

# Overall ORCHIDE Solution Architecture and Design


## D3.1

	<i>Name &amp; Role (for signature see electronic workflow)</i>
<b>Written by</b>	Dragoș PETRE (THR), Adèle KARAM HANKACHE (TAS), Virgile ROBLES (TAR)
<b>Verified by</b>	Sergiu WEISZ (UNSTPB), Maria-Elena MIHĂILESCU (UNSTPB), Virgil ROBLES (TAR), Adele KARAM HANKACHE (TAS), Andreea Cătălina PIETRICICA (THR), Mădălin-Andrei ANDREICA (THR)
<b>Advisory Board</b>	Guillaume Pierre (IRISA), Nicolas Longepe (ESA), Fabien Vigeant (CNES), Daniel Smith (DSI)
<b>Consortium</b>	KPL, TAR, TAS, THR, UNSTPB
<b>Approved by</b>	Adèle KARAM HANKACHE, Coordinator (TAS-F)

Approval evidence is kept within the document management system

**PUBLIC**

This document is not to be reproduced, modified, adapted, published, translated in any material form in whole or in part nor disclosed to any third party without the prior written permission of THALES ALENIA SPACE.

 Co-Funded by the European Union Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

## Change Records

ISSUE	DATE	§ CHANGE RECORDS	AUTHOR
001	15.07.2024	First issue of the document	Dragos Petre
002	25.07.2024	Integrated first round of feedback and updated the architecture	Dragos Petre
003	31.07.2024	Added the SDK architecture	Virgile Robles

**PUBLIC**

This document is not to be reproduced, modified, adapted, published, translated in any material form in whole or in part nor disclosed to any third party without the prior written permission of THALES ALENIA SPACE.



Co-Funded by the European Union Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

## Table of contents

<b>1</b>	<b>Introduction .....</b>	<b>5</b>
1.1	Scope and purpose.....	5
1.2	Applicable documents.....	5
1.3	Reference documents.....	5
1.4	Definitions and Acronyms.....	5
1.5	Document outline.....	6
<b>2</b>	<b>Solution Architecture Overview .....</b>	<b>7</b>
2.1	Functional Overview .....	7
2.2	System Overview .....	7
2.3	Architecture Principles .....	8
2.3.1	<i>Re-Usability</i> .....	9
2.3.2	<i>Extensibility</i> .....	9
2.3.3	<i>Compatibility</i> .....	9
2.3.4	<i>Scalability</i> .....	9
2.3.5	<i>General Constraints</i> .....	9
2.4	Key components and modules .....	10
2.4.1	<i>ORCHIDE Ground System</i> .....	10
2.4.2	<i>ORCHIDE Space System</i> .....	13
2.5	Architectural Views .....	13
<b>3</b>	<b>Logical View .....</b>	<b>14</b>
3.1	Actors.....	14
3.1.1	<i>Service Provider</i> .....	15
3.1.2	<i>Satellite Owner</i> .....	15
3.1.3	<i>Application Developer</i> .....	15
3.2	Logical Components of the Architecture .....	16
3.2.1	<i>Orchestrator</i> .....	16
3.2.2	<i>Ground System</i> .....	18
3.3	Description of the Logical Interfaces .....	19
3.3.1	<i>Orchestrator External Logical Interfaces</i> .....	20
3.3.2	<i>Orchestrator Internal Logical Interfaces</i> .....	21
3.3.3	<i>Ground System Logical Interfaces</i> .....	22
<b>4</b>	<b>Physical Architecture of the System .....</b>	<b>23</b>
4.1	Components General Description .....	23
4.1.1	<i>ORCHIDE Ground System</i> .....	23
4.1.2	<i>ORCHIDE Orchestrator</i> .....	23
4.2	Orchestrator Physical Architecture Overview.....	24
<b>5</b>	<b>SDK .....</b>	<b>27</b>
5.1	Application Builder .....	27
5.1.1	<i>Dependency expression</i> .....	27
5.1.2	<i>Automatic build</i> .....	29
5.1.3	<i>Packaging for ORCHIDE</i> .....	29

**PUBLIC**

This document is not to be reproduced, modified, adapted, published, translated in any material form in whole or in part nor disclosed to any third party without the prior written permission of THALES ALENIA SPACE.




Co-Funded by the European Union Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

5.2 Workflow Builder ..... 30  
A.1 Annex.....32  
A.1.1 Images ..... 32

**PUBLIC**

This document is not to be reproduced, modified, adapted, published, translated in any material form in whole or in part nor disclosed to any third party without the prior written permission of THALES ALENIA SPACE.

 Co-Funded by the European Union Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.

## 1 Introduction

### 1.1 Scope and purpose

The Solution Architecture and Design Document outlines the comprehensive design and architecture of the ORCHIDE system, serving as a foundational guide for its development, deployment, and maintenance. It includes a high-level system overview, explains each building block of the system, and details the interactions between them.

### 1.2 Applicable documents

Internal code / DRL	Reference	Issue	Title
AD1	HORIZON-CL4-2022-SPACE-01-11	001	HEUROPE - ORCHIDE - Part B – VF (ORCHIDE Proposal)

### 1.3 Reference documents

Internal code / DRL	Reference	Issue	Title
D2.1	0005-0018165925	001	Mission Analysis and ORCHIDE Business Impact
D2.2	0005-0018165939	003	State of the Art of edge computing orchestration and Unikernels technologies
D2.3	N/A	005	User Requirements Document and Quality Performance Metrics
D3.2	0005-0018572751	001	Overall ORCHIDE Solution Development and Integration Plan

### 1.4 Definitions and Acronyms

<b>ORCHIDE</b>	Orchestration of Reliable Computing on Heterogeneous Infrastructures Deployed at the Edge
<b>SDK</b>	Software Development Kit
<b>API</b>	Application Programming Interface
<b>FPGA</b>	Field-programmable Gate Array
<b>GPU</b>	Graphics Processing Unit
<b>HMI</b>	Human-Machine Interface
<b>HW</b>	Hardware
<b>OS</b>	Operating System
<b>PAAS</b>	Platform as a Service

**PUBLIC**

This document is not to be reproduced, modified, adapted, published, translated in any material form in whole or in part nor disclosed to any third party without the prior written permission of THALES ALENIA SPACE.

<b>VM</b>	Virtual Machine
<b>COTS</b>	Commercial of the Shelf
<b>POSIX</b>	Portable Operating System Interface
<b>OCI</b>	Open Container Interface
<b>CI</b>	Continuous Integration
<b>CD</b>	Continuous Deployment
<b>I/O</b>	Input/Output

## 1.5 Document outline

Section 1 describes the purpose, scope and application field of this document. In addition, it gives useful information for a better understanding of the document such as applicable documents, reference documents, specific definitions used or refer to in the document and at least the list of acronyms used. A brief presentation of its content completes this section.

Section 2 presents the ORCHIDE solution architectural overview, defining the main components of the system and the constraints the system must follow in its development. Moreover, this section goes into the general ideas behind each of the identified components.

Section 3 introduces the logical view of the ORCHIDE system, giving insights of the logical building blocks identified and defined. It goes into the general details of each of the building blocks and how they interconnect with each other, offering an overview of both the internal, as well as the external logical interfaces. Additionally, the interfaces are described in more details and mapped to the previously identified user requirements.

Section 4 describes the physical architecture of the ORCHIDE solution. It offers details about the physical components of the system and the mapping between them and the logical building blocks. This section also presents the possible solutions for the identified components and gives more details on the structure of the different processing nodes.

Section 5 provides a description of the architecture of the ORCHIDE SDK. It gives information about the application builder and the workflow builder, diving deeper into their feature and the connection between their features and the defined logical interfaces.

Appendix 1 contains the main diagrams for the logical and physical overview of the system.

**PUBLIC**

## 2 Solution Architecture Overview

This section presents the ORCHIDE solution architectural overview, defining the main components of the system. It offers an introduction in the functionalities and components of the system, as well as the constraints that must be considered when developing the solution.

### 2.1 Functional Overview

The ORCHIDE system aims to allow a safe and secure deployment and orchestration of image processing applications within Earth Observation satellites, whatever the hardware processing resources, and whatever the hosting software execution platform.

The ORCHIDE system is designed to support diverse missions and users, offering a broad spectrum of services in space. ORCHIDE's impact extends beyond operational flexibility to include significant benefits in mission scalability and adaptability. It can support missions in Meteorology, Environmental Monitoring, Oceanography, and Observation/Reconnaissance, optimizing costs and reducing power consumption.

For detailed information, please refer to Deliverable D2.1, which outlines the different actors and ORCHIDE scenarios comprehensively.

