluation and re-evaluation cycle. Analytics services that utilize ML models can leverage this LOCUS block to reduce their required customized implementation of such operations and focus on the actual utilization of the predictive result as generated analytics information.
ML Pipeline Controller	Another system block of the platform that is strongly related to AI is the ML pipeline controller. It exists in the form of a programmatic API layer between the LOCUS Analytics Function and the LOCUS MANO and virtualized infrastructure that tries to enforce embedded hyper-parameter tuning and containerized pre-processing pipelines with minimum developer interaction.

3.2.5 Management, Orchestration and Virtualization Infrastructure Blocks

The whole set of localization analytics services and functions that the LOCUS Platform supports are deployed and executed into a distributed virtualized environment, with the aim of enhancing flexibility and agility in how localization and analytics data is collected, processed, manipulated and exposed to service consumers. To reach this, on the one hand the LOCUS platform embeds an infrastructure virtualization layer that abstracts edge and core computing locations to provide a common virtualization platform where virtual functions and

services can be deployed. On the other hand, dedicated management and orchestration system blocks allow to model localization analytics services and functions as virtual services and functions in compliance with the ETSI NFV principles, providing the required coordination logic to instantiate and operate them in full automation. These blocks are briefly summarized in the table below (Table 3.4).

Table 3.4 Management, Orchestration and Virtualization Infrastructure Blocks

Block	Description
Management and Orchestration (MANO)	The MANO is the core coordination engine for the deployment, execution and operation of the various localization analytics services and functions on top of the virtualized infrastructure. It provides to the LOCUS platform automation capabilities in the management and orchestration of virtual resources to dynamically deploy and operate the required virtual functions and services to fulfil the analytics service requirements as expressed and issued by the platform control bocks (primarily the Analytics Service Coordinator). It follows the ETSI NFV principles and supports automated virtual functions and services instantiation, scaling, runtime update, monitoring and termination.
Infrastructure Virtualization Layer	The LOCUS Platform is a system designed to utilize virtual resources for static and dynamic system blocks. It primarily uses container-based virtualization, but it also includes blocks that for the sake of performance prefer a bare-metal allocation of Virtual Machines (VMs). The Infrastructure Virtualization layer provides a common and unified access to virtual resources (network, compute, storage) in the distributed edge/core infrastructure, as required to run the various localization analytics services and functions, as well as the LOCUS platform control and management blocks and functions, which in turn can run as virtualized services. For this, proper isolation mechanisms are required.
Analytics Service Image Repository	An important system block of the LOCUS MANO infrastructure is a dedicated repository for the storage, administration, version control and access of the container images that can be deployed by the system. It is accessed by the LOCUS Platform contributors during the onboarding phase of a new LOCUS analytics function and it provides an interface to the container instantiation engine to download the images during the operation phase of the system.

3.2.6 *Persistence and Message Queue Blocks*

The system blocks dedicated to data persistence, data movement and messaging lie at the core of the LOCUS Platform. These blocks are using a dual stack of streaming and tabular storage in order to fit all the functional requirements discussed in previous chapters. The tabular and streaming data are also linked together via change data capture and data sink system blocks which allow for interoperability between the two data lake modes.

Table 3.5 Persistence and Message Queue System Blocks

Block	Description
Persistence Module	The LOCUS Persistence Module is a multi-table, multi-schema environment for structured (tabular) and unstructured data. It includes mechanisms for raw data transformations into structured (via designated write interfaces) and big data query engines for fast read operations, as well as direct analytic execution via aggregation functions, window functions and predicate pushback for filtering and selection. It is important to note that this block will not be utilizing a containerized deployment, instead it will be deployed on top of bare metal VMs, in order to maximize the IO speed (containers use raw socket network interfaces for IO which has relatively slower performance). The scale-out capabilities, however, of this system should not be lost therefore the selected technologies make it easy to increase/ decrease nodes (even if we are talking about replicas virtual machines or Primary/ Secondary nodes).
Message Queue Module	Message queues are systems that are dedicated to pub/sub data access and process intercommunication via message exchange. The LOCUS Platform will serve many use cases where streaming data manipulation both internally (inter-locus-function) and externally (with 3rd party streaming data consumers that have very low latency tolerance) is necessary. Therefore, this system block must feature multi-topic, multi-message-format, multiple network protocols and support temporary persistence capabilities (i.e. message retention periods), but also high read (processing) capabilities by the means of replication on the access layer.
Interoperability Modules	The two different data approaches of the LOCUS data architecture design require two corresponding interoperability modules that will translate the persistence (in the form of SQL-type tables) into messages and topics (for the message queue) and vice-versa. Therefore, this section is dedicated to the analysis of these interfacing system blocks. Different LOCUS use cases may require parametrization of these systems in order to support dual streaming and flat file stack. These

	two systems are: a) Persistence change data capture module: This system is responsible for the communication between persistence landing zones and message queue. It translates insertion/ update/ deletion events (specified by a table query) into message events to be added in a designated Change Data Capture (CDC) topic B) Message queue sink into persistence module: This system is dedicated to the conversion of messages (received at a specific topic of the message queue) into insertions (update or deletions). In order to do so, the messages should follow a specific payload format that can be easily converted into SQL insert/ upsert/ update/ delete events.

3.2.7 Core Blocks

The LOCUS Core system blocks are the summary of analytics functions that have been analysed in the functional requirements of the system and that have been instantiated in the same, homogeneous way as internal platform, REST-based services. They are all discovered by the service discovery mechanism and linked into the API Gateway internally via appropriate reverse http proxy mechanism to the end user.

Table 3.6 Core System Blocks

Block	Description
LEN Service Blocks	The LEN service blocks include all the LOCUS analytics functions that have been designed in the activities of WP3 that provide the core input for the rest of the analytics functions. They are functions to convert 3GPP and non-3GPP based input data (derived from execution of the data collection module) into localization information for mobile nodes inside specified venues, areas and network coverage zones. The LEN service blocks differentiate themselves from the rest of the analytics functions as they are not directly provided by the API gateway to external users. Instead, they generate location information (in a specific agreed output schema) as inputs for the rest of the LOCUS Analytics Functions (LEN, SNM).
SNM Service Blocks	SNM service blocks include all the LOCUS analytics function service blocks that are dedicated to services for network management. They are functions that implement optimization, analytics, and ML-generated results to assist network management and operations for the 3GPP and non-3GPP networks. These functions generate results that are either directly or indirectly linked to external 3rd party users via the message queue blocks or the API gateway. They are important building blocks to develop applications that are heat-map web applications for network KPI maps, and other geographical

	representations of network KPIs that are closely related to the analytics that are already part of the operational infrastructure for mobile network operators (OSS).
NSE Service Blocks	NSE service blocks are instantiation of LOCUS analytics functions for serving localization-based analytics for non-network related high level consumer and use cases. These cases can vary from customer retail to transportation analytics and also cases for security/ disaster prevention, smart city/ smart venue analytics. They are also direct consumers of the outputs of LEN functions that are in turn converted it into more useful, higher-level analytics and correlated with other non-network related information that can be accessible via data collection function from the external data sources of the platform. The applications that consume these services access them via the API Gateway or via appropriate topics and can be either connection between backend systems or appropriate front-end applications that can directly visualize the correlated geospatial and metric information (related to the vertical applications).
Data Collection Module	One of the most important system blocks for the operation of the Core analytics function are their data collection phase. To perform this operation, the LOCUS Platform must have a dedicated system, capable of multiple executions for communicating with the various external sources and saving the information in the various system persistence and/or message mechanisms. The Data Collection module can work in a continuous (based on time triggered scheduled operations) or on demand manner (based on a request/ response mechanism) acquiring data from internal (3GPP) and internet sources and using (optionally) the anonymization functions (provided by other blocks) to alter the information and to ensure anonymization of data.
Anonymization Module	The anonymization module is a dedicated system that provides services of data manipulation for the abstraction of private/ sensitive information. It can be accessed via two different interfaces, during the data collection phase (it is strongly linked with the ingestion processed for standardized privacy ISO requirements) and during the ingestion of external information from sensitive sources that require de-personalization of data before its persistence in either the Message Queue or the Persistence module.

3.3 Function Assignment to System Blocks

In this section, a detailed mapping of each of the LOCUS Platform functions defined in the previous sections and its corresponding system block and group that it has been assigned to

is presented (see Table 3.7). LEN, SNM and NSE functions are omitted as they are indisputably assigned to the LOCUS Core system blocks with the same label, i.e. LEN, SNM and NSE service blocks respectively.

Table 3.7 Function catalogue with corresponding system group and block

Function Name	System Block Group	System Block
Data Collection 3GPP / External	Data Collection	Data Collection
Unstructured Data Persistence	Persistence and Message Queue	Data Persistence
Structured/Relational Data Persistence	Persistence and Message Queue	Data Persistence
Unstructured Data Query	Persistence and Message Queue	Data Persistence
Structured Data Query	Persistence and Message Queue	Data Persistence
Structured Dataset Transformation to Unstructured or Semi-Structured	Persistence and Message Queue	Data Persistence
Unstructured Dataset Transformation to Structured	Persistence and Message Queue	Data Persistence
Unstructured Dataset Transformation to Structured	Persistence and Message Queue	Data Persistence
Message Queue Topic Generate / Destroy	Persistence and Message Queue	Data Message Queue
Internal Message Queue Produce / Consume Data	Persistence and Message Queue	Data Message Queue
3rd Party Message Queue Data Consume	Persistence and Message Queue	Data Message Queue
Persistent data change data capture function into messages	Persistence and Message Queue	Persistence / Message Interoperability
Message Broker generic sink into landing table	Persistence and Message Queue	Persistence / Message Interoperability
Analytics Function Service Instantiation / Despawn	Platform Control	Analytics Service Coordinator
Analytics Service Configuration Processing	Platform Control	Data Operation Controller
Analytics Service Health Check	Platform Control	Data Operation Controller
Analytics Service Discovery Function	Analytics API	Service Discovery
User Access Control Manipulation	Analytics API	Access Control
Analytics Service Metadata manipulation (onboarding/ edit)	Analytics API	API Catalogue
Analytics service subscription	Analytics API	Service Subscription Module
Intent-based analytics services subscription	Analytics API	Intent Module

Analytics 3rd party Consumer API Gateway	Analytics API	Analytics NB API Gateway
Authentication	Security & Privacy	Authentication / Access Control
Security Data Clustering	Security & Privacy	Anonymization module
Security Data Cleaning	Security & Privacy	Anonymization module
Sanitization	Security & Privacy	Anonymization module
k-anonymity	Security & Privacy	Anonymization module
Obfuscation	Security & Privacy	Anonymization module
Policy definition	Security & Privacy	Authentication / Access Control
Result Aggregation (query)	Security & Privacy	Anonymization module
Virtual Resource Management	Management, Orchestration and Virtualization Infrastructure	Infrastructure Virtualization Layer
Container Management	Management, Orchestration and Virtualization Infrastructure	Infrastructure Virtualization Layer
Virtual Resource Monitoring	Management, Orchestration and Virtualization Infrastructure	Infrastructure Virtualization Layer / Management and Orchestration (MANO)
Service Orchestration	Management, Orchestration and Virtualization Infrastructure	Management and Orchestration (MANO)
Resource Orchestration	Management, Orchestration and Virtualization Infrastructure	Management and Orchestration (MANO)
Service Optimization	Management, Orchestration and Virtualization Infrastructure	Management and Orchestration (MANO)
Fault management	Management, Orchestration and Virtualization Infrastructure	Management and Orchestration (MANO)
VNF and NFV Network Service Catalogue	Management, Orchestration and Virtualization Infrastructure	Management and Orchestration (MANO)
Container image repository	Management, Orchestration and Virtualization Infrastructure	Infrastructure Virtualization Layer
Virtual Machine image repository	Management, Orchestration and Virtualization Infrastructure	Infrastructure Virtualization Layer

3.4 System Interface Analysis

In this section, the interactions between each of the system blocks -which are grouped together into different sections according to the data or operation “flow” they serve- are analysed. Each interface has more than one system block involved, can be single or bi-directional and must also specify the information exchange (data structure, fields, format, etc.), as well as the exchange mechanism itself (e.g. http, programmatic, file system).

3.4.1 System Flow Grouping

A good approach to the analysis of the system block interfaces is a high-level grouping based on the four elementary flows that exist within the LOCUS Platform. These flows are mentioned in the table below (Table 3.8) and are presented graphically in Figure 3.2:

Table 3.8 High level grouping of platform flows

Flow type	Description
Southbound Flow	Includes all the interactions between system blocks that involve the initial ingestion of input data into the LOCUS Platform, anonymization, pre-processing and ingestion on the various persistence and messaging systems, as well as their final delivery as inputs in the LOCUS analytics functions.
Northbound Flow	Includes all the interactions that occur from a 3rd party API (or queue) consumer leading downwards to the execution of analytics, reading from saved analytics results through the gateway, as well as security and discovery mechanisms.
Internal LOCUS Analytics Flows	Includes all the interfaces that are active during the runtime of a LOCUS analytics function. The capabilities of such function are all based on the requirements that derive from the various UC that have been included in the LOCUS Platform design and vary with respect to processing, persistence, networking and signalling capabilities. Special attention is also given to the embedded, ML-related interfaces (between the analytics functions and the various performance optimization blocks for elementary mathematical operations). Such interfaces in general are of programmatic nature.
LOCUS Platform Access Flow	In this flow, all the "signalling" interactions between system blocks that are relevant to the activation of an analytics service from a user are included. This action triggers a set of events that extend to the virtualization layer in order to ensure that the function is properly instantiated and active.

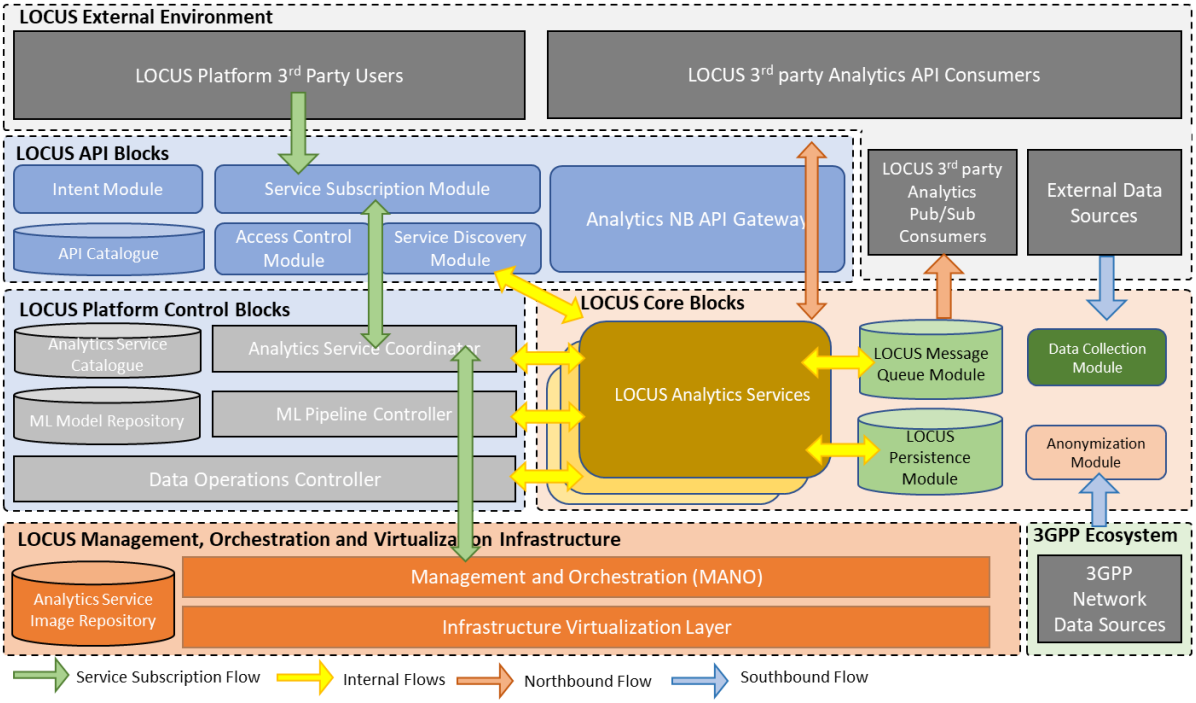

Figure 3.2 High level system block diagram indicating data flows

3.4.2 Southbound Flow Interface Analysis

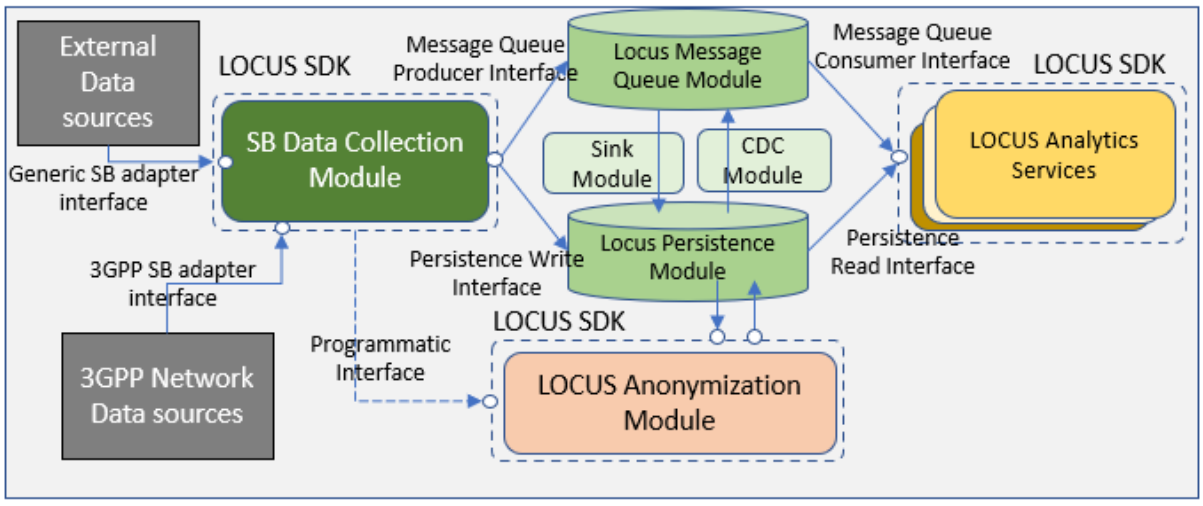

Figure 3.3 Southbound flow interfaces – Schematic representation

The Southbound flow of the LOCUS Platform includes all the interactions between system blocks at the edge of the platform interfacing with external (or 3GPP ecosystem) data sources and performing pre-processing, anonymization, fusion or persistence of the data, until they become the input for the processing of the core analytics functions (LEN, SNM, etc.).

Table 3.9 Southbound flow interfaces

Interface name	Involved Blocks(s)	Interface type	Interface Mechanism	Interface message details
Generic southbound adapter interface	External data sources, SB Data Collection Module	HTTP, SFTP7, SSH8, Message Broker	Polling, Pub-Sub	Variable structured payloads containing measurements in JSON, JSON-table-schema, XML, or CSV format
3GPP southbound adapter interface	3GPP network data sources, SB Data Collection Module	SOAP9, HTTP	Polling	Location Data (Viavi Arieso Format [24]), Network NME measurement data (CSV format), XML format
Anonymization programmatic interface (library)	SB Data Collection Module, Anonymization Module	API (library)	Processing	Tabular data inputs and outputs with obfuscated columns based on the appropriate anonymization targets (memory arrays)
Message queue producer/ consumer interface	SB Data Collection Module or Anonymization Module, LOCUS Message Queue	TCP, UDP10, other application protocol on top of TCP	Duplex (Ingoing, Outgoing)	Messages containing data in JSON-table-schema format (self-contained SQL information) or other JSON format based on use case- specific schema
Persistence read/ write interface	SB Data Collection Module,	JDBC TCP protocol according to	Duplex (Ingoing, Outgoing) &	TCP payloads of SQL result set to be persisted in the

7 Secure File Transfer Protocol

8 Secure Shell

9 Simple Object Access Protocol

10 User Datagram Protocol

	Anonymization Module, Persistence Module	the implementation of the database	acknowledgement	destination schema/ table
Message queue sink module interface	Sink Module, LOCUS Message Queue, Persistence Module	TCP, UDP	Pub-Sub	JSON messages translated into insert/ update/ delete statements on target persistence tables
Persistence table change-data-capture interface	CDC module, Persistence Module, LOCUS Message Queue	JDBC TCP protocol according to the implementation of the database	Pub-Sub	JSON messages translated into insert/ update/ delete statements on target persistence tables

3.4.3 Northbound Flow Interface Analysis

The northbound platform flow includes all the relevant interfaces that serve the final delivery of analytics outcomes (saved or real-time generated) to the 3rd party application users of the system in either http (REST) access or direct consumption of designated message topics.