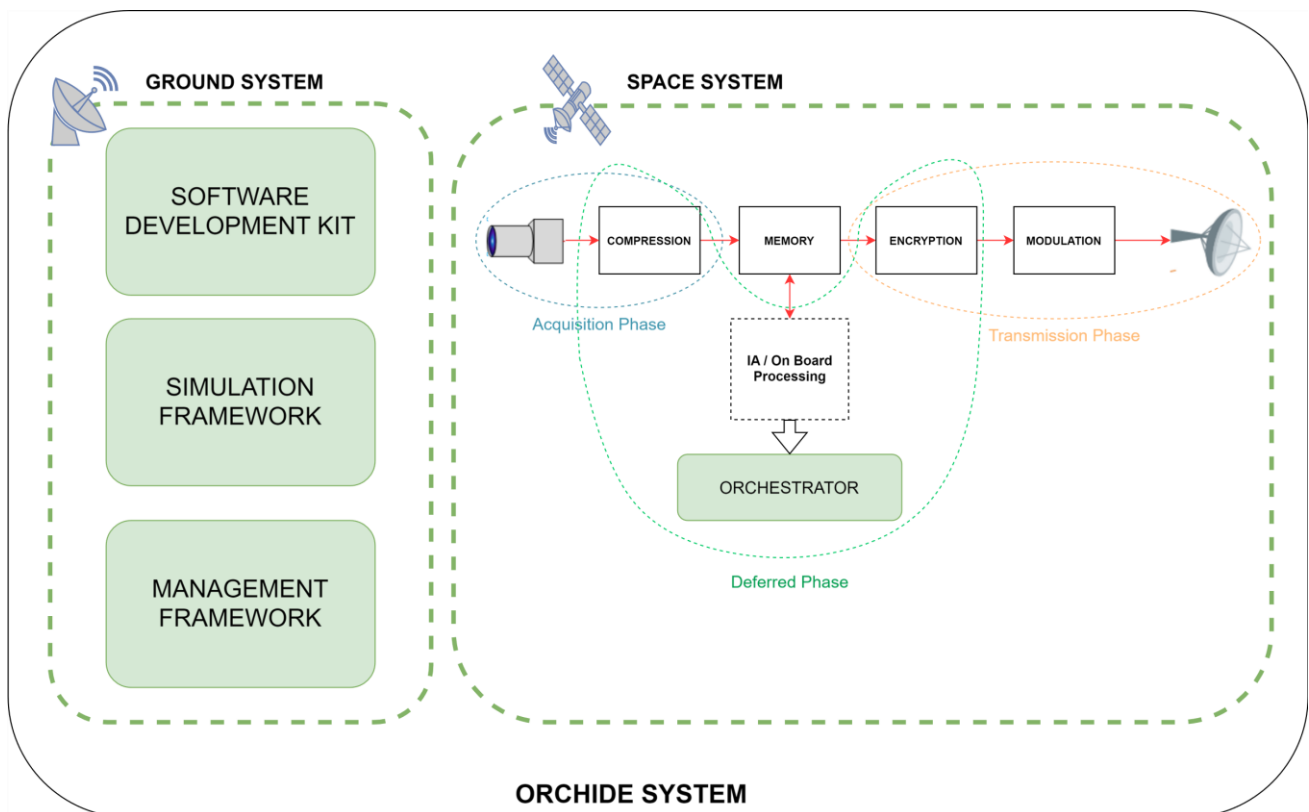
ORCHIDE serves as a software designed to manage computational tasks. Each application within ORCHIDE is customized to fulfil specific user requirements and relies on specific datasets available onboard.

The concept of an application pipeline emerges from the necessity for service providers to construct a sequence of applications. These may integrate existing software for tasks such as radiometric or geometric correction. Additional functionalities like cloud detection and conditional execution can further enhance the final application's performance within the pipeline execution.

### 2.2 System Overview

The ORCHIDE system is comprised of two distinct software deployments: Ground System and Space System. The rationale behind this division stems from the distinct manner in which various entities engage with the system. Within the Ground System, all tools and frameworks are accessible and usable by all participants, aiding in the development and deployment of applications and workflows. In contrast, the Space System, which encompasses the satellite medium, is highly secure and has limited communication windows, allowing access solely to the satellite owner and its operators. Consequently, the orchestrator, being a mission-critical component with security considerations, operates exclusively within the Space System.

**PUBLIC**



**Figure 1. ORCHIDE System Overview**

As displayed in Figure 1, the ORCHIDE Ground System includes a software development kit, a simulation framework, and essential tools for communication with the transmission mechanism to transmit and receive data, enabling monitoring and operation of the Space System.

The ORCHIDE Space System includes the Orchestrator, a distributed system for in-orbit data processing. It provides essential tools to dynamically deploy and orchestrate services, enabling efficient sharing of on-board resources and management of priorities among end-user objectives to prevent interference. Additionally, the Orchestrator can dynamically reconfigure the system, adapting to changing operational requirements and optimizing performance as needed.

The Space System has three different phases:

- Acquisition phase: capture new images or data from the satellite's sensors, compress them and prepare them to be stored.
- Deferred phase: process the data according to specific workflows and select results for transmission. The ORCHIDE orchestrator is part of this phase.
- Transmission phase: prepare and transmit the collected and processed data from the satellite to the ground station.

Out of the three stages, only the Deferred Phase is part of the ORCHIDE scope. The other phases are outside the scope of the ORCHIDE project.

## 2.3 Architecture Principles

In the following sections we will talk about the main design drivers that shape the architecture of the ORCHIDE system.

**PUBLIC**



### 2.3.1 Re-Usability

ORCHIDE is a system that is meant to be used and deployed on various contexts with various needs. Its architecture needs to leverage modularity in a way that identifies and isolated the core features for re-usability in various contexts. The architecture also needs to enforce the separation of concerns between the core concepts, the platform and integration aspects and the context-specific aspects.

### 2.3.2 Extensibility

The business vision of ORCHIDE is oriented towards establishing it as an extensible system that acts as a generic platform to orchestrate a wide range of software types on diverse hardware. The architecture of ORCHIDE is designed to maintain flexibility and evolution, enabling it to function seamlessly across various hardware and software configurations. It supports a heterogeneous multi-node onboard data centre capable of hosting diverse software platforms. The orchestration solution is designed to be technology-agnostic, ensuring independence from underlying hardware solutions. It also serves as a portable solution adaptable to different hardware and software environments.

### 2.3.3 Compatibility

The ORCHIDE system components need to be deployable on various kind of infrastructures, such as KPLabs' Leopard platform. So, the architecture of the system should rely on standardized packaging and deployment methods and tools, such as containerization and orchestration, that guarantee the minimal layer of abstraction needed to make the software components infrastructure agnostic.

### 2.3.4 Scalability

The software components should be able to upscale and downscale without service or architecture disruption, to take benefit from the full capacity of the underlying infrastructure to accommodate to the processing needs. Depending on the specific use case, the software should implement custom scaling logic within its workflows. This could involve dynamically adjusting the number of parallel tasks based on real-time conditions or external triggers.

### 2.3.5 General Constraints

The paragraphs that follow outline the constraints that need to be considered during the development of the ORCHIDE solution. These constraints encompass the development, software, and hardware specifications that were established and articulated in the preceding deliverables.

#### 2.3.5.1 Development Constraints

When developing the ORCHIDE solution as an open-source project, it is crucial to ensure compatibility with the appropriate licenses. This means that the developers need to consider the licensing requirements of the software components they use and ensure adherence to those licenses.

Additionally, it is desirable for the SDK (Software Development Kit) to be as independent as possible from specific tool vendors based outside the European Union. By making the SDK vendor-neutral, it promotes flexibility and avoids the risk of vendor lock-in. This independence fosters collaboration and innovation within the ORCHIDE ecosystem, allowing developers to choose and integrate tools that best suit their needs, regardless of the vendor's origin.

By promoting compatibility with the right licenses and maintaining a vendor-neutral SDK, the ORCHIDE solution can create an inclusive environment for developers, encouraging collaboration, and enabling the ecosystem to thrive through diverse perspectives and innovative approaches.

**2.3.5.2 Software Constraints**

The ORCHIDE solution must be compatible with the Kubernetes API. This means that the solution must provide a way to interface with the Kubernetes API and, moreover, offer an environment that allows Kubernetes based solutions to run. To accomplish this, ORCHIDE is limited to using operating systems that support Kubernetes based solutions, implicitly excluding real-time operating systems, as they do not offer this feature.

Moreover, the ORCHIDE solution must offer compatibility with at least two different unikernel solutions, and at least one containerisation solution.