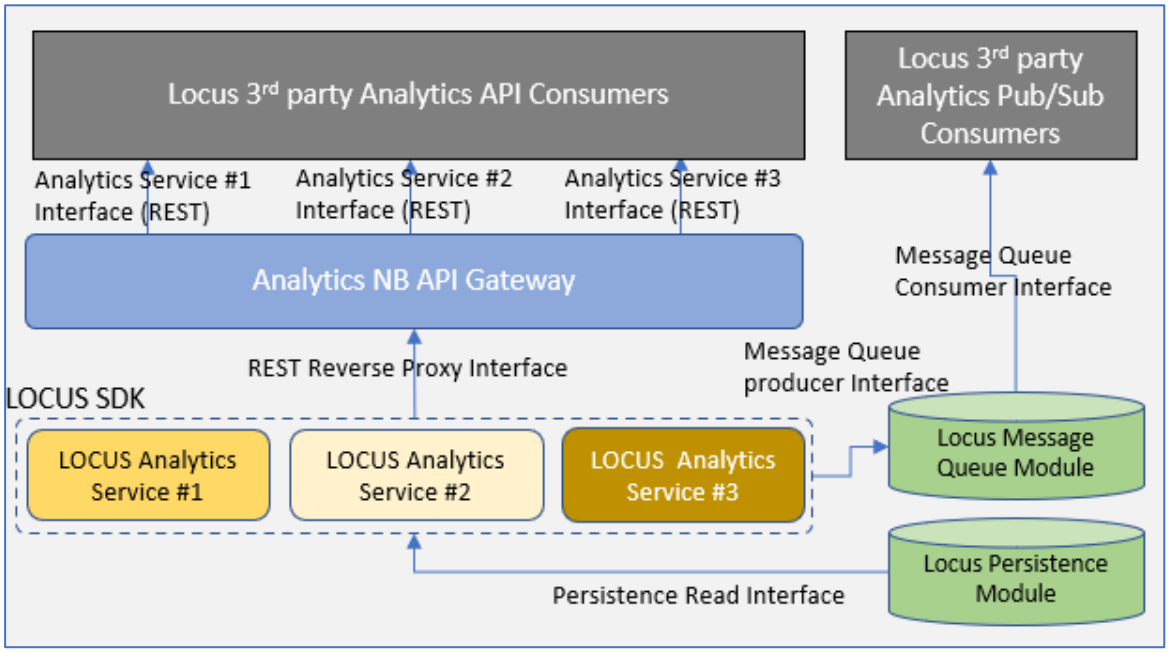

Figure 3.4 Northbound flow interfaces – Schematic representation

Table 3.10 Northbound flow interfaces

Interface name	Involved Block(s)	Interface type	Interface Mechanism	Interface message details
Persistence read	Persistence Module, LOCUS Analytics Service #N (LOCUS SDK)	JDBC TCP protocol according to the implementation of the database	Outgoing	TCP payloads of SQL result set to be processed internally in the analytics function and redirected upwards
Message queue producer interface	LOCUS Analytics Service #N (LOCUS SDK), LOCUS Message Queue	TCP, UDP, other application protocol on top of TCP	Outgoing	Messages containing data in JSON, XML, CSV, Avro [25] or Protobuf [26]
Message queue Consumer interface	LOCUS Message Queue, LOCUS 3rd party Consumer Application	TCP, UDP, other application protocol on top of TCP	Outgoing	Messages containing data in JSON, XML, CSV, Avro or Protobuf
HTTP (REST) Reverse Proxy Interface	LOCUS Analytics Service #N (LOCUS SDK), LOCUS API Gateway	HTTP	Duplex	HTTP request/ response data that differentiate according to each analytics function's API design
LOCUS Analytics Service #N REST Interface (via Gateway)	LOCUS API Gateway, LOCUS 3rd party API Consumer	HTTPS/TLSv3	Duplex	HTTP request/ response data that differentiate according to each analytics function's API design

3.4.4 API Activation and Subscription flow Interface Analysis

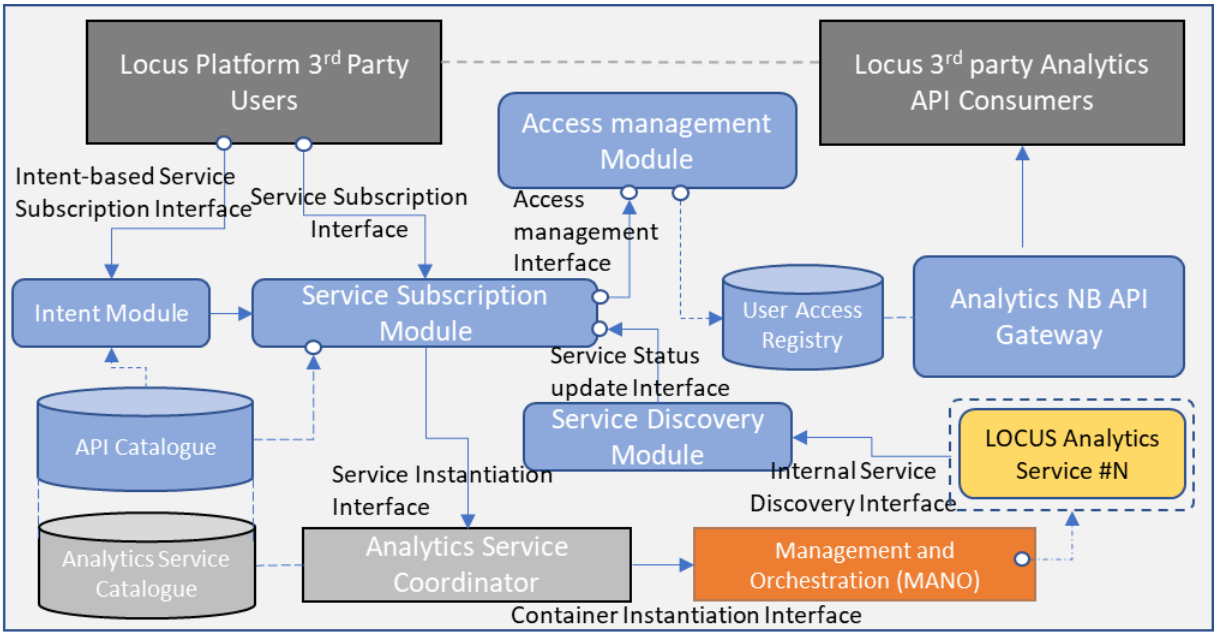

Figure 3.5 API activation and subscription flow interfaces – Schematic representation

This set of interfaces are related to intercommunicating systems that are part of the analytics service activation lifecycle. They start from an initial API access request at the top level of the platform access layer and it triggers all the required operations up to the virtualization layer. It then generates a feedback flow that is meant to notify the end user that the service is available and accessible (also if it requires an initial training to change its status to active).

Table 3.11 API activation and subscription flow interfaces

Interface name	Involved Blocks(s)	Interface type	Interface Mechanism	Interface message details
Service Subscription Interface	LOCUS Platform 3rd Party Users, Service Subscription Module	HTTPS/TLSv3	Duplex	Appropriate combination/ exchange of http post/ get/ delete statements triggered by the platform front-end application or directly invoked in an automated manner aiming at changing the status of the active API subscriptions for the specified users
Intent-based Service Subscription Interface	LOCUS Platform 3rd Party Users, Intent Service	HTTPS/TLSv3	Duplex	Appropriate combination/ exchange of http post/ get/ delete statements triggered by the platform front-end application or directly invoked in an automated manner aiming at discovery of an existing locus

	Subscription Module			analytics service that matches a natural text description
Service Instantiation Interface	Service Subscription Module, Analytics Service Coordination	HTTP	Duplex (request & acknowledgement)	An internal http call that aims at triggering the container instantiation and replication for the specified analytics function, linking between the API access layer and the platform operations layer
Container Instantiation Interface	Analytics Service Coordination, Container Controller	Raw socket	Duplex	Internal system call for the instantiation of a specified container based on an existing image file
Internal Service Discovery Interface	LOCUS analytics service #N (LOCUS SDK), Service Discovery Module	HTTP	Outgoing (+acknowledgement)	Signalling between any locus analytics function (triggered after the instantiation and linking with the container mountpoint) that designates the specific function resource to be active in specific network IP/port
Service Status Update Interface	Service Subscription Module, Service Discovery Module	HTTP	Polling	Interface between the Service Discovery Module and the Service Subscription Module in order to trigger access capabilities for the specific user/ service pair
Access management Interface	Service Subscription Module, Access Management Module	Programmatic/ HTTP (based on implementation)	Duplex	Communication between those two blocks ensures that an underlying internal analytics service instance is ready for a specific user and it unlocks the reverse proxy API gateway access

3.4.5 *Internal API Flow Interface Analysis*

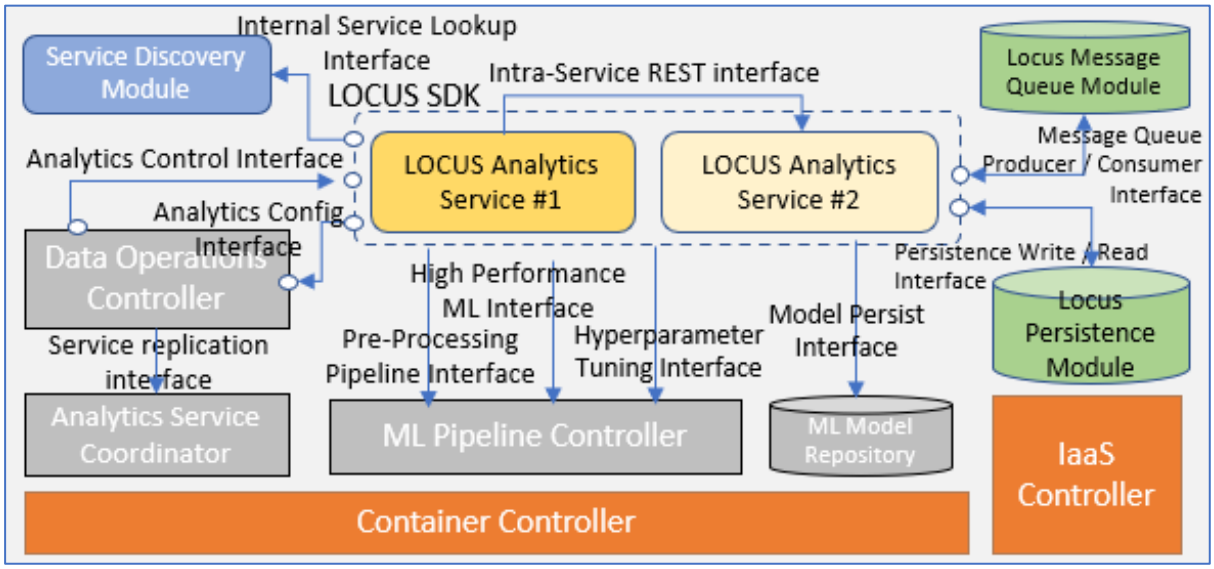

Figure 3.6 Internal API flow interfaces – Schematic representation

In the centre of the LOCUS Platform lies a large set of interfaces among inter-working blocks that act as the core toolchain for the development of robust, containerized, high-performance analytics services. These interfaces vary from programmatic to light-weight network (signalling) to high-throughput network interfaces (TCP/ JDBC) in order to use the best possible transport mode for every different communication case. All these interfaces will also be available for any implementor of a LOCUS analytics function in the form of an application SDK (LOCUS SDK) which will be analysed in further chapters.

Table 3.12 Internal API flow interfaces

Interface name	Involved Blocks(s)	Interface type	Interface Mechanism	Interface message details
Message Queue Producer/ Consumer Interface	LOCUS Analytics Function #N, LOCUS Message Queue	TCP, UDP, other application protocol on top of TCP	Duplex (Ingoing, Outgoing)	Primary interface for the exchange of messages containing data in JSON, XML, CSV, Avro or Protobuf
Persistence Read/ Write Interface	LOCUS Analytics Function #N, Persistence Module	JDBC TCP protocol according to the implementation of the database	Duplex (Ingoing, Outgoing) & acknowledgement	Primary interface for the exchange of SQL data via TCP payloads of SQL result set to be persisted or read in the destination schema/ table. Note: The read and write operation can be separate system blocks according to technology selection

High Performance ML Interface	LOCUS Analytics Function #N, ML Pipeline Controller	Programmatic Interface (raw socket, file system)	Duplex (read/write)	Analytics implementation that requires access to optimized ML code (linked with low level C++ binaries) is provided via a direct programmatic interface for seamless code integration (e.g. via python bindings)
Pre-processing Pipeline Interface	LOCUS Analytics Function #N, ML Pipeline Controller	Programmatic Interface (raw socket, file system) or Http	Duplex (read/ write)	Analytics implementation that requires access to optimized pre-processing code (maybe executed in a high availability dedicated container resources pool) is provided via a direct programmatic interface for seamless code integration (e.g. via python bindings) or via a remote block invocation (http call)
Hyper-parameter Tuning Interface	LOCUS Analytics Function #N, ML Pipeline Controller	Programmatic Interface (raw socket, file system)	Duplex	Analytics functions that utilize internal optimization loops (including ML fit functions) require a high-performance interface for the abstraction of hyperparameter tuning. Via this interface the low-level ML pipeline API provides the analytics service with the optimally tuned model result (in-memory)
Model Persist Interface	LOCUS Analytics Function #N, ML Model repository	HTTP	Ingoing / Outgoing (& acknowledgement)	Analytics functions that include ML training and especially continuous retraining and health-checking against real time streams of data require direct access to a repository for the current production model. The payload is exchanged in serialized base64 [27] form of multiple model binary types (e.g. python pickle [28])
Analytics Config Interface	LOCUS Analytics Function #N, Data	HTTP	Ingoing (& acknowledgement)	External triggers for various analytics requests need to be translated into pipeline executions for the data operations controller.

	Operations Controller			This interface allows for all the required input information to be transferred to the data operations controller in order to trigger a pipeline or to generate a new one
Analytic Control Interface	LOCUS Analytics Function #N, Data Operations Controller	HTTP	Outgoing (& acknowledgement)	The data operations controller is using REST API calls to understand the status of a task that is executed by the processing containers of each analytics function. The message exchanged are health-checks that take simple string values such as 'processing', 'ready', 'available' encapsulated in JSON format
Intra-Service REST Interface	LOCUS Analytics Function #N, LOCUS Analytics Function #!N	HTTP	Duplex	This REST interface is used for the direct communication between REST blocks of analytics functions. It provides the easiest way of communicating information between functions, but it is specific to its message payload to each different analytics function (i.e. specific to each implementation)
Internal Service Lookup Interface	LOCUS Analytics Function #N, Analytics Service Discovery Module	HTTP	Duplex	This REST interface is used when a LOCUS analytics service (or any other internal LOCUS Platform module) is searching for the network information (IP/ port) of a specific instantiated LOCUS SDK-based functional block (including LEN, SNM, NSE services, also anonymization module). The Analytics Service Discovery Module responds with a JSON object that is pointing to the instance or instances (if multiple) that may serve as appropriate targets to access via the Intra-Service REST Interface

3.5 Data Schema Analysis

In this section, all the schemas that are related to the LOCUS Platform and utilized in interfaces, messages or repository system blocks are analysed. A very important aspect of the design of the data schema is to use open-source reference schemas as inspiration for the design, to consolidate as many LOCUS analytics function data dependencies into a unified schema and to include data formats for the various outcome representations that are included in other research projects.

3.5.1 Open Source / External Interoperable Data Formats

It is an important objective of the LOCUS Platform to utilize as much as possible inter-operable, well-documented, industry-standard data formats to describe the inputs and outputs of its interfaces both internally but also, more importantly, externally - at the interfacing with any external 3rd party application or with the 3GPP-specific interfaces.

3.5.1.1 GeoJSON Format

Figure 3.7 Example taken from 3rd party software documentation of a shape in the GeoJSON format [29]

The GeoJSON [10] format is a very widely used extension of the JSON format that is used to describe various geometries such as data points, lines (sequence of data points) and closed loops of points (polygons). It is also supporting multiple combinations of these primitives in order to be able to describe complex geometries that are found in geographical applications

of the ICT domain. The LOCUS platform is strongly related with the consumption and generation of geographical/ geospatial data therefore a well-known an interoperable format is key for better accessibility of the analytics outcomes. In particularly, WP4 and WP5 developed functions that either consume or produce geometric shapes (e.g. the POI identification or path identification functions, as described in the section 2.5) are a good example of services that generate outputs in the GeoJSON format. In addition, the localization enabler functions use the GeoJSON format to specify filtering boundary (area) for their localization computations. Another positive attribute of the GeoJSON format is that there exists a plethora of SQL implementations that translate GeoJSON format into native SQL functions that perform a number of the LOCUS Analytics Functions (as described in the section 2) making this format a somewhat native input/ output

3.5.1.2 JSON SQL Table Schema Format

The JSON SQL [30] table schema format is a widely used extension of the JSON format that is used to store self-contained SQL data rows. These objects are self-contained SQL rows which means that they include both the payload (the data of the row) and also the SQL schema that is the destination to store them. Even though the data overhead of including the schema as well as the payload information is considerable, it is a very helpful meta-data entry for any 3rd party system that is processing the information and can also be mandatory for some interfaces of the LOCUS Platform, such as the CDC and the Message Queue sink modules (that directly translate to and from persistence modules and message queues).

```
{
  "name": "name of field (e.g. column name)",
  "title": "A nicer human readable label or title for the field",
  "type": "A string specifying the type",
  "format": "A string specifying a format",
  "description": "A description for the field",
  "constraints": {
      # a constraints-descriptor
  }
```

Figure 3.8 Example of the descriptors in the JSON SQL schema format

3.5.1.3 3GPP Location Input Data - MDT Specification

Localization enabler LOCUS Analytics Functions are specialized functions that translate raw network measurement data into location information. In order to be interoperable with the current data specifications of some 3GPP network components, the inputs of the LEN functions can be adapted to the specification provided by TS 32.421 [31] describing trace concepts and requirements for the minimization of drive tests (MDT). These data include fields related to the Multicast-broadcast single-frequency network (MBFSN) such as MBSFN area

identity, Carrier frequency, MBSFN RSRP, MBSFN RSRQ, MCH BLER for signalling, MCH BLER for data, and related MCH index.

3.5.2 Platform Schema Analysis

In this section, we analyse all the Platform-specific repositories and schemas that serve functions from the re-usable global system blocks. These blocks lie in different parts of the platform and are either of the structured nature (i.e. SQL tables), partially structured formats (i.e. SQL tables including one or multiple blocks of binary or JSON payload) or unstructured formats (saved in a file system and accessed via SFTP and/or other binary payload exchange interfaces). In each case, we have selected the data representation that is more appropriate for the requirements of each function. When high compression is required, tabular format is preferred and embedded SQL databases whereas when high performance in the effective serialization/ deserialization IO is required binary formats are selected.

3.5.2.1 API Catalogue Schema

The API Catalogue is a tabular schema that is accessed by the service subscription module and also (via proxy) by the Intent-based Service Discovery mechanism. It includes multiple fields that are related to the existing analytics services' description, usage scenarios, input data, output data, internal sub-functions, listing of API calls and their respective HTTP methods for accessing.

Table 3.13 API Catalogue schema description

Field	Type	Description
Analytics Service ID	LONG	An autogenerated incremental long ID for each distinct analytics service
Analytics Service Name	VARCHAR	The system name of the specified analytics function that is requested
Analytics Service Domain	VARCHAR	A high-level characterization of the domain for the specific analytics service (e.g. Vehicular, Cellular Network, Retail)
Analytics Service Tags	Array(VARCHAR)	Multiple string entries of relevant meta-data tags that are related to this specific analytics service
Analytics Service Internal Info	JSON BLOB	Data structure containing all the available internal REST endpoints, the accepted http methods (POST, GET, DELETE) and

		their respective input and output data types in JSON format
Analytics Service internal Image Url	VARCHAR (foreign key to analytics service function image URL)	A logical foreign key to the internal identifier of the LOCUS Analytics function service that assists in the internal mapping of the high-level API entry with the Analytics Service Coordination entity and the Analytics Service Repository

3.5.2.2 Access Control Schema

This is a schema dedicated for the access control operations of the external (3rd party) users of the LOCUS Platform. Based on the mappings of Table 3.14, 3rd party users of the platform are allowed access to services with endpoints (by proxy based on the meta-data of the analytics API internal info as presented in Table 3.13). Furthermore, there are appropriate considerations for API access limitations (e.g. based on rate limiting mechanisms).