**2.3.5.3 Hardware Constraints**

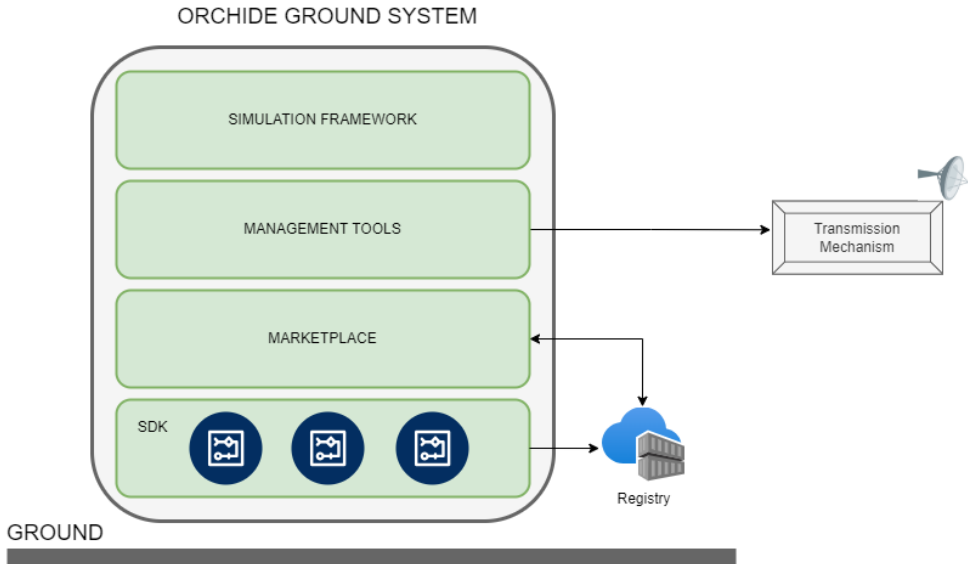
The ORCHIDE solution must be completely agnostic of the underlying hardware. The hardware-agnostic nature of the solution allows it to seamlessly function on various hardware platforms without dependencies or restrictions. This flexibility empowers organizations to choose the hardware that best suits their needs, promotes interoperability with existing infrastructure, and future-proofs the solution by adapting to evolving technology without major disruptions or costs.

Moreover, the ORCHIDE solution must leverage the existence of high-performance hardware such as FPGAs or GPUs when they are available.

**2.4 Key components and modules**

In the following sections we will detail the main components and modules of the ORCHIDE system.

**2.4.1 ORCHIDE Ground System**

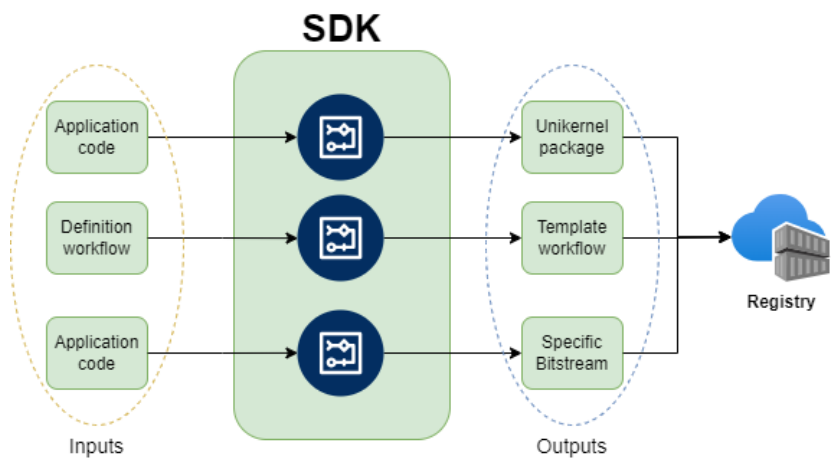


**Figure 2. ORCHIDE Ground System**

**PUBLIC**

Based on Figure 2, the application development process leverages the ORCHIDE SDK, which provides robust automation tools for constructing application packages and establishing workflows to define pipelines. This SDK not only facilitates the automation of these processes but also handles the deployment of files to a registry. Subsequently, these files are uploaded to the satellite's storage node using the Transmission Mechanism. It's important to clarify that this transmission mechanism is external to the ORCHIDE solution and is selected and overseen by the Satellite Manufacturers and Owners.

**2.4.1.1 Software Development Kit**



**Figure 3. SDK Overview**

An SDK (Software Development Kit) is a set of tools, libraries, and documentation that developers use to create software applications for a specific platform or framework. It provides a standardized set of resources and functionalities that simplify the development process and enable developers to build applications more efficiently.

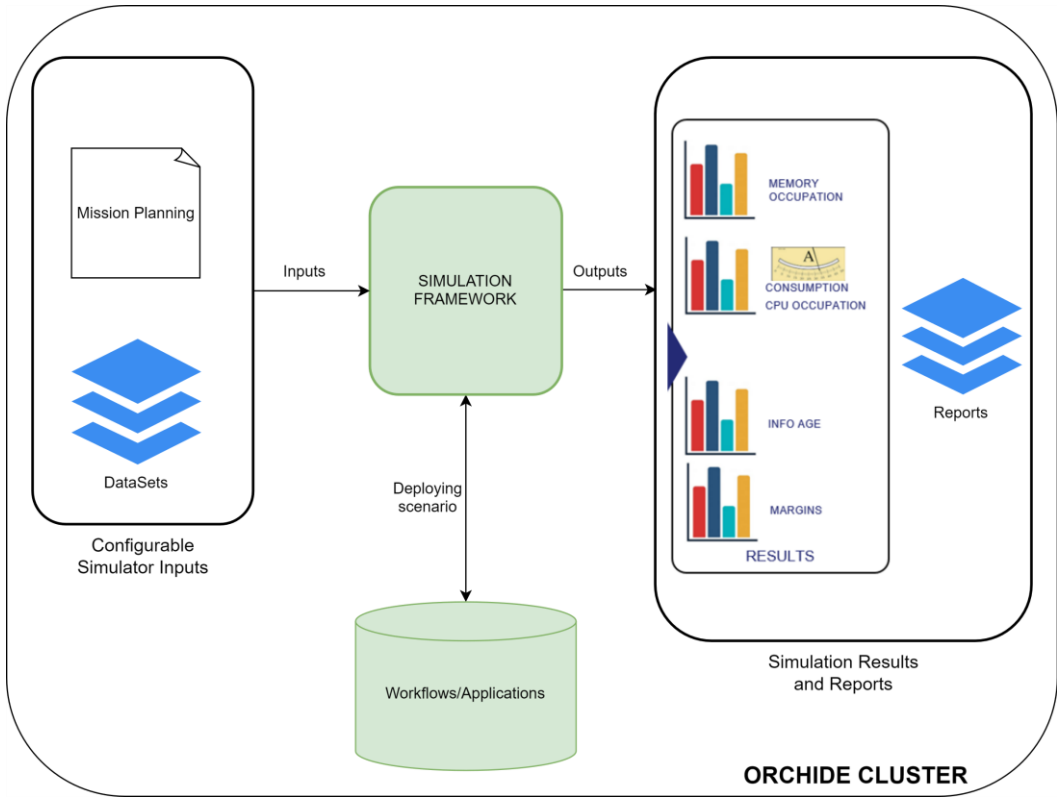
An SDK is crucial because it eliminates the need to start development from scratch, accelerating the creation of new software solutions. It offers pre-built components, ready-made code snippets, and APIs that enable developers to easily access and utilize platform-specific features and services. This accessibility helps to reduce development time, minimize errors, and promote consistency across applications built on the same platform.

The ORCHIDE SDK must provide the necessary tools for building and packaging executable applications, as shown in Figure 3, such as unikernels, bitstreams or OpenCL kernels. The SDK must support input from the application developers in a variety of different forms: various programming languages, frameworks, structures, etc. Conversely, it must strive to produce an output (packaged software) that complies with the given hardware requirements. Additionally, it must allow the definition of a workflow for a service composed of multiple executable applications.

Its architecture and functionalities are detailed in Section 5 below.

**PUBLIC**

**2.4.1.2 Simulation Framework**



**Figure 4. ORCHIDE Simulator Flow**

The simulator, displayed in Figure 4, will consider specific scenarios from each user and execute them on a simulated platform reflecting the configuration of the Edge computing infrastructure, be it a complex distributed cluster or a single compute node, and it will monitor specific parameters such as: the memory occupation, the CPU occupation, the reactivity, the margins. Modelling tools will allow to configure the emulation of the platform and to capture the user scenarios in terms of end-to-end image processing requests. The disruptive approach will be to execute directly the ORCHIDE orchestrator within the simulator, to reach a complete representativeness, and to allow the coupling of the simulator to the real systems to use it as a Digital Twin like solution. An HMI (Human-Machine Interface) will be developed or integrated, relying on web-based technologies, and will allow multiple users to remotely access the simulation results, and to interact with the simulation platform in a multi-user mode.

In the Ground Segment system, we will incorporate a representative instance of the ORCHIDE orchestrator within a dedicated test bench environment. This setup will serve as a crucial component for testing and validating various mission scenarios.