Table 3.14 Access Control schema description

Field	Type	Description
LOCUS Platform User ID	VARCHAR	An autogenerated unique platform user ID that denotes a specific user (company or owner of 3rd party application) accessing some analytics service results via the standardized API gateway route
Analytics Service ID	LONG	A foreign key relation to a specific analytics service function (from the analytics function schema) which denotes that the specific user has access to this analytic service (and all the related internal children routes of the URL tree)
API Access Restriction	JSON BLOB	A description of the access type for the API split between different modes: • Unlimited Access: where a user can access every resource without any limitation being enforced on the requests

		• Fixed #Calls: Where the access to the LOCUS APIs is limited by the #of requests on the API Gateway (according to the numberOfRequests field) • Fixed Duration: Where the access to the LOCUS APIs is limited by the request timestamp (according to the maxDateToAccess Date field)
Rule Timestamp	Date	The date that this rule was generated (for the internal clean-up operation)

3.5.2.3 Analytics Service Catalogue Schema

The Analytics Service Catalogue schema is dedicated to the persistence of the meta-data in the form of descriptors that are required for the operation of each LOCUS analytics service. It is defined that a LOCUS analytics service can consist of multiple dependant system blocks (i.e. functions), each performing different parts of the required actions to execute the analytics, as well as supporting system blocks (based on open-source or proprietary images) that will be required for the execution of their internal tasks. During the onboarding phase of a LOCUS analytics service, the required descriptors are stored along with information about the current version/ release. The same corresponding information is propagated to the API Catalogue schema, as these two must be in sync in order to accurately depict the current production-ready image of the service. The schema is a meta-data schema of links to file system paths based on following structure (Table 3.15).

Table 3.15 Analytics Service Catalogue schema description

Field	Type	Description
Service Name	VARCHAR	An identifier for the LOCUS analytics service
Service Subsystem Names	List (VARCHAR)	An identifier for the sub-systems of the LOCUS analytics service functions
Date Generated	Date	A date/ time indication of when this image has been changed in the image repository

Major/Minor Version	VARCHAR	An indication of the current entry's software version to match with the rest of the systems
Instantiated	LONG	Analytics on number of times this service has been instantiated in the platform (for estimations of load)
Service Execution Requirements	BLOB	A JSON blob specifying the analytics service execution requirements to be processed by the Analytics Service Coordinator, e.g. specific input data sources required linked to other analytics services or functions, edge proximity constraints.
Reference to MANO Descriptor	LONG	Reference identifier that allows to have access to the MANO descriptors that contain the deployment requirements of the analytics service and its functions (CPU required, memory, networking) to pre-allocate resources and to assist in intelligent allocation

3.5.3 Unified Analytics Schema as an Integration Point

In this section, the LOCUS Platform schemas that are related to inputs and outputs of the various analytics functions are presented. These include schemas that are accessed by the privacy/ anonymization functions, inputs for the localization (LEN) functions, outputs of the LEN functions, inputs for the SNM/ NSE functions, as well as their respective outputs. It should be noted that this section is focused on schemas that are interoperable/ integration points for the whole platform. This means that they are not dedicated to a single analytics function, but they are being accessed as input or outputs from various analytics services.

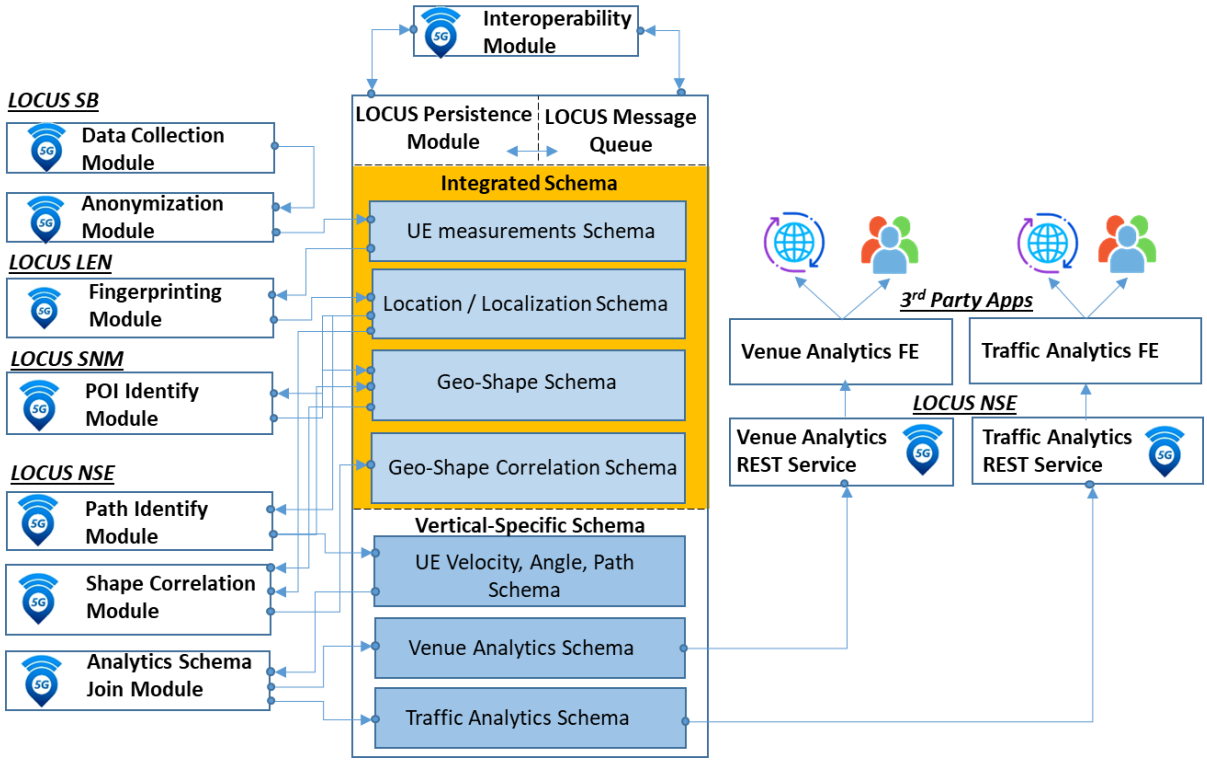

Figure 3.9 The LOCUS unified schema as an integration point

In Figure 3.9, an example of how the unified schema works as an integration point between various LOCUS Platform system blocks of different categories is shown. In the chosen example, an end-to-end pipeline starting from the LOCUS southbound system blocks moving the raw information to the user equipment measurement schema (originating via the 3GPP connection), and then being anonymized by the anonymization function and used by a localization enabler (fingerprinting) to convert into location stored in the location schema. The location schema is one of the core integration points of the LOCUS Platform, as it serves equally as input and output for either the LEN or the SNM/ NSE functions. It is therefore designed to prevent conflict between parallel read/ write operations by specifying appropriate flags and categorical fields. At the same time, it can be accessed by a service with total abstraction of the data source or localization enabler that produced the result based on higher level input such as accuracy, geographical area or time period. The other interoperability schemas are related to geographical/ geometries related or generated by the platform via functions developed on top of the localization information. The example depicts two LOCUS functions (POI identify service and path identify service) which produce shapes (GeoJSON) that can be both stored in the geo-scape schema. Other functions (like the shape correlation function) can be used on shapes generated by multiple functions to provide user-correlation entries in the geo-shape correlation schema which in turn can be accessed by multiple analytics vertical implementations (e.g. the venue analytics or the traffic analytics service in this case).

3.5.3.1 User Equipment Measurement Schema

Inspired by the 3GPP specification of MDT [31], this schema is dedicated to capturing low level (physical layer) measurement information from different mobile phones acting as measuring devices either actively (by using a sampling application) or passively (capturing and translation of 3GPP-related signalling). It encapsulates all the relevant information that can be used by the various localization enabler functions to train or to perform direct computation of the location for a specific user equipment device.

Table 3.16 User Equipment Measurement schema description

Column	Column Type	Description	Example
LOCUS UE ID	String	Identifier for a UE terminal or other move-able object that the platform is able to detect	Mob 33
Timestamp	Long	Timestamp associated with this location measurement	1130314130
Latitude	Double	Latitude of location	31.311
Longitude	Double	Longitude of location	22.131
LOCUS Radio Element ID	String	Identifier of the cell in the 3GPP network (usually ECI, or cellid concat wite site id) or WiFI access point ssid for hybrid use case	1239191 (ECI)
Radio Element Type	String	Identifier to determine the technology type for the measurements (WiFi, 3G, 4G, etc.)	eNodeB
Cell Latitude	Double	Latitude of the specific cell (join from cell meta-data)	31.311
Cell Longitude	Double	Longitude of the specific cell (join from cell meta-data)	22.131
DoA	Double[]	Direction of arrival of the serving cell expressed as a 3d-vector(length =1)	[0.1,-0.3,0.5]
AoA	Double	Angle of arrival after the processing between the location and the cell location	77degrees

TDoA	Long	Microseconds time difference of arrival	30412 μs
RSRP	Double	Reference signal received strength for the mobile terminal in dBm	-77
RSRQ	Double	Reference signal received strength quality (dB)	-10
RSSI	Double	Received signal strength indicator (dBm)	-89

3.5.3.2 Location / Localization Schema

The Location/ Localization schema is the output schema for all the localization enabler functions. It provides location information for mobile nodes inside the network area along with additional meta-data on the location estimation based on the methodology that was used for the localization, the relevant accuracy for the measurement and other information such as shapes that are being correlated with the location measurements (in case of relative location, not based on latitude and longitude) which is something that the indoor localization enablers utilize.

Table 3.17 Location/ Localization schema description

Column	Column Type	Description	Example
LOCUS_UE_ID	String	Identifier for a UE terminal or other move-able object that the platform is able to detect	Mob 33
Timestamp	Long	Timestamp associated with this location measurement	1.13E+09
Latitude	Double	Latitude of location	31.311
Longitude	Double	Longitude of location	22.131
RelativeX	Double	Relative coordinate X of the location	520
RelativeY	Double	Relative coordinate Y of the location	33

RelativeShape	String	Shape which will be used for the relative location. Linked with Geo-shape schema	mcarthur_1
Accuracy Class	String	Low, Medium, High - this will be expanded based on data survey	Low
Source	String	ID of the LOCUS WP3 analytics service that generated this estimate	WP3 DNN Fingerprinting

3.5.3.3 Geo-Shape Schema

The Geo-Shape schema is the LOCUS Platform's schema design for any geospatial/ geographical information that is generated by the various implementations of analytics functions. In general, location and geospatial shapes are very strongly related and -in an SQL context- they interact by means of geospatial correlation functions that are user-defined-function extensions of the SQL interface performing operations such as "area contains point", "area intersects with point", etc. Via this repository we can also propagate shapes as final outcomes of various verticals, such as the POI identification or Path detection use case.

Table 3.18 Geo-Shape schema description

Column	Column Type	Description	Example
ShapeID	String	The identifier (unique) of a shape in the LOCUS platform	POI_313231
parentShapeID	String	If this shape is by any means linked to another shape (e.g. by a LOCUS function that takes a scope as input)	mcarthurglen
Timestamp Generated	Long	Timestamp associated with the generation of this shape in the platform	1.13E+09

Type	String	Described the entry type e.g. Path, POI, BoundingRectangle, Point	POI
Source	String	ID of the LOCUS WP4 and or WP5 analytics service that generated this or whether this was generated by User-definition or SB-interface	WP4_Clustering
GeoJSON	String	The literal GeoJSON of this shape (Feature)	{ "type": "Feature", "properties": {"party": "Republican"}, "geometry": { "type": "Polygon", "coordinates": [[[-104.05, 48.99], [-97.22, 48.98], [-96.58, 45.94], [-104.03, 45.94], [-104.05, 48.99]]] } }

3.5.3.4 Geo-Shape Correlation Schema

The geo-shape correlation Schema is complementary to the location and geo-shape schema. It is used for the persistence of mobile locations in conjunction with existing shapes for a specific time frame and reference area. It is generated as output of the shape /location correlation function which is implemented directly on the persistence layer using convex-hull, graham scan, interpolation and other geometrical functions.

Table 3.19 Geo-Shape Correlation schema description

Column	Column Type	Description	Example
LOCUS_UE_ID	String	Identifier for a UE terminal or other move-able object that the platform is able to detect	Mob 33
Timestamp	Long	Timestamp associated with this location measurement	1.13E+09
ShapeID	String	The identifier (unique) of a shape in the LOCUS platform	POI_313231

3.6 Platform Technology Selection

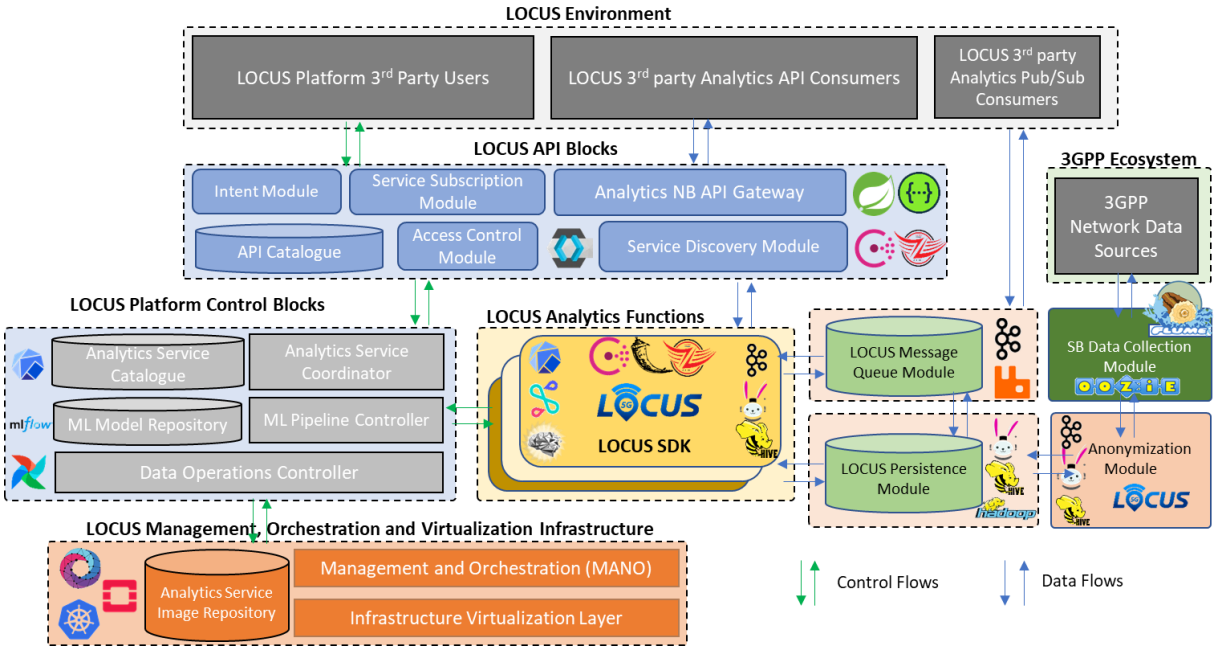

Figure 3.10 LOCUS blocks and selected technologies

This section of the system architecture is dedicated to the detailed search for implementations, systems, de-facto standards and best practices that can be applied to the LOCUS Platform to acquire a finalized instantiation of all its functional capabilities. For this purpose, we have selected a series of candidate technologies for some of the LOCUS platform system blocks, split between technologies that fully perform the platform's functions and technologies that can be used to develop the specialized, LOCUS-specific functions related to the work performed by the LOCUS partners. These technologies are also part of the deliverable D5.3 [23] and have been refined to fit the finalized architecture of this document.

Table 3.20 Platform technologies

LOCUS System Block	Technology	Justification
LOCUS MANO	ETSI OSM [32]	ETSI OSM is an industry and telco operator led open-source initiative that provide a reference implementation of the ETSI NFV MANO architecture. It is basically the de-facto standard solution for managing and orchestrating virtualized network functions (VNFs) and Network Services according to the ETSI NFV specifications. ETSI OSM implements various ETSI NFV standards, including northbound APIs for VNF and Network Service lifecycle management, and VNF and Network Service descriptors data model. In LOCUS, it is used to cover the MANO functionalities and system blocks, as detailed in deliverable D4.3 [16]. As a key feature, ETSI OSM is natively capable to manage hybrid virtualization infrastructures where legacy VNFs and containerized network functions (called KNFs) are deployed on top of Openstack [33] and Kubernetes [34] mixed infrastructures.
Infrastructure Virtualization	Kubernetes [34], Openstack [33]	As detailed in deliverable D4.3 [16], Kubernetes and Openstack are jointly used in LOCUS to realize the hybrid virtualization platform. Openstack and Kubernetes are both de-facto standard virtualized infrastructure management solutions, and their combined use allows to deal with containerized and Virtual Machine based applications. In addition, it enables a flexible and agile deployment of virtual functions and services at both edge (through Kubernetes) and cloud (through both Kubernetes and Openstack) locations. With this hybrid implementation approach, the Infrastructure Virtualization layer also includes repositories for the virtual function software images, being them either Virtual Machines (by using the related Openstack image management service) or containers (by using legacy registry solutions such those offered by Docker [35]).
Analytics Service Coordinator	Custom	The Analytics Service Coordinator is a LOCUS custom system block and is implemented as an entirely new software from scratch. It will be based on lightweight but production-ready microservice frameworks of Python [36] programming language, such as the Flask framework [37], to implement the proper

		coordination logic for the deployment and execution of analytics service pipelines interacting with the LOCUS MANO and the Data Operations Controller.
Analytics Service Catalogue	Custom	The Analytics Service Catalogue is also implemented as a custom software in LOCUS, meaning it is not based on any existing open-source technology. Being it a catalogue for analytics services available in the LOCUS platform, it will provide a database service and will be also on lightweight but production-ready microservice frameworks of Python programming language such as the Flask framework [37] to access and manipulate the catalogue data.
ML Model Repository	ML-Flow [38]	ML-Flow is an open-source system designed to add dev-ops, deployment and production readiness to the world of machine learning. It has capabilities such as model repository, experiment repository and orchestrates version control and meta-data storage to perform automated tasks that stabilize the delivery of updated models in modern production environments. The LOCUS platform is utilizing its model repository functionality, for storage of the trained models that are used in the LOCUS analytics functions. This system is accessed by the client-side library of LOCUS SDK and it is also used to store model performance metrics such as accuracy, retention, recall and other AI-specific KPIs to maintain their performance.
Service Discovery Module	Consul [39]	Consul is an open-source technology for REST service discovery functions, such as service registration and service network address reverse lookup. It provides a centralized repository for microservices to register their IP/port pairs for each of their services that can be accessed by appropriate client libraries to use the network addresses for direct communication within a network (in our case the LOCUS Analytics functions internal network).
Access Control Module	Keycloak - Spring Security [40], [41]	Keycloak is an open-source authentication and identity checking provider following the SSO-architecture. Spring [41] and Flask [37] applications can use this 3rd party system as a single source of authentication (external) and the user plane meta-data are handled by the local database of each Keycloak instance. Spring security is an extension on the Sprint REST framework that allows for implementation of filtering and security rules that is based on the Keycloak rules (acquired via their internal signalling).

Data Collection Module	Apache Oozie [42], Apache Flume [43]	Apache Oozie is an automated task scheduling tool that is built in many big data cloud image providers. It uses a specialized XML-based data structure to define the types of jobs that it will execute and is used in conjunction with scripting languages or other, built-in client binaries. For the data collection module, the most appropriate client-side 3rd party software is the Apache Flume, which is a very widely used tool for the download of external data source raw data and delivery at specified target locations. This is done by configuration files which specify the input and output sources. The templates for both of these systems will be provided by template files existing in the LOCUS SDK.
Data Persistence Module	Apache HDFS [44], Apache Hive [45], Trino [46]	The technologies that will act as the LOCUS Platform's Big Data Lake, the persistence module, are the de-facto standard for multi-VM multi-disk, high availability, multi-node environments. Apache HDFS is the storage layer at the lowest level. It is used in multiple cloud persistence deployments as it provides easy addition/ removal of storage nodes (connected to virtual or actual disks) and it provides high throughput (HTTP-based) read and write operations. Apache Hive is a technology that is built on top of Apache HDFS and is basically an SQL tabular data storage layer. It utilizes Hadoop map-reduce binaries to transform various data sources (that exist within the HDFS) into different file formats that are more optimized for compression or analytics (e.g. Parquet [47] or ORC [48] format). In addition, it gives the user access to an SQL compliant meta-data storage that can allow for other query engines and SQL interfaces to perform SQL-like operations (which are subsequently converted into other types of operation on top of the HDFS nodes). Finally the Trino.IO (formerly known as the PrestoDB project by Facebook) is an open-source distributed query engine that is optimized for multi-machine scaling on top of SQL data storage catalogues such as Apache Hive. By splitting the read and write operations between those two technologies, we take advantage of the speed of Trino and the stability of Hive, MR-based transformation operations both operating on top of the HDFS-based file storage.
Message Queue	Apache Kafka [49],RabbitMQ [50]	For the message queue subsystem of the LOCUS Platform, we have selected a dual stack of Apache Kafka and RabbitMQ. They both provide topic generation, multiple payload type formats such as JSON, XML, CSV, Avro, and binary and they can be accessed easily by consumer/ producer clients, which require

		very low code complexity but are very robust and performant (especially Kafka which adds additional mechanisms for parallel processing of streams and integrity)
Data Operations Controller	Apache Airflow [51]	Apache Airflow is a multi-purpose high level data operations orchestrator for a very wide variety of underlying systems and/or tasks. It is using the principles of DAG (Directed Acyclic Graph) for the design of parallel, serial, dependant and independent jobs to build high level architectures of data pipelines and also various "Operators" (e.g. ShellOperator, HttpOperator, PythonOperator, JavaOperator) to perform tasks (essentially link with the underlying binaries. In the LOCUS Platform, it will play the role of the Data Operations Controller, essentially invoking the large compute tasks of the various LOCUS Analytics functions that read/ write large amounts of data (via their designated SQL/ message interfaces). The sequence of invocation for these jobs is very crucial for their healthy operation, which is why each LOCUS function has to provide its own, respective DAG for executing its internal step-based tasks.
API-Gateway, Service Subscription Module	Spring Boot Microservices [52], Zuul [53], Consul [39], Swagger [54]	Spring Boot is a Java-based web framework for the development of production-ready REST applications. It is a technology that can be used as a public API gateway since it is flexible, scalable, light-weight and includes all the embedded security features that were mentioned in the technology selection for the authentication/ security module. In order to convert this into an API gateway, further extension/ integration libraries of the Spring ecosystem must be utilized, such as the Zuul integration (provides HTTP reverse proxy on internal systems, i.e. the various LOCUS analytics function systems and integration with Consul for the discovery of the network address/ addresses of these services). In addition, the Spring Boot framework provides seamless integration with the Swagger REST interface which generates meta-data from the LOCUS analytics functions towards the end user that are essential for the user manual, input and output data type communication and testing operation of the public API.
Intent-based Service Discovery	Python [36], Flask [37], Fast API [55]	For the implementation of an Intent-based Service Discovery the recommended technologies than can be used are lightweight, production-ready microservice frameworks of python programming language, such as the Flask framework or the Fast API, the latter being essentially a more advanced version of Flask that supports parallel (asynchronous) processing of requests and

		built-in Swagger documentation (same technology as the API gateway).
ML Operations	Kubeflow [56]	Kubeflow is a scalable ML platform that runs on Kubernetes and exploits all its capabilities to facilitate the development, deployment and operations of virtualized ML solutions. It provides means to flexibly pre-process data and train various models in virtualized and containerized environments. Kubeflow is used in LOCUS to take care of the virtualized ML pipeline operations, in a portable and scalable way. A pipeline, in Kubeflow, is a description of a machine learning workflow, including all of the steps in the workflow and how they combine in the form of a graph. A pipeline includes the definition of the inputs required to run the pipeline and the inputs and outputs of each step. It covers various steps in a ML pipeline, from training, to data pre-processing, data transformation and serving. In LOCUS it is used to assist ETSI OSM in specific ML operations on top of the virtualized functions and services. In particular, for the serving part, Seldon [57] can be used to expose the trained models as REST microservices in Kubernetes that can thus be invoked and controlled by the Data Operations Controller.

4 LOCUS Analytics Function SDK

Within the LOCUS project, multiple partners contribute to the development of the LOCUS Core analytics functions as part of the finalized LOCUS Platform's internal features. These functions are developed and experimented on local environments at initial stages, while this is followed by the platform onboarding phase. In this phase the various algorithms, methods, implementations are migrated into a pre-defined template for LOCUS analytics functions that is designed according to the required capabilities of these functions, the system blocks in the LOCUS ecosystem that implement these functions and the predefined interface methods for accessing these resources. This template-based migration mechanism provides enough flexibility to the developers of the various function (LEN, SNM, NSE) developers while increasing the code maintainability, stability and separation of concerns that is required for the fastest possible development of the actual implementations. The LOCUS SDK is split into two main categories of features: a) Adapter libraries that are used for network and programmatic interfacing between each LOCUS analytics function and its environment via standard python-based method invocations and b) Configuration file templates for all the LOCUS environment 3rd party software/ Open Source software components (such as Kubernetes [34], KubeFlow [56], OpenStack [33], Apache Oozie [42], Apache Flume [43], Apache Hive [45], Apache Kafka [49], Apache Airflow [51], etc.), which perform installation and initialization of tasks that are required by the LOCUS functions to operate. The configuration files and the core function implementation are synchronized with the platform's lifecycle according to different phases (e.g. onboarding, clean-up) that will be analysed in the final subsection of this chapter.

4.1 SDK High Level Design

This template project presented in Figure 4.1 is used by all the LOCUS partners to implement their contributions to the LOCUS Platform. The SDK includes libraries and dependencies for interfacing with all the LOCUS Platform internal systems. The main goals for the use of such template are: a) Separation between development and deployment; b) Code re-use for connectors, control plane, lifecycle management; c) Specifications' description and availability of information to other partners within the project. The technology of the LOCUS Platform SDK is related to the selected technologies of the previous section for the main system blocks. Specifically, every service that uses the LOCUS SDK will be an Embedded Microservice (Python3 [36], Flask [37]) with endpoints for control plane, network registration using Consul [39] and instantiation via docker-compose [35]. The supported programming languages for the LOCUS SDK is the widely used in general purpose programming, computations, analytics and machine learning language Python [36]. In general the SDK supports the following function types:

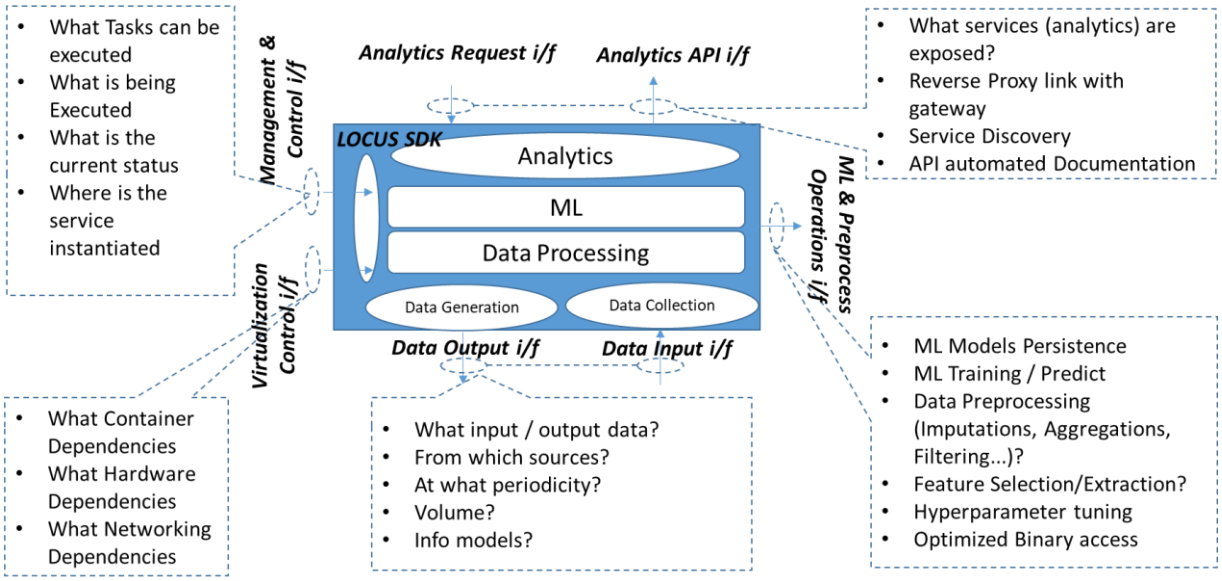

Figure 4.1 SDK high level design template

- Provide specification for the function's dependencies (Ingestion of Data, other LOCUS or 3rd party systems);
- Persistence and Message Data Manipulation (Generate tables / topics, read/ write);
- API Gateway functions (Service Registration, Service Discovery, API Documentation);
- Communication with data operations controller (Execution, Cancelling of Analytics function, Status Monitoring);
- Performance optimization of Analytics, pre-processing and ML Operations based on programmatic, REST and other interfaces (KubeFlow [56], Kubernetes [34]).

4.2 SDK Adapter Library Analysis

The existing (modular) adapter sub-libraries within the LOCUS SDK allow for the application developers to easily interface with the rest of the LOCUS internal environment.

Based on the technology selection for the internal LOCUS Platform system blocks, the LOCUS SDK is required to provide connectivity with:

a) Structured database via different write/read interface (Hive [45]/ Trino [46]) based on python implementation of JDBC converted directly into pandas[58] array;

b) Unstructured storage read/write based on python implementation of the HDFS client library (https) [44] converted into array or object type;

c) Interfacing with the internal message queue (write, read from topics) supporting both message queue technologies Kafka [49] and RabbitMQ [50];

d) Pre-implemented control interfaces (REST, python flask) with the Data Operations Controller (Airflow [51] via Flask control endpoints [37]) for coordination, on demand execution and termination of executed task;

e) Registration to the proxy/ reverse proxy capabilities with the API Gateway (Consul [39], Zuul [53]) for communication with the 3rd party application users;

f) ML model serialization, deserialization and persistence based on ML-Flow REST API that holds additional meta-data for the various executions, model versions, performance KPIs and execution environments required for the embedded models.

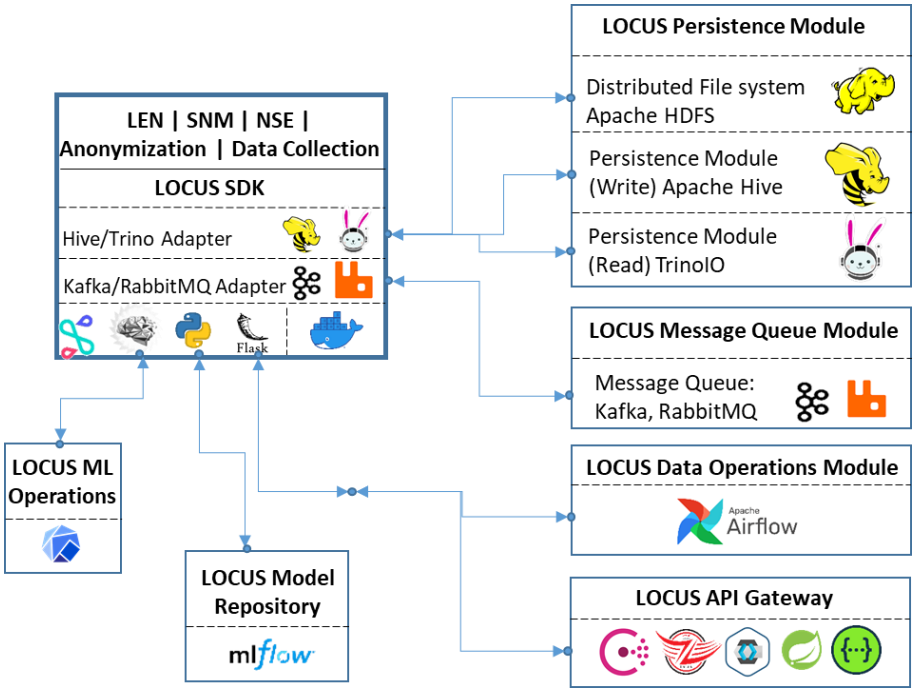

Figure 4.2 LOCUS SDK technology adapters

These capabilities are provided via implemented libraries that can be used on demand based on each use case developer's functional needs. The SDK user will be able to specify the included connectors in a modular way, meaning that all unnecessary functions can be excluded for more lightweight analytics service container images. Each connector provides the SDK user with the API to perform all the necessary functions related to the specific interface, abstracting the actual complexity of the operations and reducing the code duplication among all partners. Finally, the LOCUS SDK will provide pre-compiled docker-compose files for dockerized deployment and Kubernetes integration.

4.3 SDK Configuration File Analysis

Apart from the adapter functions that are provided by the LOCUS SDK, the basic LOCUS Function template contains all the required files (configuration, definition, code) that are being processed by the LOCUS Platform's different system blocks to perform various

initialization operations. These files can be edited by the developers to prepare the environment for the necessary resources and/or functionalities of the specific LOCUS function.

Table 4.1 LOCUS Configuration files

Configuration File	Relevant LOCUS System	Purpose
docker-compose.yml	Container Management & Orchestration (Docker [35], Kubernetes [34])	• CPU/ memory/ networking dependencies • dependencies on internal or external LOCUS functions • Dependencies on other services (3rd party)
DockerFile (txt)	Service Image Registry, Container Management & Orchestration (Docker [35], Kubernetes [34])	Describes commands to the image build subsystem that will allow for the building and exporting of the analytics function's container into the service image registry
catalogue-metadata.json	Analytics Service Catalogue Repository (Spring [41], SQL)	Required meta-data that provide information about the functionalities of this LOCUS analytics function to allow 3rd party users to access it
oozie schedule (.xml)	Data Collection function Module	File describing the chrono scheduling/ triggering of the data collection function. The configuration of this file can either include a self-contained python-based data collection operation or CLI-script or it can be used in conjunction with the apache flume properties file to invoke a pre-determined task
flume-conf. properties.template	Data Collection function Module	Configuration for the Apache Flume's [43] data acquisition engine which is used by the scheduled Apache Oozie [42] jobs. These can be various client libraries, such as SQL (JDBC) , HTTP, SFTP, etc.

consul.json	Service Discovery Module	Contains meta-data that are required for the registration of the LOCUS analytics function into the Consul [39] network address registry (allowing for meta-data-based reverse search)
kafka.json, rabbit.json	LOCUS Messaging Module	Contains all the information for the required Kafka [49] and RabbitMQ [50] topics that will be converted into actual queues using the topic initialization function
hive.json	LOCUS Persistence Module	Contains all the required Hive tables that will be accessed by the LOCUS analytics function, these tables will be created via a 'create if-not-exist' statement. It includes information such as the table type (based on the Hive data types, like CSV, Parquet, ORC) and, for each data column, its respective data type (e.g. String, Double, DateTime)
dag.py	Data Operations Controller (Apache Airflow [51])	Contains the Python-based definition of the Airflow DAG which will be executed to generate the LOCUS analytics-function-specific high level pipeline. After its installation in the Airflow system, the relevant graph (of the subtasks) will be visible in the system's dashboard
kubeflow.py	ML Operations Controller (Kubeflow [56]), Virtualization Manager (Kubernetes [34])	Contains the Python-based definition of the high load pre-processing pipeline that the Data Operations Controller will indirectly invoke via its high-level tasks (E.g. train_model or predict_model). This file is being compile by the dsl-compile command in the Kubeflow build phase

routes.py	API Gateway, Function, Service Discovery Module, Data Operations Controller	Template file for the definition of the REST APIs that are exposed by the LOCUS analytics function based on Flask's annotation-based route definition. It also includes the predefined routes that are used by Consul and the Data Operation Controller for health-checking of the LOCUS analytics function

4.4 SDK Lifecycle of Analytics Functions

In this subsection of the LOCUS SDK Analysis, we go through the various operational phases of the LOCUS Analytics Functions from their registration to activation up until the clean-up phase. These phases are related to automated triggers, or external events, such as API requests that propagate from the user plane (3rd party apps).

Table 4.2 LOCUS operational phases

LOCUS Function Phase	Description
Onboarding Phase	The delivery of a LOCUS Analytics Function to be integrated in the system is referred to as the onboarding phase. A zip file containing all the code, configuration, changes is extracted by an automated task. Then, the dockerfile is used to build the image for the image repository. The service image catalogue meta-data is populated by the JSON entry (via an insert operation on its database) and the image is also registered in the Kubernetes capabilities.
Activation Phase	Triggered by an API activation (subscription) event, the interface propagates this message to the virtualization layer. Each user is the owner of its own dedicated container for each LOCUS analytics function and Kubernetes [34] handles the acquisition of the latest image and generation of the container based on the docker-compose file. After the activation of the container, the python mountpoint is executing the __main__ method which generates a Flask microservice [37] that registers itself to Consul [39], and also to the API gateway using the Zuul [53] and Swagger [54] interface. The Data Collection function is also activated and initializes the Apache Flume [43] execution entry to perform the

	data collection on the frequency and time scheduling that is specified by the oozie.xml. In addition, all the required Kafka [49], RabbitMQ [50] and Hive [45] tables are being generated in the environment (or checked if they already exist).
Main Pipeline Execution	The dag.py file provided by the developers specifies the high-level pipeline that is executed on the instantiation phase. This is handled by Airflow [51] and executed immediately after the spawning of the process. The data operations controller is using http polling to check the status of the LOCUS analytics function at the designated control endpoints. When it acquires a "ready" signal, it invokes the specific action via the /action/{actionName} POST HTTP message. The main pipeline is executed and the same signalling interface is returning a code for "working" until the job is finished. This way the DAG is controlling the sequence of actions that are being executed. In some cases of LOCUS Analytics functions with ML models, the phases that are being executed are more fixed, such as "model_check", "model_train", and "model_predict".
External Pipeline Execution	In the cases of external pipeline execution, the designated REST endpoint of the LOCUS analytics function is triggering the DAG execution. This is done via the Airflow configuration interface which basically passes on the execution arguments to Airflow and triggers the main pipeline (described in the previous phase). In this case the polling operation comes from the user plane (external user), but it utilizes internally the same signalling and control interface.
Clean-up Phase	After the deactivation of a LOCUS Analytics function from the API Subscription API, a clean-up sequence is initialized that will make sure that the virtual resources provided by the virtualized layer will stay available for the next execution. This includes clean-up of Hive [45], Kafka [49], RabbitMQ [50] topics and tables that are not used by other functions, deactivation of containers, deletion of containers, removal of reverse HTTP proxy entries , Swagger entries and Consul entries and deactivation of data collection functions.

5 LOCUS Platform Implementation in the WP6 Activities

In this chapter, the LOCUS Platform's finalized architecture is presented in the context of the demonstration activities under development in WP6. It is evident that the technology selection/ suggestions that have been proposed in the previous chapters, as well as the SDK programming practices/ principles are being applied in a demonstratable software that closely resembles an actual realization of the described high level system architecture of LOCUS. In fact, this platform is undeniably not just a concept, but a realizable, implementable and competitive location-based analytics services platform that can greatly contribute to the 3GPP ecosystem. Lastly, it should be noted, that PoC#2 is instantiated as a standalone system block of the LOCUS Platform therefore implementation activities details are to be included for this PoC in this section.

5.1 PoC#1 Instance System Blocks

PoC#1 aims at leveraging the LOCUS architecture to demonstrate geolocation-powered use-cases in network management.

At the time of writing this document PoC#1 is undergoing development and many of technical details are still to be finalized. With that said, the overall picture consists of instantiating a set of LOCUS system blocks necessary for the collection, delivery and processing of geolocation and network data, both to demonstrate the network management capabilities and LOCUS architecture's ability to enable these capabilities.