The simulator can directly invoke the different services available on the platform, akin to the real ORCHIDE orchestrator. These services are specifically designed to handle the execution and management of various workflow processes. This direct invocation capability streamlines the execution of complex mission tasks.

**2.4.1.3 Management Tools**

The management tools for the ORCHIDE satellite system will be designed to provide the following functionalities to the operator:

**PUBLIC**

- Gather and display critical metrics and logs. This framework collects on his database system parameters such as satellite hardware health, telemetry data, and performance metrics, application logs. Data is aggregated from multiple sensors and subsystems, ensuring comprehensive coverage of the satellite's operational state. These metrics and logs are then transmitted to a centralized processing unit, where specialized algorithms analyse the data for anomalies, trends, and performance benchmarks. The processed information is displayed through an intuitive dashboard interface, providing operators with a clear, view of the ORCHIDE system and the available hardware used on board.
- Deploy applications and services on-board by interfacing directly with the transmission mechanism, which can transmit new configurations, applications, services, and updates for the ORCHIDE system.
- Allow authorized users to issue commands to the space system for removing old or unneeded data from the memory, facilitating the maintenance of the system's memory health.

## 2.4.2 ORCHIDE Space System

This chapter details the role of the only component of the ORCHIDE Space System, namely the orchestrator, and the outlines that its design and development will follow.

### 2.4.2.1 Orchestrator

The Orchestrator is a robust solution for in-orbit data processing. This solution offers essential tools to dynamically orchestrate services on the available Hardware on the satellite.

The available hardware on the satellite will be considered a unified processing cluster. There are two identified hardware cases: the first case in which the main node will handle the functions of the Orchestrator, while the other hardware components will serve as processing nodes where computations will be executed, and the second case in which the main node will serve both the functions of the Orchestrator, as well as the function of a processing node. The second case is representative for the Leopard satellite platform developed and deployed by KPLabs.

The role of the Orchestrator is for managing various elements within the space system by overseeing the processing workflows, which consists of a sequence of functions distributed across processing nodes. Each processing step, such as image processing, compression, encryption, or AI tasks, is a sub-part of this workflow and is executed on one or more available hardware considered as processing nodes.

The Orchestrator shall be designed as a modular solution. It will offer the versatility to function independently as a standalone solution or offer the capability to integrate into other existing edge solutions as a building block. This design allows for greater flexibility and adaptability.

## 2.5 Architectural Views

For the ORCHIDE system, we have adopted the MBSE (Model-Based Systems Engineering) approach to model the system. MBSE employs structured modelling techniques to develop and document complex systems, encompassing various views to comprehensively describe different aspects of a system. In Deliverable D2.1, we have focused on the System view, and subsequently, we will detail the Logical and Physical views of the system. Below is presented a brief summary of the logical and physical views:

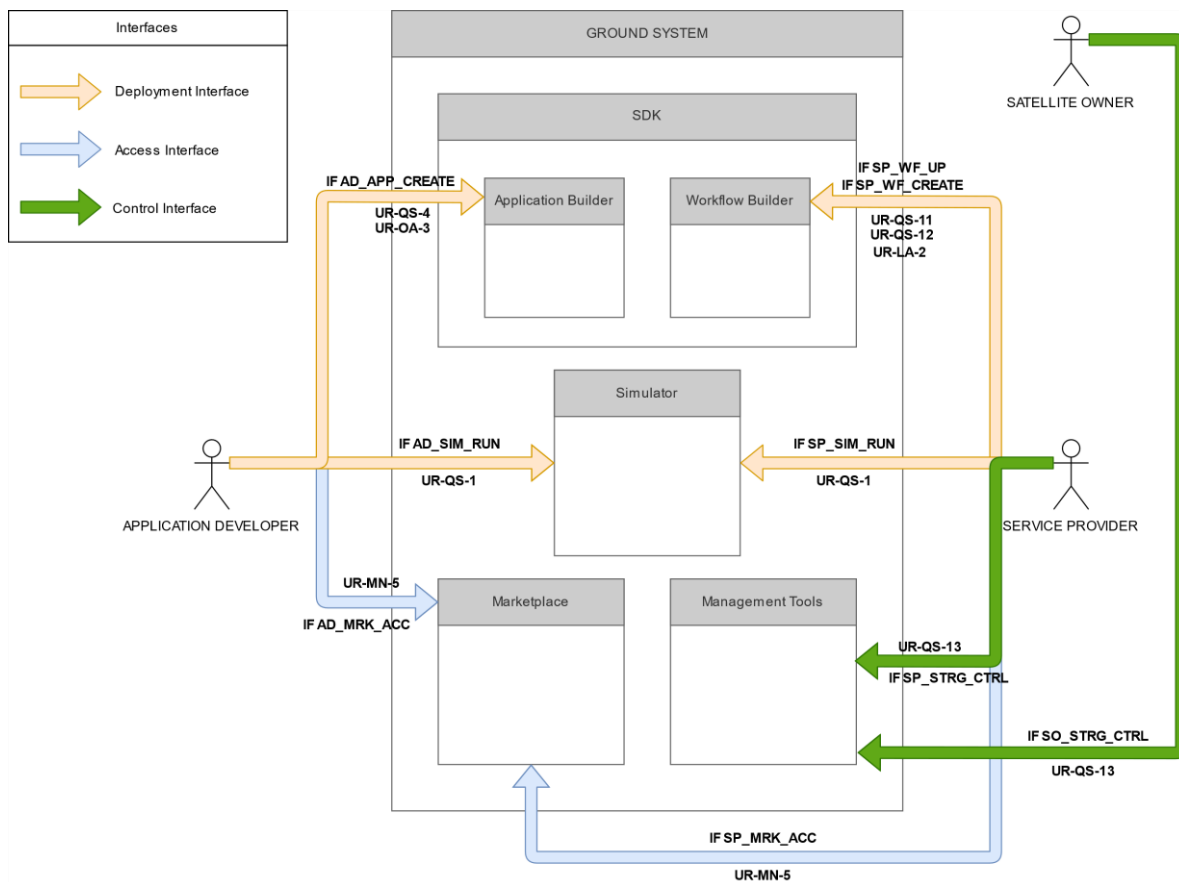
- Logical View:
  - Purpose: The logical view focuses on capturing the functional aspects and behaviour of the system without considering specific implementation details.
  - Components: It identifies and defines the system's components, their interactions, and the functional relationships between them.
  - Modelling Techniques: Use of diagrams like functional block diagrams, activity diagrams, sequence diagrams, and state diagrams to depict the system's behaviour and interactions.
- Physical View:
  - Purpose: The physical view describes the physical architecture and deployment aspects of the system, including hardware components and their interconnections.

**PUBLIC**

- Components: It details the physical components of the system such as processors, memory, sensors, actuators, communication interfaces, and their physical arrangement.
- Modelling Techniques: Use of diagrams like deployment diagrams, physical block diagrams, and network diagrams to illustrate the physical structure and connectivity of system components.

### 3 Logical View

This chapter offers details about the logical architecture of the system and an in-depth understanding of how it is organized and how its components interact with each other. It describes their relation within the system and the interfaces of communication between them and the outside system. A logical architecture chapter helps grasp the high-level design principles and patterns employed in the system, enabling readers to comprehend the system's behaviour and make informed decisions regarding its implementation, integration, and future enhancements.



**Figure 5. Ground System Logical View with Logical Interfaces**

The above Figure 5 and Figure 14 found in A.1.1 present the Logical View of the ORCHIDE system split into the two software components. The figures display the logical building blocks of each of the systems together with the interfaces that enable communication between the actors and the components, and between the two components. The interfaces are named and mapped to the corresponding requirements they serve, as presented in Deliverable D2.3.

#### 3.1 Actors

This chapter offers information about how each of the actors identified in Deliverable D2.1 will interact with the two software components at a logical level.

**PUBLIC**



### 3.1.1 Service Provider

The Service Provider is responsible for building an application pipeline to fulfil a specific service or requirement for an end user. They have the knowledge, expertise, and resources to design, develop, and deploy the necessary applications and processes within ORCHIDE.

From a logical standpoint, it is essential for the Service Provider to have the necessary capabilities to interface with the ORCHIDE system. This allows the Service Provider to deploy their own workflows and effectively manage the memory utilization of the application running under a specific workflow.

Efficient memory management plays a crucial role in ensuring the system's overall health and resource management. The Service Provider can manually identify and remove old or unnecessary data from the system's storage. This capability contributes to optimizing the system's resources and maintaining its performance.

Additionally, the Service Provider utilizes the SDK to configure or update their workflows. They can make necessary adjustments to the priorities of the different owned workflows, giving them more control over the execution and allocation of system resources based on their specific requirements.

### 3.1.2 Satellite Owner

The Satellite Owner is an entity that owns the hardware infrastructure, such as satellites, that provide processing time and capabilities to service providers within the ORCHIDE system. It is responsible for establishing agreements or contracts with the service providers for the rental of these resources.

The Satellite Owner plays a pivotal role in the ORCHIDE system, occupying a key position among the essential participants. From a logical standpoint, it is imperative for the Satellite Owner to have direct access to an interface through which they can retrieve crucial system status, logs, and metrics. These components are vital for ensuring the system's overall health and functionality.

Furthermore, the Satellite Owner requires a control interface that grants them the ability to perform in-orbit software maintenance. This capability is crucial for the system's sustained operability and adaptability, as it allows for necessary updates and adjustments to be made.

Like the Service Provider, Satellite Owner also needs control over the satellite's storage. This control enables them to maintain optimal system performance and storage efficiency by removing obsolete or unnecessary data, ensuring that resources are utilized effectively.

Additionally, Satellite Owner leverages the SDK to configure or update their workflows. This grants them the flexibility to make necessary adjustments to workflow priorities, providing finer control over system resource allocation and execution based on their specific requirements.

### 3.1.3 Application Developer

The Application Developer is in charge of coding and developing the processing application that will be deployed within the ORCHIDE system. The key responsibilities of the Application Developer include application development, testing, simulation, integration and documentation.

The Application Developer is responsible for creating and implementing applications in the system using the SDK. They utilize the SDK's tools to build and deploy their containerized applications, following the workflows provided by the Service Provider.

Moreover, the Ground System, through its Simulator component, offers the Application Developer a way to test the behaviour and compatibility of its applications in a Digital Twin-like environment, resembling the real ORCHIDE Satellite Solution.

**PUBLIC**



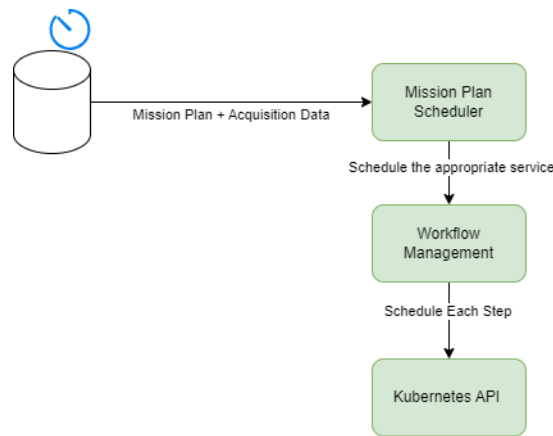
## 3.2 Logical Components of the Architecture

This chapter offers an in-depth view of each of the logical components that form the ORCHIDE solution, describing the purpose of each component of the system at a logical level.

### 3.2.1 Orchestrator

The ORCHIDE Orchestrator is one of the two software deployments of the system. Figure 14 displays the logical architecture of the Orchestrator and the external logical interfaces. The sections that follow describe each of its logical components in more details.

#### 3.2.1.1 Mission Manager



**Figure 6. Mission Manager Overview**

The Mission Manager, displayed in Figure 6 above, plays a vital role in ensuring that the ORCHIDE system follows the mission plan provided by the Satellite Owner. By offering a control logical interface, as displayed in Figure 14, it allows the Satellite Owner to access and reconfigure the system, enabling in-orbit software maintenance. Once the mission plan and acquisition data are received, the Mission Manager transfers them to the Mission Scheduler. The Mission Scheduler then utilizes the mission plan to schedule the necessary services, which are subsequently sent to the Workflow Manager for execution.

##### 3.2.1.1.1 Workflow Manager

This component is the core of the pipeline workflow ORCHIDE aims for. The Workflow Manager will have to manage the different stages, as well as the parallelism aspect of the application pipelines based on the given configuration. The Workflow Manager must offer configuration options, that the Service Provider can configure through the SDK. These configurations must include the number of stages of an application pipeline, the parallel workflow in case it is needed by an application, and the priority of the pipeline.

##### 3.2.1.1.2 Mission Planner

The Mission Planner, as a subcomponent of the Mission Manager, has the responsibility of determining the appropriate service to be initiated based on the mission plan and the acquired data from various onboard instruments. It evaluates the conditions required to trigger each service, considering both the starting conditions and the priority assigned to each service within the mission plan. This prioritization plays a role in distinguishing between services that share the same starting conditions. By considering these factors, the Mission Planner ensures a well-informed and efficient selection of services, tailored to the specific requirements outlined in the mission plan.



### 3.2.1.1.3 Communication Manager

The Communication Manager is responsible for the selection and preparation of data to be transmitted to the ground station. It performs the task of scheduling the delivery of data based on priorities set by the Service Provider and agreed upon with the End User. By employing a scheduler, the Communication Manager determines the order and selection of results to be transmitted, always considering the designated priorities. This ensures efficient and organized data transmission, aligning with the agreed-upon preferences and optimizing the overall communication process.

### 3.2.1.2 Storage Manager

The Storage Manager is a component of the orchestrator, responsible for managing the system's accessible memory and the applications. Its primary role is to store the outputs generated by the application pipelines, whether it's the output of a single stage or the final output.

Concurrently, the Storage Manager ensures efficient storage access and handles any potential concurrency issues that might arise when dealing with parallel pipelines, such as storage access arbitration. Additionally, it provides the capability for manual removal of data from the storage, facilitating better storage management by allowing older or unneeded data to be cleared.

### 3.2.1.3 Execution Platform

The Execution Platform is the logical component in charge of the execution of the service stages, from planning the stage that was scheduled, to launching and managing the lifetime of the deployed applications. The Execution Platform connects the two logical components under it, the Stage Planner and the Runtime, to assure they can interface with each other, thus forming a complete execution workflow.

Moreover, the Execution Platform connects to the Security Manager to assure that the applications it runs are verified and have passed all the needed checks before being launched.

#### 3.2.1.3.1 Stage Planner

The planner has the responsibility to manage and oversee the scheduling of jobs within the system. It guarantees the execution of applications according to predefined schedules or triggers. Serving as a central coordinator, it aids in optimizing the allocation of computing resources and maximizing the efficient utilization of available resources. The planner plays a pivotal role in the system as it handles the initiation of the execution for various applications and facilitates the seamless context switching between them.

#### 3.2.1.3.2 Runtime

The responsibility of launching and managing the lifecycle of applications deployed in the ORCHIDE system falls upon the Runtime. Positioned under the Execution Platform and connected to the Stage Planner, this component plays a crucial role. When an application is scheduled to start, the Runtime is responsible for initiating its launch and overseeing its execution in alignment with the workflow defined by the Workflow Manager.

### 3.2.1.4 Security Manager

The Security Manager is the component that handles the security of the ORCHIDE satellite system. It is tasked with handling the main security features of the system such as encryption and decryption of application data, and authentication of the application and services that are deployed.

The encryption of data is an important aspect of the system, as data belonging to different entities will be stored in the same shared memory, so ORCHIDE must provide a way to assure the end users that their deployments and results are safer from any malicious attacks that can be launched from inside the system, compared to having all the data stored unencrypted.

**PUBLIC**

Moreover, the Security Manager must ensure that the deployed services and applications are verified beforehand and that they are matching their signatures. By verifying and authenticating the software that is deployed on the system, ORCHIDE can provide a high degree of trust for the users and a safe environment for all the various services.

### 3.2.1.5 Monitor Manager

The Monitor Management component is for collecting the system status of the ORCHIDE system, as well as gathering important metrics and system-generated logs. It serves an essential role in ensuring the system's operational visibility.

To facilitate the active monitoring of the ORCHIDE system, the Monitor Management component provides the Satellite Owner with the means to access and retrieve this essential data. This access enables the Satellite Owner to stay informed about the system performance and take any necessary actions to maintain its functionality and efficiency.

## 3.2.2 Ground System

The Ground System is the second software deployment of the system. It contains the SDK, the Management Tools, the Simulator, and the Marketplace. These services offer the users ways to build their applications into binaries that fit the requirements of the ORCHIDE system, enables them to test their applications and workflows on a platform that is like the real target environment, and provides access to a platform to store their binaries and interact with other users.

### 3.2.2.1 SDK

#### 3.2.2.1.1 Workflow Builder

The Workflow Builder is a component within the SDK, providing users with the ability to create configuration files that are subsequently utilized by the Workflow Manager. These configuration files enable users to specify and update the workflow settings necessary for their applications.

As a part of the SDK, the Workflow Generator is the sole avenue by which users can configure and modify the workflow requirements of their applications. Its functionality empowers users to tailor the workflow to their specific needs and facilitates seamless updates as required.

#### 3.2.2.1.2 Application Builder

This SDK component has the responsibility of providing users with the capability to build applications as binary packages, such as MirageOS and Unikraft. It also facilitates the deployment of bitstreams to configure the FPGAs and the deployment of OpenCL kernels. Developers can write applications in any of the supported programming languages provided by the packaging solutions. Additionally, the Application Builder offers integration with popular AI frameworks like TensorFlow and others, allowing developers to seamlessly utilize these frameworks in their applications.