The instantiated blocks for PoC#1 can be categorized into 3 categories: Common blocks shared among the story lines emanating from WP4, and story-line specific blocks to deliver specific network management capabilities. This split doesn't eliminate the ability to cohabitate the storylines in one instantiation of all the blocks.

5.1.1 Common Blocks

PoC#1 relies on an instantiation of LOCUS's Management and Orchestration Block (MANO) and the virtualization layer as described in section 3.2.5. Use-case specific virtualized services will rely on the persistence and message queue blocks described in 3.2.6 to establish communication and transport data.

Localization Enablers (LEN) services will also be instantiated to provide geolocation information. Currently plans for implementation concern mostly 5G-based LEN and Hybrid LEN services mentioned in section 2.4, the algorithms of which are described in D3.2 [59] Sections 2, 3 and D3.3 [60] Section 7, in addition to GNSS.

A data collection module will be instantiated to collect necessary network measurements from UEs and base stations, by interfacing with the University of Malaga's (UMA) testbed.

5.1.2 UMA Use-case Blocks

The PoC#1 implements a storyline based on a UMA supported UC targeting the use of contextualized indicators to support network failure management. These will be implemented in the PoC as separate virtualized services. These algorithms are described in D4.1 [61] with further developments to be reported in D4.2 [62].

5.1.3 Orange-supported Use-case Blocks

The PoC#1 implements a storyline based on a UC supported by Orange targeting the use of location based Radio Environment Maps to enable Coverage Optimization and Cell Outage compensation, these algorithms are described in D4.1 [61] with further developments to be reported in D4.2 [62].

5.2 PoC#3 Instance Blocks

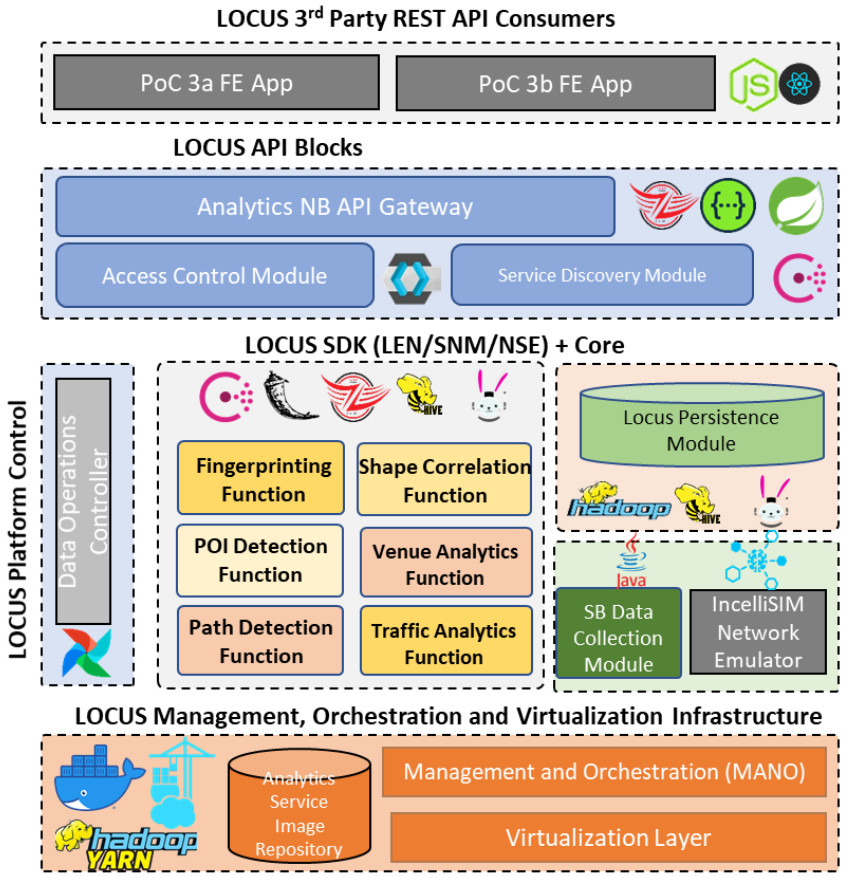

Figure 5.1 The PoC#3a & b blocks in conjunction with LOCUS Platform blocks

The PoC#3 demonstration activities are closely related to the Lambda architecture system blocks of the LOCUS Platform namely the LOCUS Persistence module, data collection functions, data operations controller and API Gateway system blocks. The two vertical

application use cases (namely the “Traffic Analytics” and “Venue Analytics”, see Figure 5.1) will be built on top of this reusable horizontal layer, and they will specialize in the form of their web UI and REST API capabilities, as well as different computations on top of the localization information.

5.2.1 Common Blocks

The horizontal, PoC#3 LOCUS Platform system blocks include:

a) an emulated Southbound 3GPP interface based on Incelligent 4G/5G/WiFi System level simulator [63];

b) a custom (Java-based) Data Collection function implementation that uses the same interface as LOCUS SDK in order to write directly to the Persistence Module;

c) the initialization of the unified persistence schema, namely the UE Measurements schema, the Location/ Localization schema, the Geo-Shape schema and the Geo-Shape Correlation schema; and,

d) a LOCUS Analytics Function LEN implementation that uses the measurement schema to produce location information by means of neural network regression [59][60].

In addition to the above, in the place of the virtualization management and orchestration system blocks, simplified but functional blocks are utilized. However the latter are supported by technologies of essentially the same stack as the one described in the LOCUS Platform technology selection, following the same specifications and making migration very easy. These technologies are a local Docker installation (in a dedicated testbed on Incelligent premises), Hadoop Yarn [64] for resource allocation of the Hive Database, and Portainer IO for container management and monitoring [65] (instead of OSM and Kubernetes which requires larger production infrastructure). Regarding LOCUS Platform Control system blocks, we have also implemented a basic operational version of the data operations controller on top of Apache Airflow [51] that is used to orchestrate the data collection, network emulation and fingerprinting service. Finally, for the LOCUS API layer, the basic functionality of the LOCUS Northbound API Gateway has been implemented based on the referenced technologies (technology-selection), as well as the SSO/ authentication system that acts as the access point to the generated UC-specific Analytics REST APIs.

5.2.2 PoC#3a Blocks

The PoC#3 horizontal system blocks described in the previous section (5.2.1) are used in this use case as a basis for the development of vertical-specific, LOCUS analytics functions based on the specifications of the LOCUS SDK. PoC#3a focuses on venue analytics which require the analysis of location/ localization information to generate identified POIs (Point of Interest) in the form of reusable shapes that are stored in the Geo-Shape schema. This information is

generated via the Data Operations Controller in a dedicated step of the scheduled pipeline that delivers the results into the unified LOCUS schemas. An additional REST API then uses this information converted into analytics for the propagation of venue/ retail related KPIs (for instance, number of users, footfall, average dwell time, etc.). This REST API is registered to the LOCUS API Gateway in order to be ultimately consumed by a front-end application designed to take the role of the 3rd party application in this end-to-end LOCUS Platform proof of concept.

5.2.3 PoC#3b Blocks

PoC#3b is another vertical implementation on top of the horizontal PoC#3 stack. This use case provides transportation analytics by analysing the location information, converting it into more fine detailed ones, such as velocity and direction vectors, and passing it through an expert system. This expert system is assembling velocity profiles which are required to be associated with identified paths in the underlying venue/ zone. For the identification of such paths, an AI-assisted LOCUS Analytics Function is developed that uses clustering techniques to generate path shapes (polylines) and store them into the unified LOCUS schema (Geo-Shapes). After both those tasks are completed, an internal REST API correlates the velocity profiles, the shapes of the identified paths into traffic dashboard information that can be easily visualized or used for automation of various transportation optimization tasks such as reconfiguration of traffic lights, identification of road accidents, traffic congestion, etc. The front-end application of this use case currently displays map widgets with the different paths identified (based on the paths added in the emulator scenario) and their respective velocity profile insights.

5.3 Future PoC enhancements

For the second half of the project's duration, the WP6 activities add more elements related to the LOCUS Platform architecture and focus on resolving issues that were risen during the first review. The changes are split into three categories: a) Connection to an actual network infrastructure Southbound source, provided by the UMA testbed, which will act as input for most of the implemented use cases and act as a real time network data source; b) migration to actual OSM-based infrastructure instead of locally deployment Docker containers in Incelligent premises (PoC#3); and c) New implemented Use Cases which will be instantiated as LOCUS Analytics Functions on the platform. In the Figure 5.2, the finalized demonstration activities system blocks are presented in the context of the LOCUS Platform. The orange circles indicate blocks that have been migrated from existing developments (i.e. the PoC developed system blocks) and with the red circles we indicate new blocks that will be added and

incorporated in order to create a finalized demo architecture that is as close as possible to the implementation guidelines provided in this document.

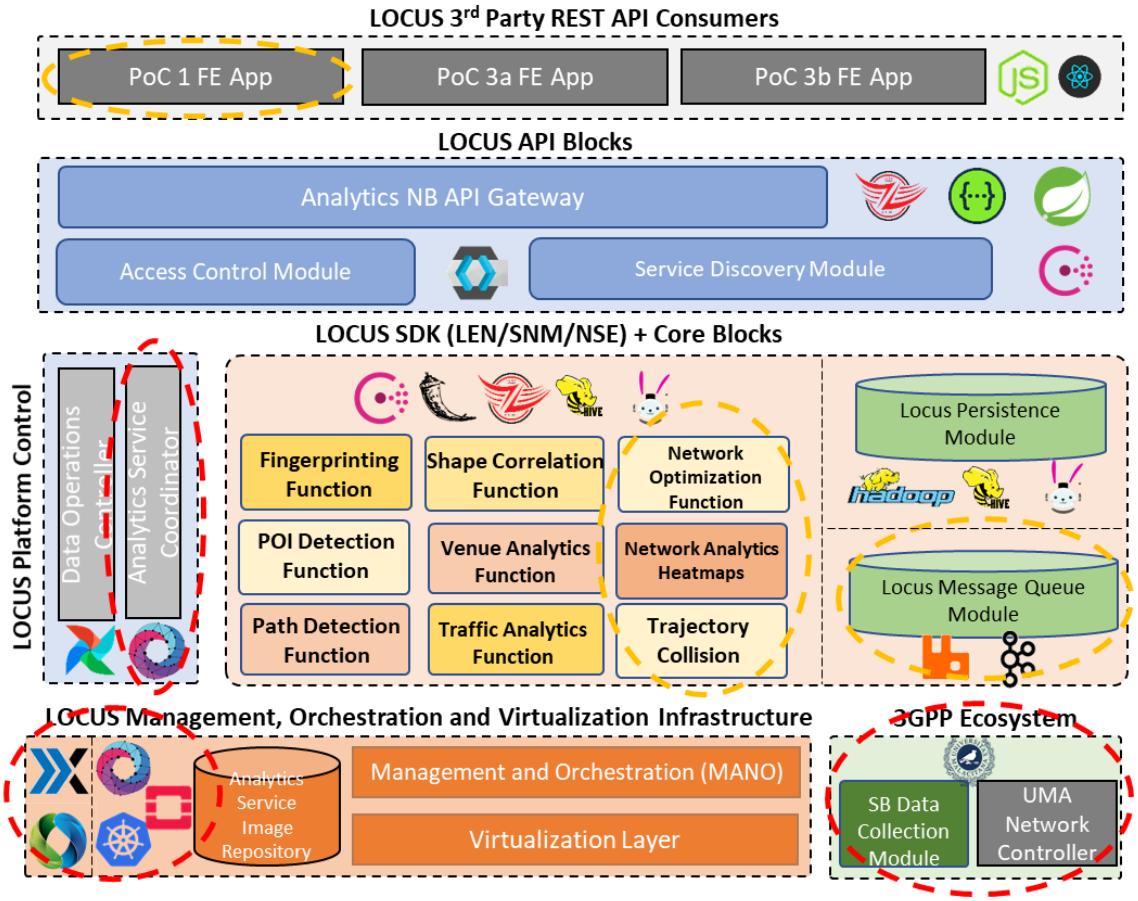

Figure 5.2 The LOCUS finalized demo infrastructure block diagram

5.3.1 Migration to OSM Environment (NXW and OTE)

In the first phase of the WP6 developments, each partner created a local environment to develop the container-based LOCUS analytics functions. The benefit of this approach was a significant speed up in the development process of the various Use cases, up to a point of maturity. After the demonstration of the review took place, the migration activities have initiated in order to move these into the testing (NXW premises) or finalized (OTE premises) OSM-based Kubernetes and hypervisor area. The effort of this migration process is greatly reduced by the fact that the initial design of the LOCUS Platform has remained intact (with respect to the virtualization technologies) therefore all the partners can easily move from localized Docker-based container deployments to the OSM infrastructure. Another important addition that needs to be taken into consideration during this migration process is the incorporation of the “Analytics Service Coordinator” system block, which in the OSM stack is referred to as the OSM Northbound API. This system will allow for an on-demand instantiation of the LOCUS functional containers (which differentiates from the PoC#3 initial design)

showcasing the agility that this technology offers in terms of resource allocation and efficiency.

5.3.2 UMA Testbed as a Southbound Source

Another important change that will be part of the second phase of the WP6 activities is the incorporation of the UMA testbed as the actual 3GPP southbound interface data source. Currently the implemented PoC#3 core system blocks use a system level 4G/5G simulator (IncelliSim [63]) to generate real time capture of physical layer measurements which are then converted into the input schema, as specified in the data model section of this document. This setup helped us develop the demonstration activities end-to-end, calibrate the intermediate ML-based analytics functions and provide a functional prototype for the first WP6 milestone. In order to make a step closer to the real environment of the LOCUS Platform ecosystem, actual (non-emulated) network feeds must be utilized in a tight integration manner with all PoC developments. The UMA testbed is providing its own schema for network measurements of location and received strength (RSRP) of its various radio access devices. According to the schema specification for the LOCUS UE Measurement schema, we will require to transform the UMA data source into a different -yet possible- data structure that will be used for a) offline training of the various WP3/WP4/WP5 models, as well as b) real-time feeding into the analytics function pipeline. This will showcase the readiness of these functional blocks to be put into actual production environments (not only simulated) and also that the platform's design is able to handle the data rates generated and supplied into the internal systems. It must be noted that PoC#1 was co-developed in the UMA environment and therefore minimal changes were required to be made to be in line with these advancements.

5.3.3 Extension to PoC#3 – Trajectory Prediction / Collision Detection

Another possible use case that has already been designed based on the specifications of the LOCUS platform is the trajectory prediction for collision detection function. This PoC's primary output is a downstream service that provides messages (alerts) regarding objects that are being monitored as they move within the boundaries of a venue and other (static) objects that can be a potential collision threat (based on projected trajectories). This use case is designed on top of the streaming part of the LOCUS Platform (the LOCUS Message Queue) to be used by all the internal system blocks as communication and interfacing point. The functions process the location information at different rates and batches; however the end result has a very quick response and it is estimated that it can be used as an actual collision prevention mechanism for low to moderate speed environments. In the following figure (Figure 5.3), the interaction between the system blocks and the UC-specific analytics functions is shown.

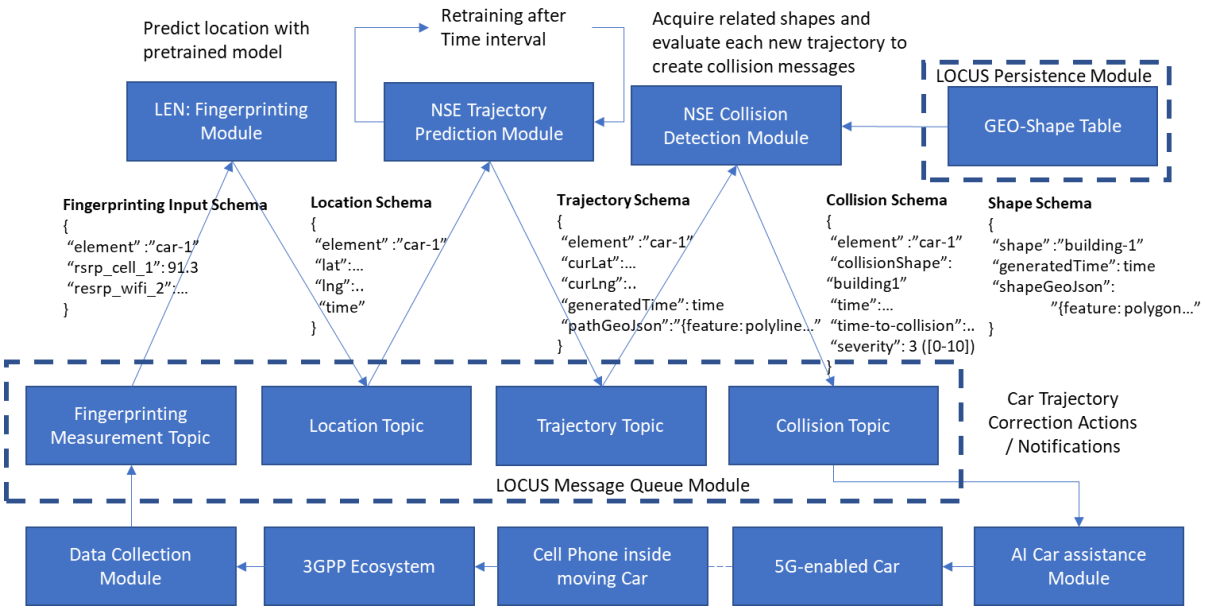

Figure 5.3 Additional PoC mapped to LOCUS system blocks

The functional blocks of this analytics service are analysed thoroughly in the table section below (Table 5.1):

Table 5.1 System blocks related to additional PoC

System Block	System Block Group	Description
LOCUS Persistence Module, Locus Message Module	LOCUS Core Blocks	Initialization of Trajectory and Collision topics, Re-use of the existing unified schema topics (Geo-Shape schema, Location Topic, Fingerprinting Measurement Topic).
Fingerprinting Service	LOCUS SDK (LEN) Blocks	This use case will re-use the existing, fingerprinting service to acquire the generated measurements in the form of location topic messages specified in the unified schema.
NSE Trajectory Prediction Service	LOCUS SDK (NSE) Blocks	This is the core ML block of the use case. It will utilize the SDK capabilities of high-performance ML libraries, ML Model persistence and SDK lifecycle operation for the training and prediction phases. The outputs of the generated trajectories will be used into a custom-generated, UC-specific topic on every new predicted trajectory.
NSE Collision Detection Service	LOCUS SDK (NSE) Blocks	This is the core analytics block of the use case. It re-uses the stored Geo-Shapes that exist within the Geo-Shape schema (Persistence

		Module) in conjunction with geospatial filtering in order to create a notion of playground obstacle zone. This is then used for the generated trajectories (including the location probabilities) to generate predictions for future collision of moving objects. These collisions have their own dedicated topic that is the main outcome of the 3rd party applications inside an assumed 5G, self-driving car infrastructure or car safety setup.
AI Car Assistance Service	LOCUS 3rd Party Pub-Sub consumer/ 3GPP ecosystem	This service is the main consumer of the generated collision events as specified in the use case. It can combine the information acquired from the locus platform to enforce real time decision making (with appropriate access to vehicle controls) in order to either notify the driver about an upcoming collision or to correct the steering / trajectory of the vehicle to a safer trajectory.

6 LOCUS Platform Mapping to Existing Standards

This section presents a small introduction of the existing standards and attempts to contextualize the work being done within the LOCUS project with respect to them, as well as describe the following steps in the standardization efforts of the LOCUS consortium with respect to the LOCUS Architecture. In order to provide a complete view of the standardization context, we include some material from D2.4 [2] which is further enhanced and complemented with new standardization-related aspects.

6.1 3GPP System Architecture for 5G

The 3GPP System Architecture specification group (SA2) has done extensive work in defining the architectural model for localization services in 5G-NR. The enhanced LoCalization Services (eLCS) specification work was carried out in Rel. 16 and the phase 2 of eLCS (eLCS-ph2), targeting high accuracy positioning for Industry IoT, will be conducted in Rel. 17. The architectural model builds on from the 4G LTE releases and supports both emergency/regulatory services and commercial applications.

The main specification document for eLCS architecture is TS 23.273, titled “5G System (5GS) Location Services (LCS)” [66]. The 5G location services related to positioning use cases are detailed in TR 22.872 [67] and asset tracking use cases are detailed in TR 22.836 [68]. The positioning accuracy requirements for vertical industries are captured in the Annex B of TS 22.261 [69]. Broadly speaking, 3GPP SA2 is defining the architecture framework for localization use cases and requirements captured in SA1.