### 3.2.2.2 Management Tools

The Management Tools provide Satellite Owners and Service Providers with the capability to access, and update deployed applications and workflows. Satellite Owners can monitor and optimize performance, while Service Providers can customize applications to meet customer needs. These tools enable proactive management, efficient updates, and tailored services for all stakeholders involved.

Moreover, the Management Tools handle the deployment of workflows and applications on-board the satellite system through the designated logical interfaces. In addition to these deployment interfaces, it offers a control interface through which the Service Provider and the Satellite Owner can manage the storage, removing unneeded or outdated data, keeping the memory healthy.

**PUBLIC**

### 3.2.2.3 Simulator

The Simulator is a software deployment on the Ground System that offers a simulated environment of the system to enable the developers of applications and services to test the behaviour and compatibility of their software inside an isolated environment that simulates the real one. This way they can ensure that the applications and services tested using the simulator will not have any compatibility issues when deployed on the real system.

### 3.2.2.4 Marketplace

The Marketplace provides the users of the ORCHIDE system a place to store their binaries, be it applications in the form of unikernels, FPGA bitstreams, or OpenCL kernels, and workflows. Additionally, the Marketplace facilitates the communication between different users by creating a medium in which one can post requests for software solutions for the ORCHIDE system, and the other users can respond by picking up the request.

## 3.3 Description of the Logical Interfaces

The direction of the logical interfaces shows the direction that data is going to. The table below displays the abbreviations used in the sections that follow.

Term	Abbreviation
Application	APP
Application Developer	AD
Access	ACC
Collection	COL
Control	CTRL
Deploy	DP
Management Tools	MT
Marketplace	MRK
Mission Plan	MIS
Satellite Owner	SO
Security	SEC
Service Provider	SP
Simulation	SIM
Status	ST
Storage	STRG
System	SYS

**PUBLIC**

Update	UP
Workflow	WF

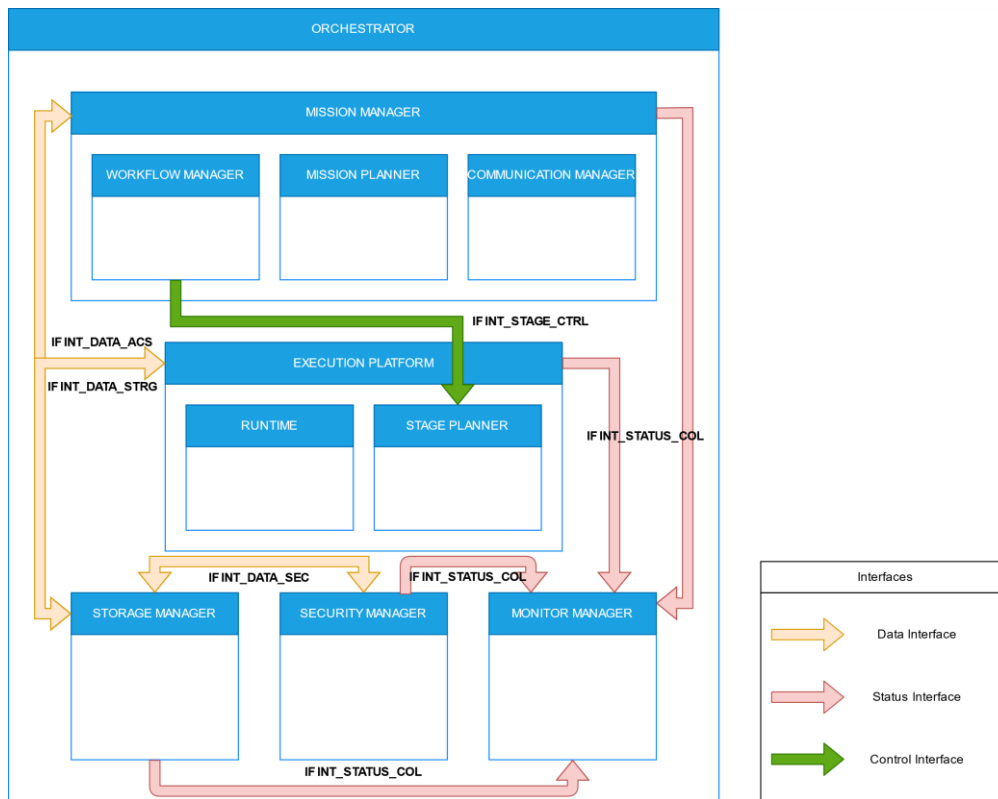
### 3.3.1 Orchestrator External Logical Interfaces

As identified in Figure 14 found in A.1.1, the table below describes each of the external interfaces of the orchestrator and the mapping with the user requirements specified in Section 2.2 of Deliverable D2.3.

Interface	User Requirement	Description
IF MT_WF_DP	UR-QS-11	An interface used for deploying the workflow to the ORCHIDE system. It connects the Management Tools to the Workflow Manager via Mission Manager at the logical level.
IF MT_WF_UP	UR-QS-12	An interface used to update the workflows. It connects the Management Tools to the Workflow Manager via Mission Manager at the logical level.
IF MT_APP_DP	UR-QS-4	An interface used for deploying an application into the workflow. It connects the Management Tools to the Mission Manager at the logical level.
IF MT_STRG_CTRL	UR-QS-13	An interface offering control over the storage of the system to the Service Provider and the Satellite Owner via the Management Tools to remove unneeded or old data. The interface connects the Management Tools and the Storage Manager at a logical level.
IF SO_SYS_CTRL	UR-MN-6	This is a control interface that allows the Satellite Owner to ensure the continued operability and adaptability of the system by providing means for in-orbit software maintenance. This interface lets the Satellite Owner to directly manage and change the Mission Manager
IF SO_ST_COL	UR-MN-8	An interface used for collecting the status of the system by the Satellite Owner. This allows the Satellite Owner to ensure the continuous monitoring of the system health. The interface connects the Satellite Operator and the Monitor Management at a logical level.
IF SO_MIS_DP	UR-LA-2	An interface used for deploying the mission plan. The interface connects the Satellite Owner to the Mission Manager. This interface must be separated from the others from the Ground System as the mission plan is a critical part of the overall deployment and only the Satellite Owner must have access to it.

**PUBLIC**

### 3.3.2 Orchestrator Internal Logical Interfaces



**Figure 7. Orchestrator Internal Logical Interfaces**

Figure 7 above displays the internal interfaces of the orchestrator component. These interfaces represent data links between different logical components, status collection interfaces, and a control interface. The table below describes each interface in detail.

Interface	Description
IF INT_DATA_ACS	This interface represents a logical communication channel between the Storage Manager and both Mission Manager and Execution Platform. The latter two components require access to read data necessary for the services and applications they execute. Moreover, the Mission Manager needs access to the mission plan that is stored and handled by the Storage Manager.
IF INT_DATA_STRG	This interface is an extension of the IF INT_DATA_ACS interface, offering the capability to store the results and logs into the memory via the Storage Manager.
IF INT_DATA_SEC	A requirement of the ORCHIDE solution is to offer increased security for the different data that is placed and produce on-board the satellite. To accomplish this, the need of a logical interface between the Storage Manager that handles the memory of the system, and the Security Manager is a must. This interface will be used for encryption and decryption of the data.
IF INT_STATUS_COL	This interface connects all the components of the Orchestrator to the Monitor Manager. The Monitor Manager is in charge of collecting and overviewing the status and health of the different components of the system, so all components of the system must have a way to report their status to it.

**PUBLIC**

IF INT_STAGE_CTRL	A control interface between the Workflow Manager and the Stage Planner is a necessity as the Workflow Manager needs to signal the Stage Planner when a new service has been selected for running and what stages need to be planned.
-------------------	--

### 3.3.3 Ground System Logical Interfaces

The below table showcases the logical interfaces between the outside actors and the Ground System with its components, as exemplified in Figure 5. Each interface corresponds to one or more user requirements described in Section 2.2 of Deliverable D2.3.

Interface	User Requirement	Description
IF AD_APP_CREATE	UR-QS-4 UR-OA-3	An interface between the Application Developer and the Application Builder subcomponent of the SDK. The Application Developer requires access to the tools used for building the ORCHIDE application binaries in the form of adaptable packages.
IF AD_SIM_RUN	UR-QS-1	An interface between the Application Developer and the Simulator. It is a necessity for the Application Developer to have an interface through which he can deploy and run applications on the ORCHIDE simulator to assure their compatibility and behaviour with and within the system.
IF AD_MRK_ACC	UR-MN-5	An interface between the Application Developer and the Marketplace. The Application Developer must have an interface that provides access to the Marketplace environment so that he can load binaries of created applications and see the available requests for applications.
IF SO_STRG_CTRL	UR-QS-13	A control interface between the Satellite Operator and the Management Tools. Through this interface, the Satellite Operator can make use of the offered tools to manage the storage and remove any unneeded or old data.
IF SP_WF_CREATE	UR-QS-11	An interface between the Service Provider and the Workflow Builder subcomponent of the SDK. The Service Provider requires access to the workflow tools offered by the SDK to create and customize the workflows.
IF SP_WF_UP	UR-QS-12 UR-LA-2	An interface between the Service Provider and the Workflow Builder subcomponent of the SDK that enables usage of the offered tools to update a workflow and its priority to keep up with the dynamic evolution of the user needs.
IF SP_SIM_RUN	UR-QS-1	An interface between the Service Provider and the Simulator that is essential for testing the deployment and behaviour of the workflows before deploying them onto the real system.