There are three basic types of location requests supported in the 3GPP eLCS architecture. The first is termed as Network Induced Location Request (NI-LR), which is initiated by the serving Access and Mobility Function (AMF) for a UE for an emergency or regulatory requirement. This type of request can override any privacy settings related to the UE. The second type is named as Mobile Terminated Location Request (MT-LR). This is applicable to commercial services and can originate from an external client or an internal network function or an application function of the Public Land Mobile Network (PLMN). The request is made to the PLMN for the location of the target UE. The privacy settings for this client/NF request by the UE is always checked and the request is rejected if it is against the privacy settings. The third type is named as Mobile Originated Location Request (MO-LR). This is used when a UE sends a location request to the serving PLMN for location related commercial application concerning that UE.

The location requests can also be categorised as immediate requests and Location Deferred Requests (LDR). For the immediate location requests, the location response is expected within a short time-frame that is specified in the Quality of Service (QoS) for the service. All of the above three request types (NI-LR, MT-LR and MO-LR) can be formulated as an immediate

request. The LDR however, can be formulated only for the MT-LR type. The external client or NF that makes the LDR expects a location response only when a certain event is triggered (like entering or leaving a certain area) or with a time periodicity.

The 5G system positioning architecture for the non-roaming scenario is depicted in Figure 6.1 below, taken from [66].

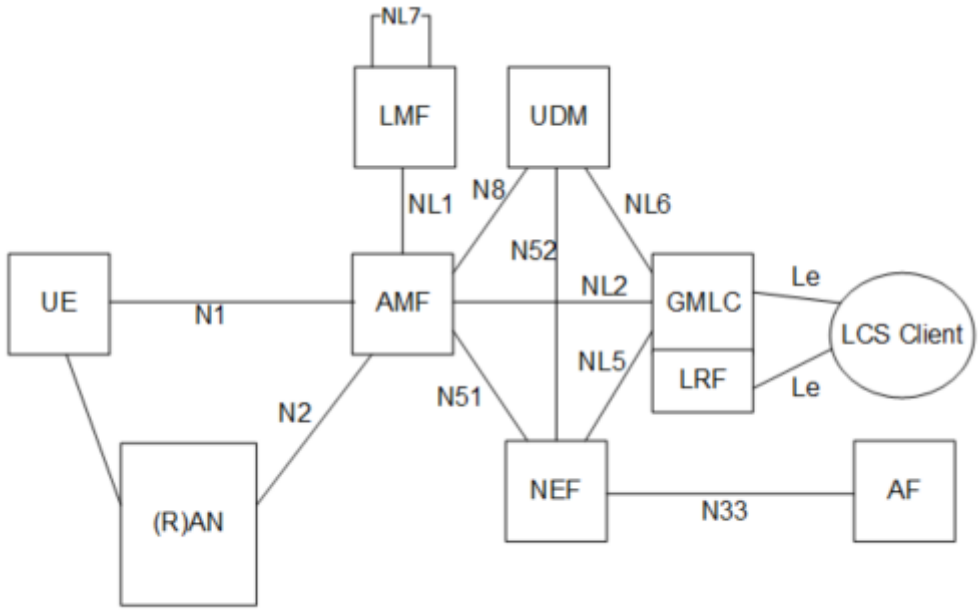

Figure 6.1 Non-roaming reference architecture for Location Services in reference point representation (from [66])

A typical MT-LR example for the call flow messages within this architecture is as follows:

a. The application function or the network function makes a localization request to the GMLC (Gateway Mobile Location Centre) through the NEF (Network Exposure Function). Equally an external location client can directly make a location request to the GMLC, through the Le interface.
b. The GMLC checks with the UDM (Unified Data Management), if the privacy setting for UE allows this request. If not, the request message flow is terminated. GMLC further checks what is the serving AMF (Access and Mobility Function) for the UE.
c. The GMLC makes the 'Provide Positioning Info Request' to the AMF.
d. The AMF provides the response to the above request to GMLC. GMLC passes down the response, to the level of AF or external client.
e. The AMF issues a 'Network Triggered Service Request' to the UE, through the Non-access Stratum (NAS).
f. The UE responds to the request through the NAS.

g. The AMF does the LMF (Location Management Function) selection. This selection is based on the physical closeness to the serving Access Node to the UE, amongst other factors.

h. The LMF instructs for the UE positioning using the Next Generation Radio Access Network (NG-RAN). This can also be the 4G-RAN (LTE), trusted or untrusted non-3GPP radio technology.

i. LMF provides the 'Location_DetermineLocation' response to the AMF.

j. AMF provides the 'Location_ProvidePositioningInfo' response to the GMLC.

k. The GMLC provides the 'LCS service response' to the external client (or the network function through the NEF).

The reference architecture with the Service Based Interfaces (SBI) is shown below in Figure 6.2. With the SBI, every network entity that has an interface line beginning with a dot can communicate with every other similar network entity. With the native virtualisation provided in 5G, these functions can reside anywhere, even in the Cloud.

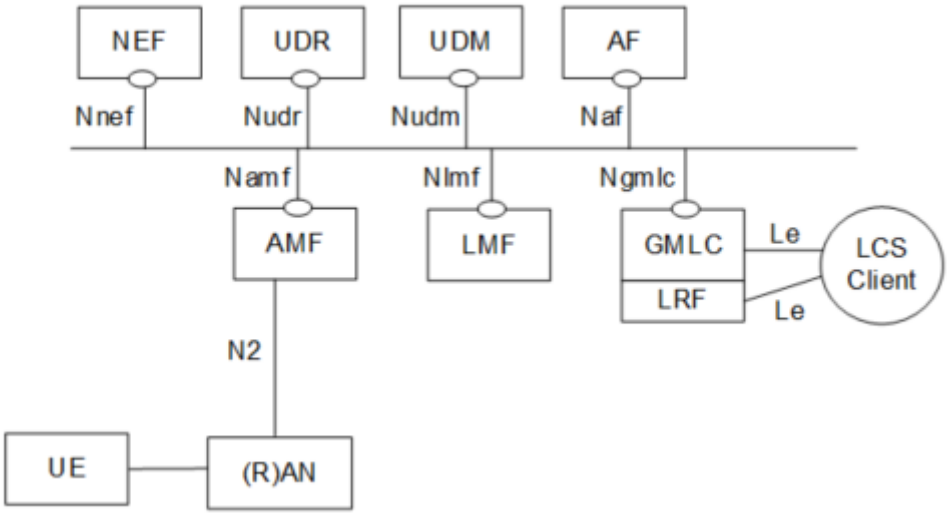

Figure 6.2 Non-roaming reference architecture for Location Services in SBI representation (from [69])

6.2 LOCUS Positioning with Respect to 3GPP and Related Initiatives

LOCUS aims at providing a platform that exploits advanced and accurate localization mechanisms in order to feed a set of analytics functions, which eventually produce localization-enabled analytics to be offered as a service to Smart Network Management and vertical/ 3rd party application providers. The objective and intention of LOCUS and particularly the work regarding LOCUS architecture and its mapping to 5GC architecture is not to reinvent the wheel, but to seamlessly integrate, if possible, its localization-analytics concepts and innovations in the 3GPP ecosystem and standards.

Therefore, in order to bring forward the concepts and technologies developed in LOCUS, i.e. a localization data analytics architecture, we have identified and primarily focused on the standards specifications, studies and technical reports describing and associated with Data analytics.

This mainly involved the series of study items related to analytics functions Network Data Analytic Function (NWDAF), Management Data Analytics Function (MDAF) and in general the enablers for Network Automation (eNA), starting from the study items such as 3GPP TR 23.791 for Rel.16 [70] or 3GPP TR 23.700-91 [71] for the upcoming Rel.17, up to their normative phase such as 3GPP TS 23.288 [72]. In addition, the 3GPP SA6 initiatives to enable new vertical applications are also relevant to the LOCUS architecture work. All of these are briefly presented and discussed in the following subsections with the aim of underlying the alignment with the LOCUS architecture and concepts and its potential to generate impact in the related standards.

6.2.1 Relation with 3GPP Network Data Analytics

Network Data Analytics plays a critical role in the 3GPP's 5G Service-Based Architecture and is a critical cog in generating real-time operational intelligence to drive network automation and service orchestration. Therefore, a dedicated network function is included in the 5G Core architecture, Network Data Analytics Function, which is breaking a new ground in 3GPP, defining standards for analytics processing that have no direct parallels in the 4G/LTE 3GPP architecture. The NWDAF, is the 5G Core function for data collection and data analytics. Initially introduced in the 3GPP TR 23.791 report [70], it can operate in a centralized or distributed model applying different levels of analytics granularity (e.g. global, per network slice). As shown in Figure 6.3, the NWDAF collects metrics or data analytics information locally elaborated from heterogeneous sources, like other 5G network functions, Application Functions (AF), the management system or even external data repositories. The NWDAF processes such data in different means, e.g. through aggregation mechanisms, prediction algorithms, etc., and it generates further analytics data stored locally and offered to other network functions or AFs. Starting from Rel. 16, the 3GPP TS 23.288 [72] specification is standardizing the NWDAF interfaces and the procedures enabling the consumption of data analytics services, supporting query/ reply, as well as subscribe/notify models. In addition, the Open Application Programming Interface (API) are also available in the 3GPP TS 29.520 specification [73].

The 3GPP TR 23.791 report [70] identifies several use cases showing the NWDAF applicability for different network control and management scenarios, with related decision-making logic spanning from the overall infrastructure down to the single network slice, service or traffic flow. Examples of NWDAF related use cases include mobility management, QoS provisioning

and adjustment, policy configuration, SLA guarantee assurance, allocation of edge resources, load balancing, etc. Depending on the specific use case, the target data include application and service-level monitoring data up to network KPIs. However, the currently identified use cases and proposed solutions do not expose the requirement for UE positioning information for further contextualized, location-enriched analytics from which they would significantly benefit (i.e. following an approach similar to LOCUS). The same stands, accordingly, for the NWDAF services (data types, semantics) specified for addressing these use cases and requirements.

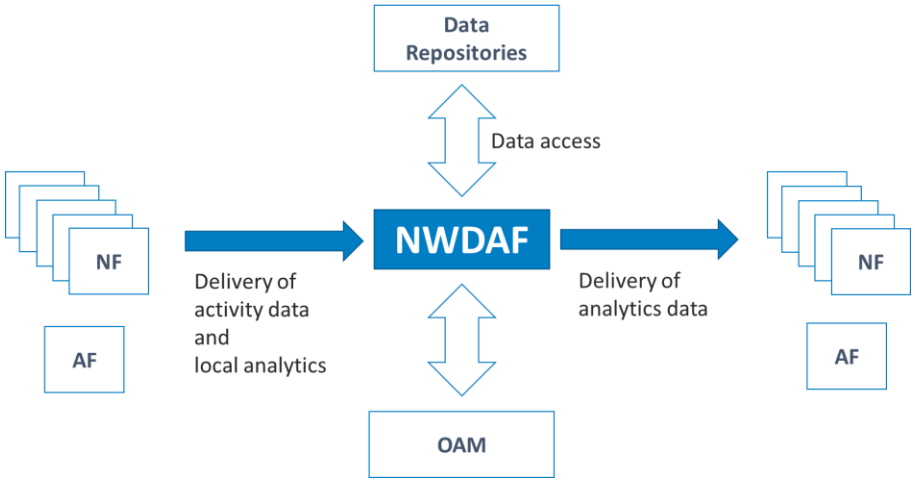

Figure 6.3 NWDAF position in the framework for 5G network automation

6.2.1.1 Location Data Analytics Function in LOCUS

To address the lack of contextualized and location-enriched analytics that as discussed above affect the NWDAF data types, semantics and services, LOCUS identified the need to enrich them with important, meta-localization information/ analytics. This would allow to improve the existing NWDAF use cases and enable the ones defined in LOCUS in the service and network management domains.

As a matter of fact, the LOCUS analytics functions can be represented as Localization Data Analytics Functions (LDAFs), to undertake the part of data analytics exclusively associated with higher level, localization-awareness within the 5G Core. This could mean either a) a tight integration with the existing NWDAF specifications, or b) a completely new entity in the 5G core. In the former case, LOCUS abides by the 3GPP (and NWDAF) service-based paradigm and follows the promoted (pub-sub, HTTP-based) interfaces description and procedures. Accordingly, LOCUS does not need to actually propose new interfaces, but can consider proposing analytics elements (i.e. analytics IDs) and structures with regard to localization analytics. This can be done by adding application attributes and KPIs as the input/ output data in some services described in 3GPP TS 23.288 [72]. In the latter case (standalone), proper interfaces (at least in terms of data and semantics) need to be specified in the context of the

5G Core, e.g. towards other network functions and NWDAF. Although this is quite ambitious and might require further effort, as detailed below similar initiatives can be found in the second phase of network data analytics in 3GPP Rel. 17, where completely new network functions that will be dedicated for data collection, distribution and storage are presented as a proposed solution for optimizing the 5G data architecture.

6.2.1.2 Alignment with evolved Network Data Analytics in 3GPP R17

Beyond the current standard work network data analytics, mostly defined in 3GPP R16, the NWDAF is evolving in R17 in different directions to respond to various emerging data analytics requirements. Indeed, it is important to see LOCUS contribution and impact on standardization by considering the use cases, identified key issues and proposed solutions as these have been reported recently in 3GPP TR 23.700-91 [71], i.e., the second phase of the study on enablers for network automation for the 5G System towards R17. The purpose of this technical report is to study enhancements for analytics and NWDAF for covering (focusing here on the ones related that can be impacted by LOCUS): a) Remaining key issues from Rel. 16, such as UE driven analytics and extensions of user data characteristics that can be used by NWDAF; b) NWDAF architecture enhancements towards multiple NWDAF instances in one PLMN including hierarchies, roles and inter-NWDAF instance cooperation; c) NWDAF features enhancement, e.g. by enabling real-time or near real-time NWDAF communication, including mechanism for data collection and its optimization, e.g. in terms of load; and d) New use cases/ key issues regarding interaction between NWDAF and ML Model & Training Service owned by the operator.

LOCUS is highly relevant and can impact all the above. For instance, we indicatively refer to use case #3 “NWDAF for supporting dispersion analytics” according to which, in R16, NWDAF can provide UE mobility statistics or prediction as specified in 3GPP TS 23.288. Although the above is true, mobility information is not sufficient to determine hot spots areas that may require additional operator attention, i.e. hot spots, POIs or trajectories on which users spend their data and thus need to be dynamically discovered through dispersion analytics. It is therefore identified as a key issue to check if the NFs can benefit from the inclusion of such dispersion analytics, i.e. new type of analytics in addition to UE mobility and communication analytics. On the other hand, the identified issues regarding real time data collection and analytics and their increased efficiency are, at least partially, tackled by the proposed LOCUS data architecture approach. The same stands for the key issue #1 which calls for a logical decomposition of NWDAF and possible interactions between logical functions. This is very interesting and important because first, it goes by LOCUS distributed architecture for serving high speed use cases (e.g. related to AGVs) and second, because the solution proposed therein for addressing this issue and the one related to optimizing data collection efficiency, proposes a completely new NF to stand in between the NWDAF and other NFs (see LOCUS proposed

decomposition and data collection function). Last but not least, as per Key Issue #19, the extension of NWDAF so as to allow for trained data model sharing between multiple NWDAF instances is intensively put forward in this next release, with a solution (solution #5) based on an ML Model sharing architecture (which mimics and can be further impacted by our LOCUS approach) being proposed, respectively.

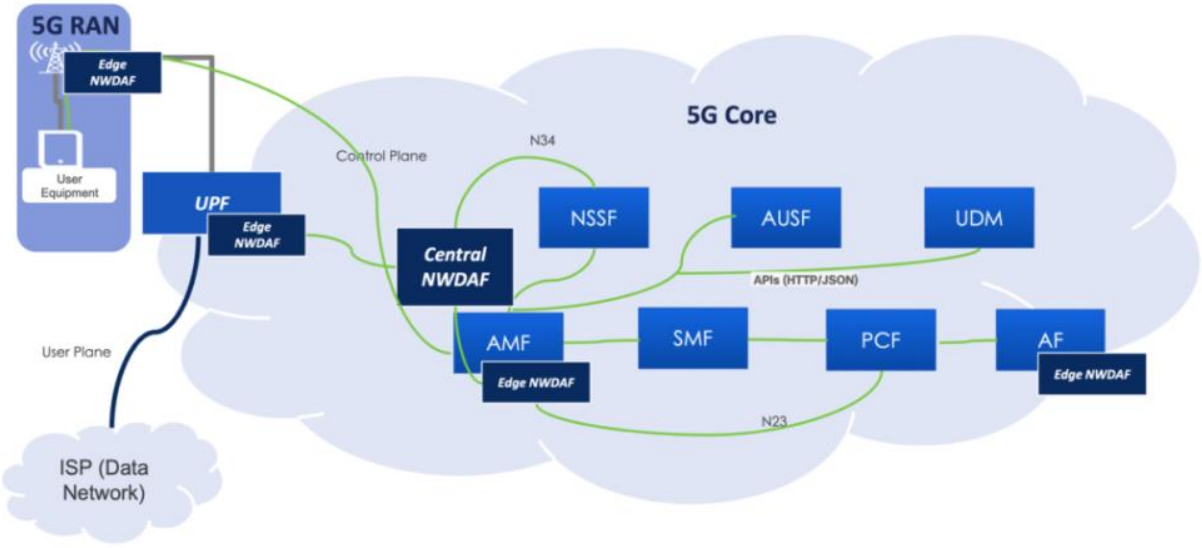

Figure 6.4 NWDAF distributed deployment model [74]

More specifically, the NWDAF is already evolving along these lines in 3GPP TS 23.288 [72] where the data analytics framework is enhanced for R17. In particular, the NWDAF deployment model is shifting towards hierarchical and distributed architectures, introducing additional interfaces and procedures for inter-NWDAF communications. In this sense, the NWDAF is evolving from a monolithic function towards a distributed functional entity with the composition of several elements which can interact and collaborate. A common approach, as depicted in Figure 6.4, is to split the NWDAF in distributed elements, with some functional entities located at the edge for real-time data collection and analytics, supported and coordinated through a central NWDAF (e.g. deployed in the cloud) with functions of data aggregation, post-processing, including when required AI algorithms and training of ML models. It is important to highlight that this approach is aligned with the LOCUS analytics services defined as a combination of analytics functions distributed across edge and core cloud locations to fulfil specific performance requirements, as well as input/ output data constraints.

In addition, even more relevant for LOCUS in terms of alignment and potential standardization impact, the latest version of 3GPP TS 23.288 [72] defined a dedicated architecture for network data analytics collection, distribution and storage. Here, additional functions are identified to facilitate the exchange and maintenance of data. As depicted in Figure 6.5, a Data Coordination and Collection Function (DCCF) coordinates the collection of data from multiple sources, their distribution and, if needed, their storage into the Analytics Data Repository

Function (ADRF). Data delivery can be managed directly by this DCCF or leverage the Messaging Framework, e.g. when there are streaming of real-time data to distribute. Here, the Messaging Framework Adaptor Function (MFAF), enables the interaction between the involved 5G system components and the Messaging Framework offering data translation, formatting and adaptation functionalities. The ADRF on the other hand stores the network analytics data and allows consumers to access historical data.

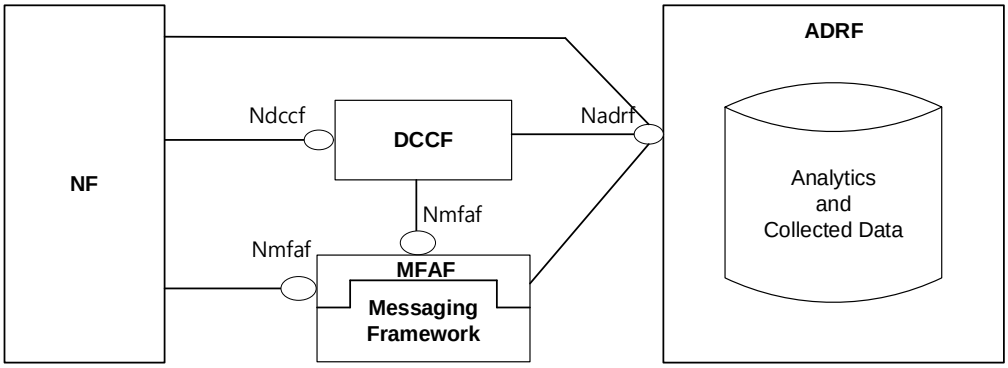

Figure 6.5 Architecture for network data analytics in R17 [72]

This architecture defined for R17 is fully aligned with the LOCUS approach described in the previous chapters, as data from multiple sources can be efficiently collected and distributed among several components and functions, with data consumers (e.g. NWDAF or LDAF instances, as implementation of LOCUS analytics functions and services) able to receive and exchange real-time streams of data (through a message bus/framework) or historical data (through a data persistency framework), depending on the specific analytics function scope and input/output data constraints.