**PUBLIC**



IF SP_STRG_CTRL	UR-QS-13	A control interface that connects the Service Provider and the Management Tools. Through it, the Service Provider can leverage the offered tools and manage the storage of the system to remove old or unnecessary data.
IF SP_MRK_ACC	UR-MN-5	An interface between the Service Provider and the Marketplace. The Service Provider must have access to the Marketplace to publish his workflows and to check for new requests.

## 4 Physical Architecture of the System

The physical architecture of the system is shown in Figure 15 found in A.1.1. It displays the four main components that constitute the physical system. The components are: ORCHIDE Ground System, Transmission System, ORCHIDE Orchestrator, Acquisition Equipment.

In the next section we describe each of the enumerated physical components that are part of the ORCHIDE scope.

### 4.1 Components General Description

The components are visually described in Figure 15, together with how they interconnect with one another and the flow of data between them. Out of the four physical components identified, only the ORCHIDE Ground System and the ORCHIDE Orchestrator are in the scope of our project. The Transmission System and the Acquisition Equipment, although critical to our system, are not part of the project and there are already existing solutions with which the system will interconnect with. In the following sections we will describe the two physical components in the scope of our project.

#### 4.1.1 ORCHIDE Ground System

The ORCHIDE Ground System physical component represent the equivalent of the logical architecture's Ground System. It is split into four behaviour physical components: the SDK, the Simulator, the Marketplace and the Management Tools. These are a behaviour representation of the logical components with the same name described in Section 3.2 above.

The SDK offers the options to build applications and workflows and to update existing workflows. The binaries generated by the building tools will be stored on the Marketplace on the Ground System. The Management Tools will get the binaries as input from the Marketplace and deploy them via the Transmission Mechanism to the ORCHIDE Orchestrator. In addition to deploying binaries, the Management Tools can be used by authorised users to issue commands to the orchestrator. The results of the orchestrator, as well as the status will be collected by the Management Tools as well.

Another component of the ORCHIDE Ground System is the Simulator, and it offers the capability to simulate a workflow or application on the Ground System in the same conditions as it would run on the ORCHIDE Orchestrator, employing a Digital Twin like approach.

#### 4.1.2 ORCHIDE Orchestrator

The ORCHIDE Orchestrator physical component is composed of three behaviour physical components: Cluster Node, Storage Node, and Communication Node.

**PUBLIC**

The Cluster Node component is an abstraction above the Mission Manager, the Execution Platform, and the Monitor Manager logical components. It will handle all the main processing of the Orchestrator, such as reading and interpreting the mission plan, configuring all the services based on the mission plan, scheduling and launching services, scheduling and running the stages and application, and collecting metrics and logs.

The Storage Node describes the behaviour of the Storage Manager logical component of the orchestrator. It stores all the data that is sent by the Ground System and the data generated by the Acquisition Equipment. It must also offer an option to remove data from memory based on the received commands.

Lastly, the Communication Node is the component in charge of handling the transmission of data out of the orchestrator to the satellite communication interface. The logical component mapped to it is the Communication Manager, as so, it must implement an algorithm for selection of data based on the priorities of the different services given in the mission plan.

## 4.2 Orchestrator Physical Architecture Overview

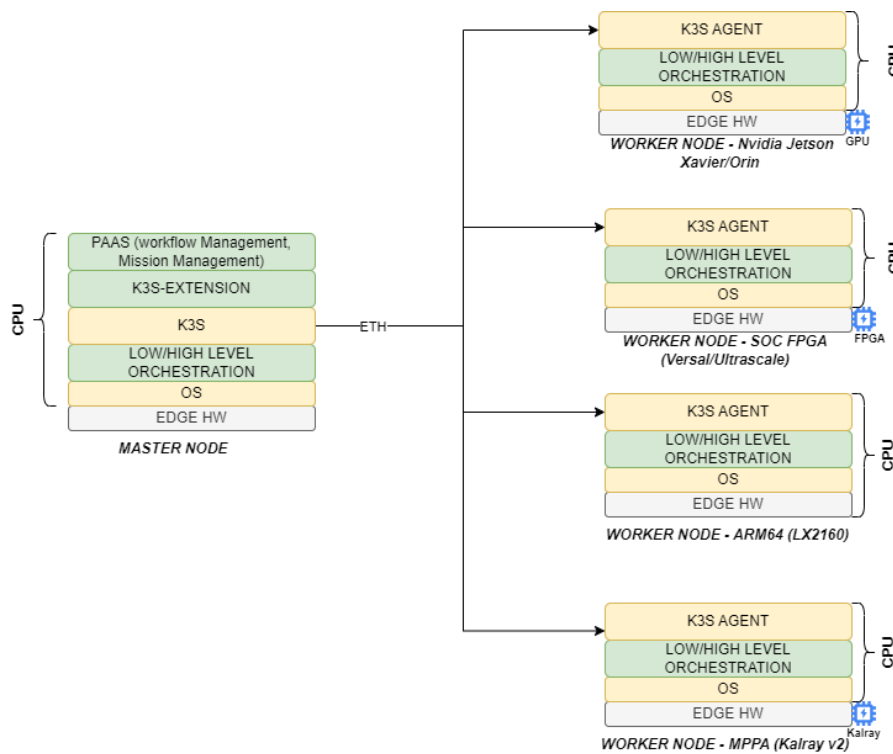


Figure 8. ORCHIDE Orchestrator On-Board Physical Architecture Overview

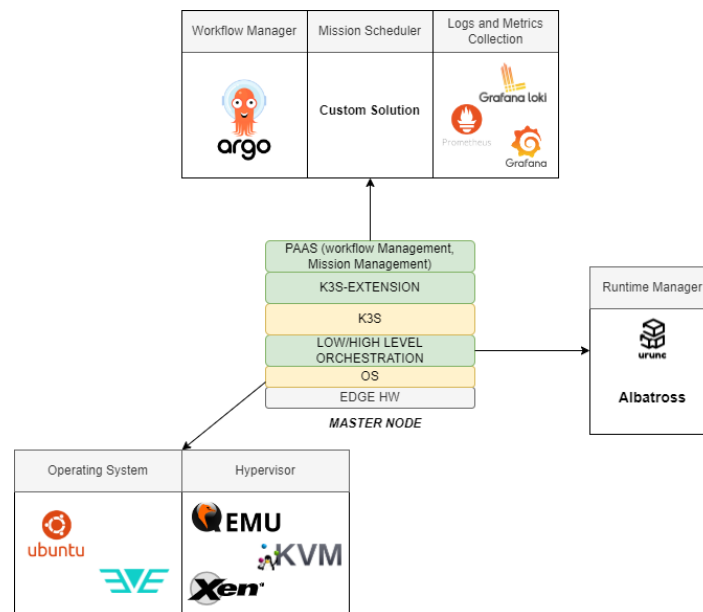
Within the ORCHIDE orchestration solution, the architecture is divided into two main components: the master node and the worker nodes.

The master node, displayed in Figure 8, is structured as a composition of various layers. These layers are:

- EDGE HARDWARE (HW):** The orchestration solution operates without knowledge of the specific underlying hardware used in the satellite. It is designed to work with the understanding that all necessary interfaces, protocol stacks, and drivers are accessible and available for its utilization. The system relies on the information provided within the mission plan, which outlines the location and allocation of these resources. By leveraging this information, the orchestration solution can effectively map and utilize the available resources, ensuring efficient coordination and deployment of services within the satellite system.