Moreover, the evolution of the network data analytics framework in 3GPP TS 23.288 [72] is also considering a functional split of the NWDAF into two functions, to clearly separate the pure analytics logic from the ML model training and management. This is following the same approach of LOCUS where dedicated ML model and pipeline management functionalities (e.g. for training and creating new versions of the same model) are clearly identified and separated from the actual LOCUS analytics functions (pre-trained) ready to be deployed and operated for specific vertical or smart network management purposes. As shown in Figure 6.6, the Model Training Logical Function (MTLF) takes care of ML models training, exposing dedicated services and APIs for external functions to discover and query them (i.e. mapping to the combination of the ML model repository and ML pipeline coordination functions in LOCUS). On the other hand, the Analytics Logical Function (AnLF) performs inference, elaborates statistics or predictions and it can act as consumer of the MTLF service. A given NWDAF instance can contain MTLF, AnLF, or both, thus allowing for a fully separation of model training and analytics functions as LOCUS does.

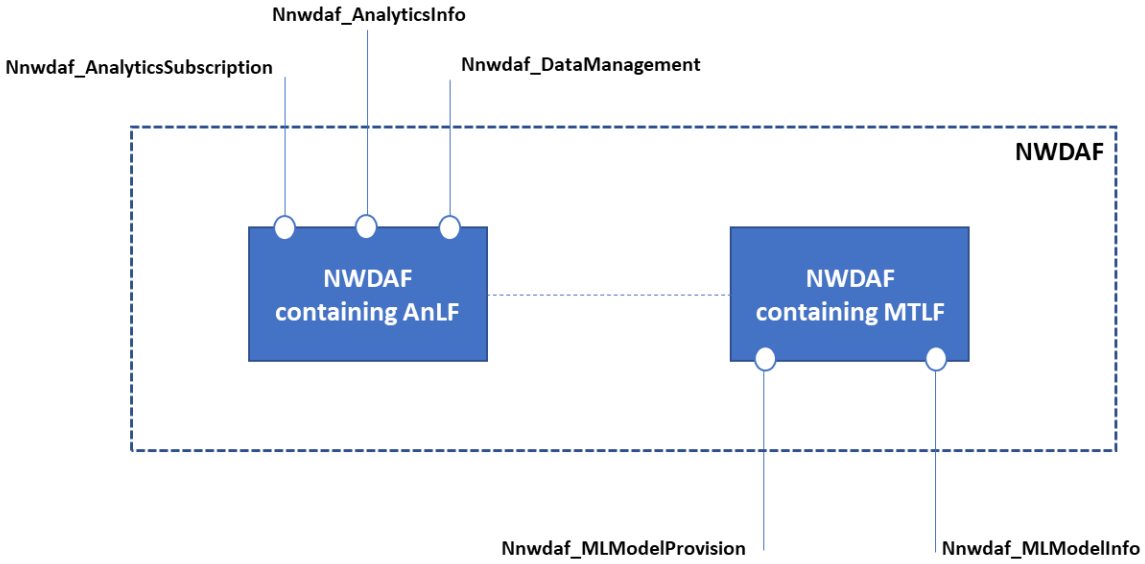

Figure 6.6 NWDAF split in AnLF and MTLF [72]

6.2.2 Relation with 3GPP Management Data Analytics

The 3GPP work around NWDAF in SA2 has mostly concentrated on network core and its associated network functions or AFs, and on analytics interoperability issues in the network user plane and control plane. Although, NWDAF inter-works with the management plane as well, collecting performance and fault management statistics, and radio analytics, the process of discovery and registry for management services as well as the reporting granularity and data preparation are open issues. At the same time, 3GPP SA5 has recently introduced the Management Data Analytics Function (MDAF), which complements and extends the scope of NWDAF with functions integral to network automation and service orchestration that involve monitoring data collection and analytics processing at more aggregate levels, with a view and scope of the network and services in their while (cross-domain) or in one or more, specific domains (Core, RAN, etc.).

The Management Data Analytics is defined in 3GPP TS 28.550 [75] and the latest study on enhancement of Management Data Analytics towards Rel. 17 is given in 3GPP TR 28.809 [76]. As part of SA5, MDAF can assist in management tasks in the phases of preparation, commissioning, operation as well as termination. For example, MDAF can support service provisioning during commissioning, identify performance issues and potential faults during operation and can also assist to predict the network and service demand to enable the timely resource provisioning and service deployments.

Similar to NWDAF and the service-based paradigm, the MDAF framework is based on Management Data Analytics Services (MDAS) that can be consumed by various consumers and in different hierarchies, for instance the management functions (i.e. producers/ consumers

for network and service management, such as the LOCUS MANO), network functions (e.g. NWDAF), network and service optimization tools/ functions, human operators, and AFs, etc.

In contrast to NWDAF, MDAF specification is still quite wide and very open, and by no means tackles the localization-based analytics and proposals elaborated in LOCUS, though for instance they share the need for location and location accuracy in the use case related to coverage issues analysis and optimization.

In general, MDAF is going to play a significant role as an enabler for the automation and cognition of the network and service management and orchestration and it leaves room for LOCUS to be impactful.

6.2.3 3GPP SA6 Initiatives to Enable New Vertical Applications

6.2.3.1 Common API Framework for 3GPP northbound APIs

CAPIF (i.e. Common API Framework for 3GPP northbound APIs) was introduced in Rel. 15 to a) enable a unified Northbound API framework across 3GPP network functions and to ensure that there is a single and harmonized approach for their development [77][78][79]; b) standardize some of the common capabilities that are exposed by the Northbound APIs.

The CAPIF framework has evolved in Rel. 16 (eCAPIF) [80], to bring in key features such as support for 3rd party domains, i.e. to allow 3rd party API providers to leverage the CAPIF framework. Within the standardization of CAPIF, the 3GPP have addressed a variety of different processes, including onboarding/ offboarding of Application Functions, service discovery and management, event subscription and notification, security and charging.

In Figure 6.7, the CAPIF architecture functional model is presented. The main entities are:

- **CAPIF Core Function (CCF):** A repository of all, PLMN and 3rd party, service APIs, allowing for the discovery of the stored APIs by the API invokers and API Exposing Function (AEFs), authentication/ authorization and logging/ charging all API invocations;
- **API Exposing Function (AEF):** The provider of the services to the API invoker;
- **API Invoker:** The applications that avails the services provided by the AEFs/service providers after discovering the related APIs from the CAPIF Core Function and requesting authorization.

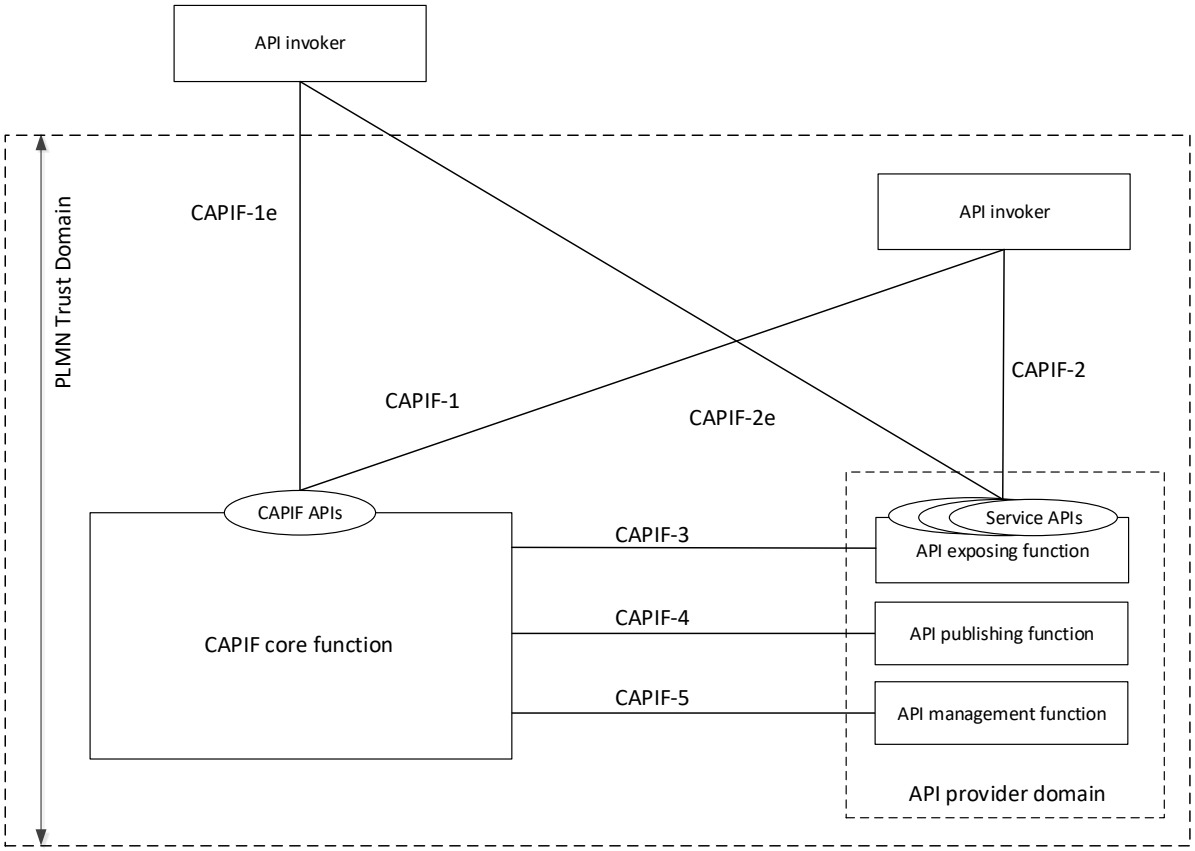

Figure 6.7 CAPIF Architecture - Functional model [80]

The CAPIF architecture is largely in line with the architecture of the functional and system blocks of the LOCUS Platform, especially in the design of the API Gateway system block group. Regarding the main entities of the CAPIF architecture in relation to the LOCUS Platform entities (see also Figure 6.8): a) The "API invoker" entity is closely related to the LOCUS entities "3rd Party Users", "3rd Party Analytics API Consumers", "3rd Party Analytics Pub-Sub Consumers" differentiated by the method of interaction with the API, b) the CCF is related to mostly system blocks from the "Analytics NB API Gateway" which essentially is an aggregation for all the internal "Core" or "LOCUS" functions as well as the LOCUS blocks "Service Subscription Module", "Service Discovery Module", "Access Control Module" and c) the AEF which closely relates to the LOCUS core system blocks based on the LOCUS Analytics Function SDK.

Component by component analysis shows that many of the CAPIF-related functional requirements are already included in the design of the LOCUS platform. In TS 23.222 [80], Table 10.1 indicates all the system components of CAPIF in relation to their functions, and in the table below there is a close comparison between the LOCUS system blocks and their corresponding CAPIF components. In the table we see whether the functional requirements are partially, fully covered or whether the functions are a direct extension of the LOCUS platform.

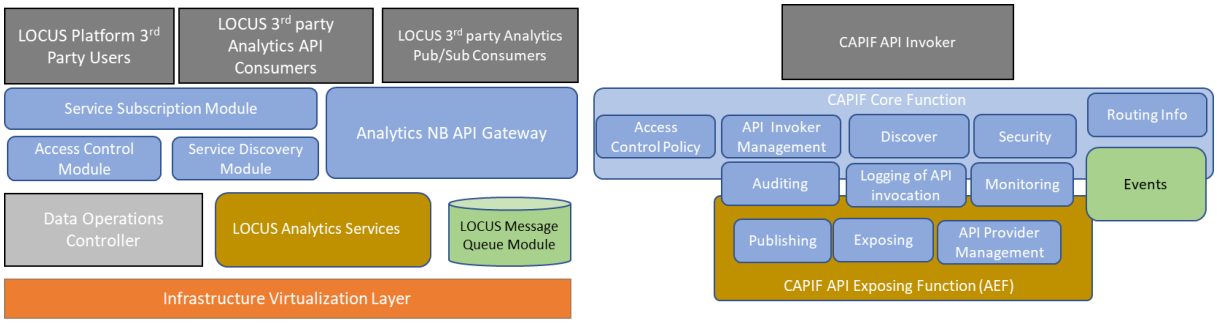

Figure 6.8 System block architecture comparison between the CAPIF blocks and the LOCUS platform related blocks

Table 6.1 CAPIF components alignment with LOCUS

CAPIF Component	LOCUS System Block (s)	Functional Coverage	Notes
Discover Service API	Service Discovery Module, Data Operations Module	Included	The Service Discovery Module assists the Data Operations Module to discover the actual network physical addresses of the LOCUS analytics functional blocks.
Publish Service API	Service Discovery Module, Analytics Function SDK	Included	The Service Discovery Module in conjunction with the LOCUS Analytics function SDK cover this function and the described interfaces.
Events API	Message Queue Module	Partially Included	The Events API is a system that is built on top of message queue block or other message buses. To incorporate this on LOCUS platform, further development must be performed on top of the LOCUS Message Queue.
API Invoker Management API	Service Subscription Module, Access Control Module	Included	The mechanism that links LOCUS platform users with their access rights on LOCUS analytics function services is similar to the mechanism described in the CAPIF specifications.
API Provider Management API	Analytics Service Coordinator	Partially Included	The difference between LOCUS and CAPIF in this block is the fact that LOCUS functions are not 3rd party operated, external providers. The LOCUS Analytics Service

			Coordinator handles the onboarding/ removal of analytics API providers.
Security API	Access Control Module, Gateway Module	Included	The security layer of LOCUS covers the functional requirements specified in the CAPIF specification.
Monitoring API	Management Orchestration and Virtualization Service, NB API Gateway	Extends	The LOCUS Platform can be fairly extended to monitor the usage of the various API calls in the NB Gateway API. Currently, the monitoring part of LOCUS is covered by the virtualization layer (OSM).
Logging of API Invocation API	NB API Gateway/ Data Operations Controller	Extends	Self-Logging, i.e., logging of the Platform's own operation is something that can be done by using some built in web backend frameworks (e.g., Spring framework that is the selected technology of the API Gateway). Therefore, this can be an easy extension of the Platform
Auditing API	Management Orchestration and Virtualization Service, Auditing API	Extends	The LOCUS Platform mainly focuses on the application of highly intensive analytics functions on top of intense amounts of real time data to provide meaningful analytics to higher layers. Auditing on the low-level operations would reduce the performance of these systems especially when the trigger for such executions is the platform itself via the Data Operations Controller. Therefore, auditing can be done in the access layers of the API Gateway but only to keep track of the usage level of the various APIs.
Access Control Policy API	Access Control Module, NB API Gateway	Partially Included	Although it has been analysed in the functional requirements, currently rate limitation enforcement has not been implemented in the LOCUS Platform.

			However, the system block architecture allows for such an extension.
Routing Info API	Management Orchestration and Virtualization	Extends	The LOCUS Platform must take into consideration the operation in a federation environment (e.g. cloud federation or multiple compute facilities), in multiple cloud providers and environments. Implementing this CAPIF function will allow for easier Core to Edge transition.

The implementation guidelines that are derived from the CAPIF specification are very closely related to the functional requirements of LOCUS API blocks as analysed shown in 3.2. More specifically, based on [77] 9.2 “Fundamental API Guidelines” and 9.3 “Architecture Design Considerations”, it is evident that the architectural principles strongly follow the principles followed during the design of the system block architecture and technology selection of the LOCUS Platform in 3.1 and 3.6. These guidelines refer to the design process of the API, namely the usage of atomic REST APIs with no local state and also the applicability of CRUD endpoints, which are identical to the ones specified in 2.6. The LOCUS Platform takes additional liberties with the implementation methodology, specifying microservice architecture instead of monolithic approaches (as described in section 9.3 of [77]), which is a more flexible way to provide the same functions in a more scalable and modern setup.

The CAPIF guidelines is also a methodology to incorporate federation of API providers, with various deployment options and methods that are a clear extension of the LOCUS ecosystem, as is presented in the system block architecture. In order to extend the functionalities of the LOCUS Platform to use 3rd party APIs, the adaptation of CAPIF principles will allow for easier onboarding of external, 3rd party components into the operation (internal) of the LOCUS analytics functions and it will further extend the capabilities of the system.

6.2.3.2 Service Enabler Architecture Layer for Verticals

Introduced in Rel. 16, SEAL (Service Enabler Architecture Layer for Verticals) has been introduced to support vertical applications which require similar core capabilities in a timely manner. 3GPP TS 23.434 [81] specifies application application-enabling services that can be reused across vertical applications (e.g. V2X applications). Services include group management, configuration management, identity/ key management, network resource management and location management.

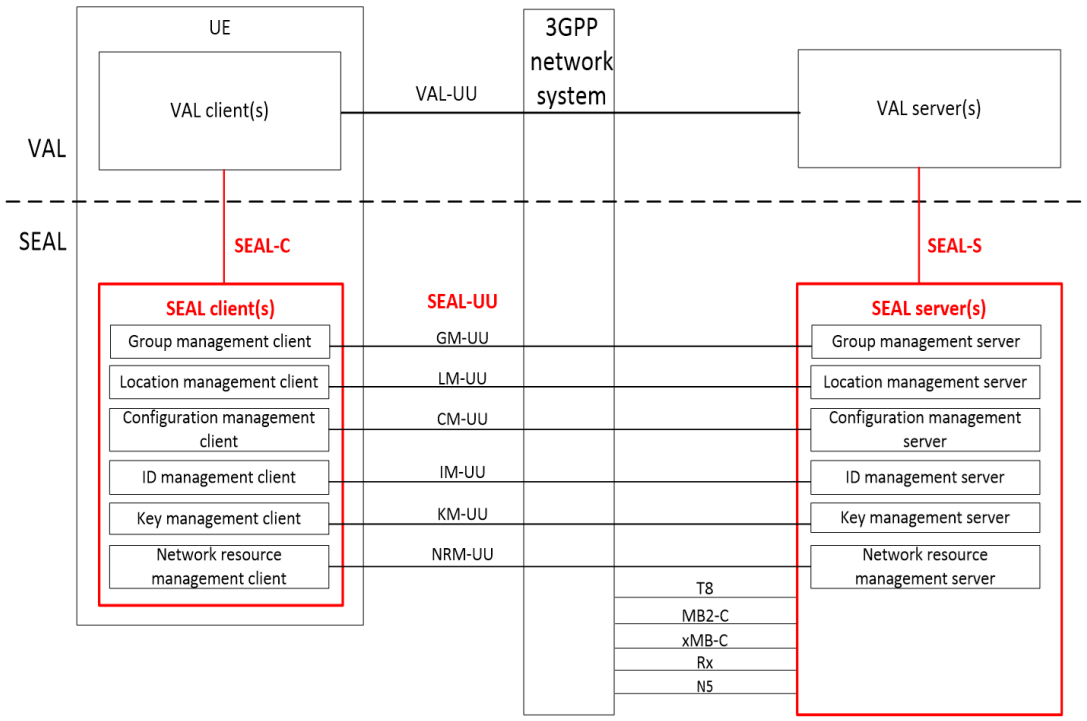

Figure 6.9 SEAL Architecture - Functional model [source: 3GPP SA6]

Figure 6.9 displays the main functional entities in the SEAL Architecture, grouped in two categories:

- **SEAL client(s):** Include client-side functionalities corresponding to the specific SEAL service, supporting Vertical Application Layer (VAL) client(s), as well as with the corresponding SEAL client between the two UEs.
- **SEAL server(s):** Include server-side functionalities corresponding to the specific SEAL service, supporting interactions with the VAL server(s) and SEAL server in distributed SEAL deployments and acting as CAPIF's AEF.

LOCUS work is expected to take into account this initiative as it is relevant to the project scope.

6.2.3.3 Enabling Edge Applications

Several new studies are underway in Rel. 17 further expanding the horizon of new vertical applications within 3GPP, including architecture for enabling Edge Applications. Development of application layer requirements and architecture for hosting Edge Applications on the Edge Data Network. Exposure of northbound APIs towards Edge Applications, and integration of the edge enabling layer with the 3GPP Network. Based on key architecture principles, such as UE mobility, Edge Application portability, service differentiation and flexible deployment 3GPP TR 23.758 [82]. The main functional entities include:

- **Edge Enabler Server:** Provides information related to the Edge Applications to the Edge Enabler Client, exposing the capabilities of 3GPP network to Edge Applications;
- **Edge Enabler Client:** Enables discovery of Edge Applications and provisioning of configuration data;
- **Edge Data Network Configuration Server:** Provides Edge Data Network configuration information to the Edge Enabler Client.

6.2.3.4 3GPP SA6 Initiatives to Enable New Vertical Applications – Altogether

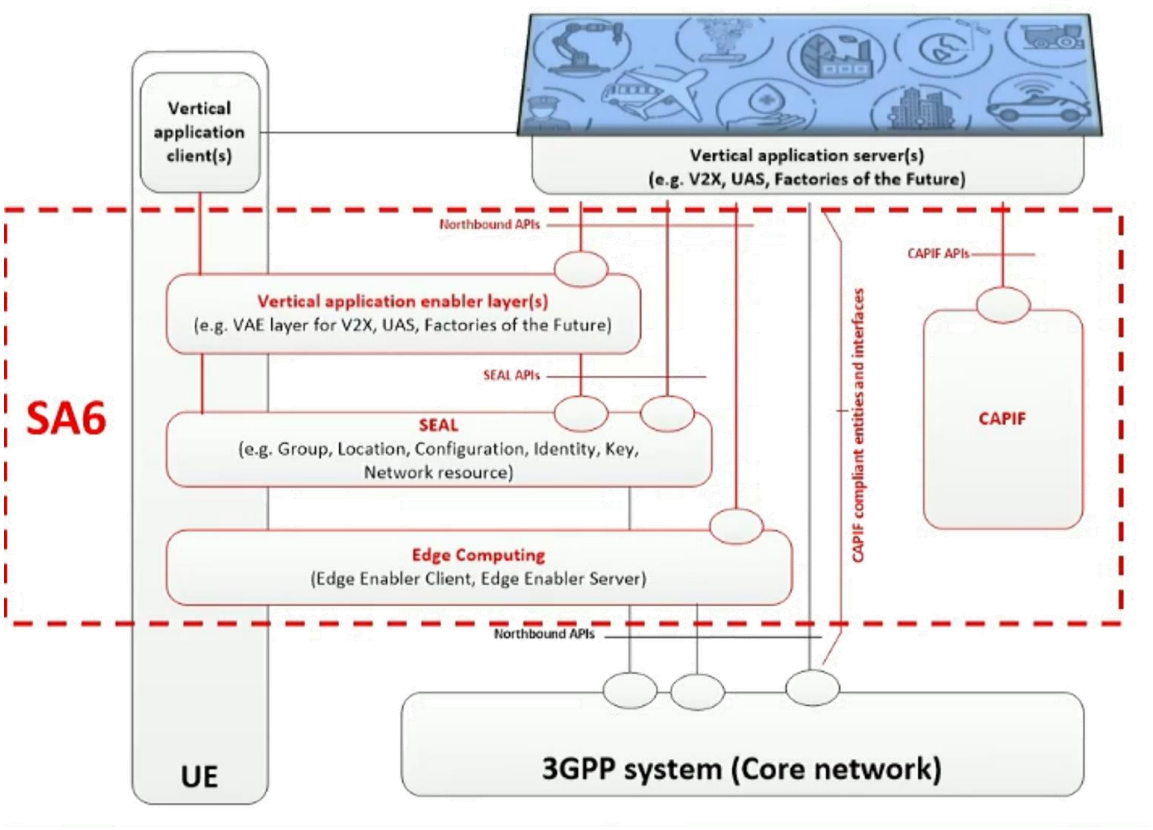

Figure 6.10 3GPP SA6 initiatives [source: 3GPP SA6]

Figure 6.10 summarizes the aforementioned initiatives within 3GPP SA6.

6.3 LOCUS positioning with respect to 5G end-to-end network management and orchestration

The LOCUS Platform, in its architecture principles and design approach is aligned with the ETSI NFV specifications for what concerns the management and orchestration of virtualized functions and services on top of virtualized infrastructures. In particular, as described in the previous chapters, the LOCUS platform relies on specific management, orchestration and

virtualization functionalities to deploy and operate the analytics functions and services as virtualized entities, indeed in full compliance with the ETSI NFV MANO architecture.

As discussed in detail in deliverable D4.3 [16], these functionalities are embedded in the LOCUS MANO, which aims at providing additional capabilities with respect to legacy NFV MANO tools to manage and coordinate the provisioning and operation of the localization analytics functions and services in support of Smart Network Management and 3rd party vertical applications. Indeed, as defined by 3GPP [83], traditional NFV MANO functionalities are key enablers for automated management of 5G network functions and services as part of the network slicing paradigm, being fundamental part of the 5G network management architecture. Therefore, the LOCUS MANO offers additional per-network-slice functionalities to be offered on-demand.

It is worth to highlight that, as reported in D4.3 [16], such approach is aligned with current evolution trends in cross-layer management and orchestration of end-to-end network slices, where network and physical infrastructure resources, NFV services and virtualized infrastructures leverage on different management functions. As depicted in Figure 6.11, this cross-layer approach allows to seamlessly integrate 5G radio resources and infrastructures, Software Defined Networking (SDN) enabled network resources and segments, NFV enabled core and edge virtualized infrastructures. In this context, end-to-end service and slice management functions rely on the NFV MANO for the deployment of virtualized services and functions (e.g. for 5G Core network) at edge and core computing locations. At the same time, end-to-end service and slice management functions take care to deliver end-to-end connectivity services (spanning 5G RAN, backhaul and transport) in support of end-to-end network slices. Here, the LOCUS MANO can be seen as an enhanced NFV MANO framework that manages localization analytics services in addition to the traditional NFV services required to implement end-to-end network slices.

In practice, as depicted in Figure 6.11, within the 5G end-to-end network management and orchestration scenario, LOCUS provides a comprehensive analytics platform that wraps the LOCUS MANO functionalities and offers through its API layer system blocks on-demand analytics functions and services. These are consumed by Smart Network Management applications (at the level of end-to-end service and slice management, for advanced 5G radio resource management) and 3rd party vertical applications (to improve vertical services with localization-based analytics data processing).

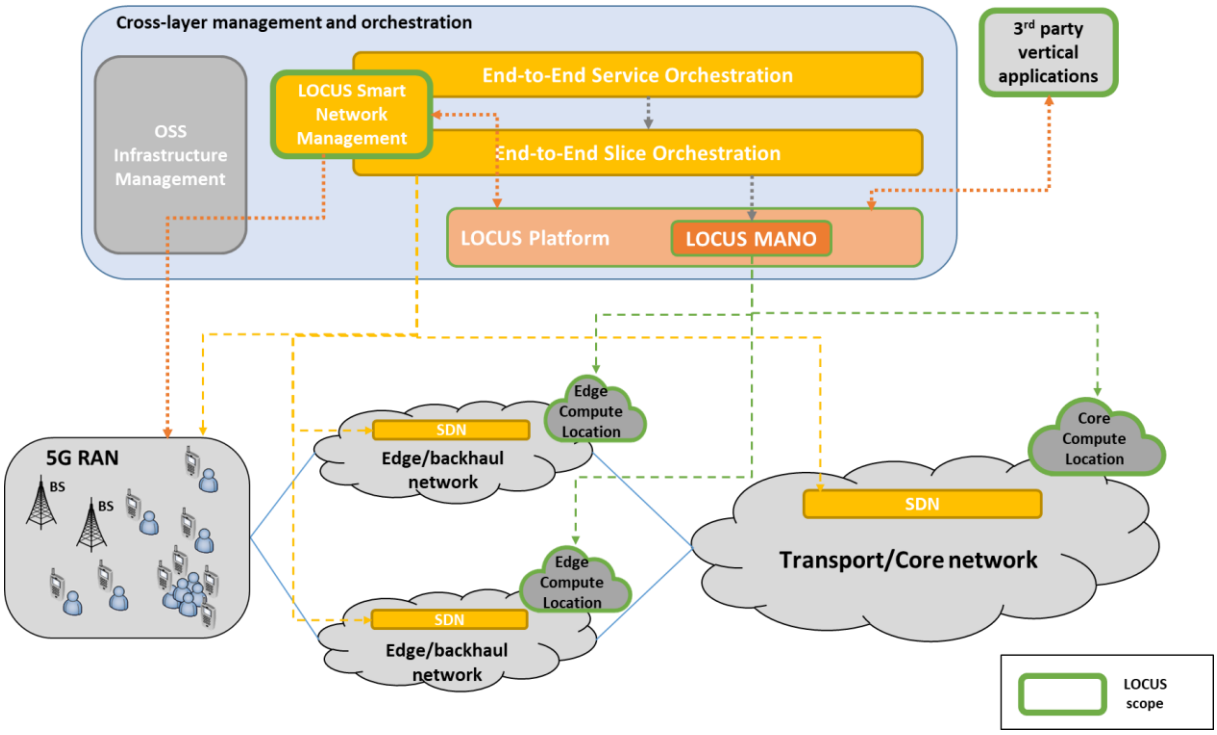

Figure 6.11 LOCUS positioning in 5G end-to-end network management and orchestration [16]

7 Conclusions

To sum up, the LOCUS Platform is a platform that supports location-based analytics. In fact, it is a multi-block system that is designed to serve multiple layers of analytics services in an environment that is both connected to the 3GPP ecosystem, but also to external entities (i.e. data sources, stakeholders, 3rd party users). Its key source of data is network signalling, which is converted into location information, a key ingredient for enabling geospatial analytics services. The LOCUS processing pipelines handle the processing of such information with considerations for the amount of data produced, processing requirements and appropriate, flexible underlying virtualization technologies to support it. In addition to the generation of localization information, the LOCUS Platform is equipped with built-in tools for advanced processing of this information to produce more sophisticated outcomes, such as geospatial analytics, geometries, etc., and to correlate this information with other sources of data (e.g. network analytics, retail analytics, transportation analytics and so on). Another key aspect of the platform is its built-in support for large scale, production-ready deployment of AI models, which are incorporated in each LOCUS use case function implementation but natively supported by the platform system blocks. The LOCUS Platform's specification includes multiple interoperable and standardized input and output data types, making the incorporation of all the generated results (intermediate or finalized) very easily accessible to 3rd party consumers or other 3GPP-based services, which can then be easily integrated. An important outcome of the work that has taken place also includes the design of an appropriate SDK for the development and extension of the LOCUS Platform, based on the selected capabilities of the system. In fact, significant effort was dedicated to the reduction of the complexity of any implementation for additional analytics by adding key libraries and interfaces in the analytics function space.

This document is not limited to the detailed design and listing of the functional capabilities of the platform, but it also includes the selection and description of selected actual technologies, based mainly on open source, that can be used to instantiate it. The LOCUS demonstration activities (that are part of WP6) will utilize these technologies to instantiate the various PoCs proving that the LOCUS Platform is not only a theoretical concept, but a realizable infrastructure regardless of the complexity it encloses. Finally, attention has been given to the correlation of the LOCUS Platform with other 3GPP entities, which are either mature or under specification design, to find common ground or areas of completion. The LOCUS Platform was initially designed as a standalone ecosystem that requires only south-bound interfacing with the 3GPP network, however after this comparison we could clearly see that internal, platform blocks can be integrated more deeply into the operational stack of 3GPP, converting LOCUS from a SaaS to a PaaS extension of the 3GPP architecture.

References

[1] Deliverable 2.1 EU LOCUS project “Scenarios, use cases and requirements”

[2] Deliverable 2.4 EU LOCUS project “System Architecture: preliminary version”

[3] 3GPP Technical Specification, TS 29.572, 5G System; Location Management Services; Stage 3, July 2019

[4] JavaScript Object Notation, https://www.json.org/json-en.html

[5] (e)Xtensible Markup Language, https://www.w3.org/TR/xml/

[6] YAML Ain't Markup Language, https://yaml.org/

[7] Abstract Syntax Notation, https://www.itu.int/en/ITU-

[8] Graph Query Language, https://spec.graphql.org/June2018 /

[9] Keyhole Markup Language, https://www.ogc.org/standards/kml

[10] Geospatial Data JSON, https://tools.ietf.org/html/rfc7946

[11] M.Kiran, P. Murphy, I. Monga, J. Dungan “Lambda architecture for cost-effective batch and speed big data processing”, IEEE International Conference on Big Data, October 2015

[12] T. Zschornig, R. Wehlitz, B. Franczyk, “A Personal Analytics Platform for the Internet of Things - Implementing Kappa Architecture with Microservice-based Stream Processing”, 19th International Conference on Enterprise Information Systems, January 2017

[13] ETSI TS 103 300-2 Intelligent Transport System (ITS); Vulnerable Road Users (VRU) awareness; Part 2: Functional Architecture and Requirements definition; Release 2, V2.1.1 (2020-05)

[14] Information Framework (SID), TM Forum, https://www.tmforum.org/information-framework-sid/

[15] ETSI GS ZSM 002 v1.1.1, "Zero-touch network and Service Management (ZSM); Reference Architecture", https://www.etsi.org/deliver/etsi_gs/ZSM/001_099/002/01.01.01_60/gs_ZSM002v010101p.pdf

[16] Deliverable 4.3 EU LOCUS project “Implementation of the Virtualization platform for network control and management: preliminary version”

[17] ETSI GS NFV-SOL 006, “Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models; NFV descriptors based on YANG Specification”, August 2020

[18] ETSI GS NFV-SOL 004, “Network Functions Virtualisation (NFV) Release 3; Protocols and Data Models; VNF Package and PNFD Archive specification”

[19] ETSI GS NFV-SOL 007, https://docbox.etsi.org/ISG/NFV/open/Publications_pdf/Specs-Reports/NFV-SOL%20007v3.3.1%20-%20GS%20-%20NSD%20file%20structure%20spec.pdf

[20] 3GPP Technical specification, TS 33.107, 3G security; Lawful interception architecture and functions, Release 15, June 2019

[21] J. Mirkovic, “Privacy-Safe Network Trace Sharing via Secure Queries,” in Proceedings of ACM CCS Workshop on Network Data Anonymization, October 2008.

[22] V. Sharma, G. Bartlett, and J. Mirkovic, “Critter: Content-Rich Traffic Trace Repository”, In Proceedings of the 2014 ACM Workshop on Information Sharing & Collaborative Security (WISCS ’14). Association for Computing Machinery, New York, NY, USA, 13–20, 2014 DOI:10.1145/2663876.2663886

[23] Deliverable 5.3 EU LOCUS project “Design of the localization & analytics as a service solution”

[24] VIAVI Solutions Inc., “VIAVI NITRO GEO – Geoperformance Overview” brochure, https://www.viavisolutions.com/en-us/literature/nitro-geo-geoperformance-overview-brochures-en.pdf (Last accessed: September 30th, 2021)

[25] Avro message format, https://avro.apache.org/docs/current/spec.html

[26] Google Protocol Buffers, https://developers.google.com/protocol-buffers/docs/proto3

[27] Base64 Specification, https://datatracker.ietf.org/doc/html/rfc4648

[28] Python Pickle object serialiation format, http://formats.kaitai.io/python_pickle

[29] ArcGIS Online website, https://doc.arcgis.com/en/arcgis-online/reference/geojson.htm

[30] JSON SQL table schema format, http://dataprotocols.org/json-table-schema

[31] 3GPP Technical Specification, TS 32.421, “Trace concepts and requirements (Release 15)”, June 2019, Retrieved from: https://www.3gpp.org/ftp/tsg_sa/WG5_TM/TSGS5_125/SA_84/32421-f10.doc

[32] ETSI Open-Source MANO, https://osm.etsi.org/

[33] Openstack, https://www.openstack.org/

[34] Kubernetes, https://kubernetes.io/

[35] Docker, https://www.docker.com

[36] Python, https://www.python.org/

[37] Flask, https://palletsprojects.com/p/flask/

[38] MLflow, https://mlflow.org/

[39] Consul, https://www.consul.io/

[40] Keycloak, https://www.keycloak.org/

[41] Spring Framework, https://spring.io/

[42] Apache Oozie Workflow Scheduler for Hadoop, https://oozie.apache.org/

[43] Apache Flume, https://flume.apache.org/

[44] Apache Hadoop, https://hadoop.apache.org/

[45] Apache Hive™, https://hive.apache.org/

[46] Trino, https://trino.io/

[47] Apache Parquet, http://parquet.apache.org/

[48] Apache ORC, https://orc.apache.org/

[49] Apache Kafka, http://kafka.apache.org/

[50] RabbitMQ, https://www.rabbitmq.com/

[51] Apache Airflow, https://airflow.apache.org/

[52] “Microservices with Spring”, https://spring.io/blog/2015/07/14/microservices-with-spring, last accessed September 30th, 2021

[53] Zuul, https://zuul-ci.org/

[54] Swagger, https://swagger.io/

[55] Fast API, https://fastapi.tiangolo.com/

[56] Kubeflow, https://www.kubeflow.org/

[57] Seldon ML, https://deploy.seldon.io/

[58] Python Pandas Library https://pandas.pydata.org/

[59] Deliverable 3.2 EU LOCUS project “5G Based Advanced Localization Solutions”

[60] Deliverable 3.3 EU LOCUS project “Integrated localization technologies: preliminary version”

[61] Deliverable 4.1 EU LOCUS project “Localization & Analytics for Smart Network Management”, preliminary version”

[62] Deliverable 4.2 EU LOCUS project “Localization & Analytics for Smart Network Management”, final version”

[63] IncelliSIM system level simulator, https://incelligent.net/products-services/211-simulator

[64] Hadoop Yarn https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[65] Portainer.IO, https://www.portainer.io/

[66] 3GPP Technical Specification, TS 23.273, “5G System (5GS) Location Services (LCS) – Stage 2”, version 16.2.0, December 2019

[67] 3GPP Technical Report, TR 22.872, “Study on positioning use cases”, version 16.1.0, September 2018

[68] 3GPP Technical Report, TR 22.836, “Study on asset tracking use cases”, version 17.1.0, December 2019

[69] 3GPP Technical Specification, TS 22.261, “Service requirements for the 5G system”, version 17.3.0, July 2020

[70] 3GPP Technical report, TR 23.791, “Study of enablers for Network Automation for 5G”, version 1.2.0, December 2018

[71] 3GPP Technical report, TR 23.700-91, Study on enablers for network automation for the 5G System (5GS); Phase 2, version 17.0.0, December 2020

[72] 3GPP Technical Specification, TS 23.288, “Architecture enhancements for 5G System (5GS) to support network data analytics services”, version 17.2.0, September 2021

[73] 3GPP Technical Specification, TS 29.520, 5G System; Network Data Analytics Services; Stage 3, version 17.4.0, September 2021

[74] TM Forum, S. Marappan, “NWDAF: Automating the 5G network with machine learning and data analytics”, June 2020

[75] 3GPP Technical Specification, TS 28.550, Management and orchestration; Performance assurance, version 16.8.0, September 2021

[76] 3GPP Technical Report, TR 28.809, “Study on enhancement of management data analytics”, version 17.0.0, April 2021

[77] 3GPP Technical Specification, TS 23.222, “Common API Framework for 3GPP Northbound APIs”, September 2017

[78] 3GPP Technical Specification, TS 33.122, “Security aspects of Common API Framework (CAPIF) for 3GPP northbound APIs”, January 2018

[79] 3GPP Technical Specification, TS 29.222, “Common API Framework for 3GPP Northbound APIs”, March 2018

[80] 3GPP Technical Specification, TS 23.222, “Common API Framework for 3GPP Northbound APIs; Stage 2”, version 17.5.0, June 2021

[81] 3GPP Technical Specification, TS 23.434, “Service Enabler Architecture Layer for Verticals (SEAL); Functional architecture and information flows”, December 2018

[82] 3GPP Technical Report, TR 23.758, “Study on application architecture for enabling Edge Applications”, March 2019

[83] 3GPP Technical Report, TR 28.801, “Telecommunication management; Study on management and orchestration of network slicing for next generation network (Release 15)”, version 15.1.0, January 2018