- **OPERATING SYSTEM (OS):** The operating system facilitates the access and mapping of hardware resources within the orchestration solution. It enables the orchestration applications to effectively utilize these resources and manage the workflow between them. By providing the necessary support, the operating system empowers the orchestration applications to interact with the hardware, ensuring coordination and efficient utilization of resources within the system.
- **LOW/HIGH LEVEL ORCHESTRATION:** This layer represents the application runtimes within the ORCHIDE system. The applications that operate on this system are designed and deployed as unikernels, which are lightweight virtual machines (VMs) with specific runtime requirements. These unique runtimes are essential for running and executing the applications effectively. By incorporating this layer into the system architecture, ORCHIDE ensures that the applications within its ecosystem can run efficiently, leveraging the optimized and specialized runtime environments provided for the unikernels.
- **K3S:** As defined in Deliverable D3.2, the cluster orchestration solution selected for the ORCHIDE system will be based on K3S. K3S is a lightweight distribution of Kubernetes that is designed for resource-constrained environments or scenarios where a smaller footprint is desired. Despite its reduced size, K3S maintains most of the core functionalities and APIs of standard Kubernetes, making it an efficient choice for our purposes.
- **K3S-EXTENSION:** This layer consists of custom-developed glue code, built upon the K3S platform, to ensure compatibility and facilitate the deployment of the applications that form the orchestration solution as a whole. By leveraging the K3S platform, the custom-developed glue code ensures integration and efficient deployment of the applications, ultimately contributing to the successful orchestration of the overall solution.
- **PAAS (Platform as a service):** This layer encompasses the suite of services we provide to the users of the system. These services offer a range of functionalities: access to the Workflow Manager for configuring workflows, the Mission Scheduler which can be configured through the mission plan, and the tools utilized for logging and metric collection. By offering these services, we aim to empower users in effectively managing their workflows, scheduling missions, and gathering important logs and metrics for system monitoring and analysis.



**Figure 9. ORCHIDE Master Node Solutions**










In accordance with Figure 9, we have selected existing solutions and the need for custom solutions across the different layers comprising the master node. The ORCHIDE project aims to reuse as many open-source solutions as it is possible to reduce the time spent on its development. The above solutions, as well as the ones presented in Figure 10 below, were selected after an analysis, documenting and filtering the existent solutions that fit the needs of the project and.

**PUBLIC**

For the worker nodes, four distinct configurations have been identified based on the various hardware platforms that will be utilized during the demonstration. This diversity in computing hardware serves to showcase ORCHIDE's ability to perform seamlessly across different underlying hardware platforms. It also allows us to explore and address various scenarios that may arise with these configurations. The solutions implemented for each layer within the worker nodes are akin to those identified for the master node. On the worker nodes, a K3S instance will be launched, which will then receive and execute commands from the master node, enabling effective coordination and execution of tasks within the ORCHIDE system.

The communication between the master node and the worker nodes will adhere to the Ethernet protocol, with the ORCHIDE system relying on the underlying connections being established and operational. The data exchange between the nodes will be encapsulated within Ethernet frames, adhering to the standards and protocols defined for Ethernet communication. Effectively utilizing this communication mechanism, ORCHIDE will ensure reliable transmission of data between the master node and the worker nodes, ultimately facilitating efficient coordination and execution of tasks within the system.

Figure 10 below showcases the possible solutions for the different components that we have identified so far. The solutions will be further explored and detailed in the future deliverables.

Workflow Manager	Mission Planner	Monitor Manager	Communication Manager
	Custom Solution		Custom Solution
Operating System	Hypervisor	Runtime	
 	  	  Albatross	
Stage Planner	Storage Manager	Security Manager	
	Custom Solution with COTS	Custom Solution with COTS	

**Figure 10. Orchestrator Logical Blocks Possible Solutions**

As shown in Figure 10 above, there are components for which identified available solutions that fit the project's needs and requirements have been identified, while for other components, like the Mission Planner or the Security Manager, custom solutions will have to be developed. These custom solutions can be fully developed from scratch or can be based on already existing solutions.

**PUBLIC**

## 5 SDK

As pointed in Section 2.4.1.1 and 3.2.1.1, the SDK is a set of tools, libraries, templates and documentation separated into multiple components that ease the job of both application developers and service providers to create applications and workflows that are compatible with the ORCHIDE solution runtime. We focus on a few of the most important ones in the following sub-sections.

### 5.1 Application Builder

One of the main components of the SDK is its application builder, which has the role of compiling and packaging application code and requirements into unikernels compatible with ORCHIDE and/or bitstreams for specialized hardware.

As established in the user requirements, ORCHIDE is to support unikernels as its main of packaging and running applications (for the reasons outlined in the state-of-the-art document). More specifically, both MirageOS and Unikraft need to be supported: the former for a focus on secure and lean applications, and the latter for performant and easy-to-port applications.

The general workflow of the builder is to:

1. Allow developers to express the dependencies and targets of their application
2. Automatically build the application as unikernels
3. Package it for ORCHIDE

#### 5.1.1 Dependency expression

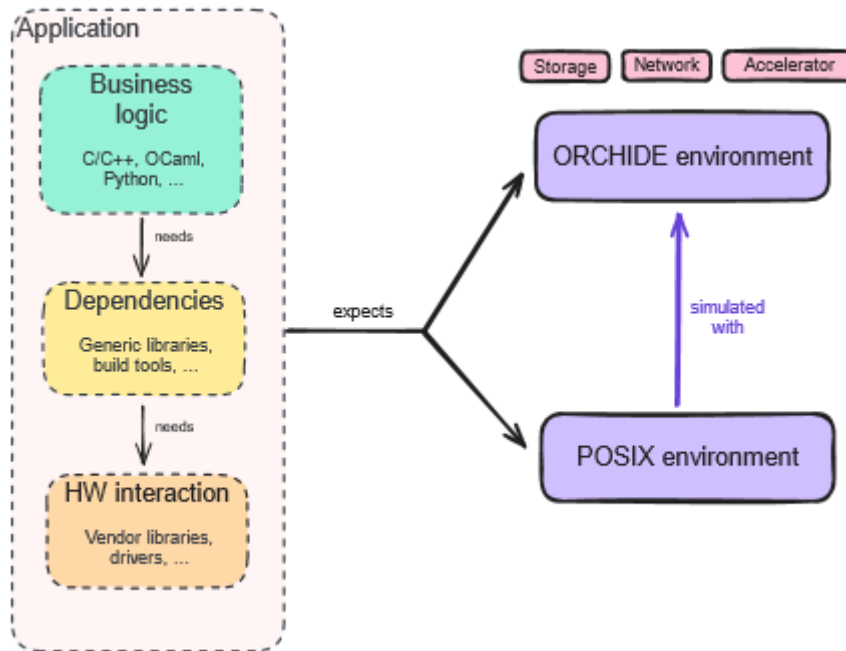
As illustrated on the left of Figure 11, application code is composed of various heterogeneous pieces that can be classified as either Business Logic (actual application logic, written in an arbitrary programming language), Dependencies (generic libraries and/or build tools from the ecosystem of the business logic's programming language) and Hardware Interaction (vendor libraries for specific hardware, drivers, system libraries).

The main difficulty of the SDK to obtain a final unikernel from these components is to understand how to hook up the final ORCHIDE on-board environment (storage, network, accelerated hardware, etc.) to the internal interfaces that the original application expects. Hence the first step is to understand these expectations, by both automatically analysing the structure of the project and interacting with the user.

The later interaction is covered by interface IF AD\_APP\_CREATE mentioned in Section 3.3.3, and together with automatic analysis must gather at least the following information about the application

- **I/O conventions:** usual applications expect a POSIX interface where both application code (through e.g. the C standard library) and low-level hardware control libraries expect available POSIX system calls to communicate with both the operating system (to get services such as filesystems, networking, etc.) and specific drivers. In the unikernel world, a POSIX interface is not a given. In particular, applications developed for MirageOS cannot expect such an interface, and must be designed with explicit conventions to interact with the outside world (we'll refer to these conventions as the "ORCHIDE environment"). Conversely, Unikraft supports a large surface of the POSIX interface (but not all of it). Hence, with the user's help, the SDK must identify what kind of application they are trying to build: POSIX or ORCHIDE, Unikraft or MirageOS.
- **Programming language:** both the business logic and dependencies are written in programming languages that must be supported as runtimes in either MirageOS or Unikraft.
- **Hardware target:** while some default choices can be made automatically, the SDK must determine for what hardware it is building. In particular, the CPU architecture expected to be supported (so a unikernel is built for each) and most importantly: if accelerated hardware (FPGA, GPU, etc.) is expected to be usable for running part of the application (and if so, which ones). In that case, the SDK must provide a clear way for the developer to scope parts of its application that should be buildable into bitstreams, OpenCL kernels, etc.

**PUBLIC**



**Figure 11. Application components and external interfaces**

To gather this information from the application developer, the SDK will use both conventions established in advance and documented for the developer (e.g. how to structure a project, template projects for different programming languages, configuration files, etc.) and an explicit Human-Machine Interface (HMI).

Figure 12 presents how applications are executed onboard: a dispatch order is received and forwarded by the K3S Agent, and the lifecycle manager must determine what virtual machine monitor to choose, and how to parametrize it so that the application is hooked to the (virtualized) environment as it expects. This shows that both the SDK and the runtime agents must be developed together, so that I/O conventions are correctly enforced by the SDK and applied at runtime.

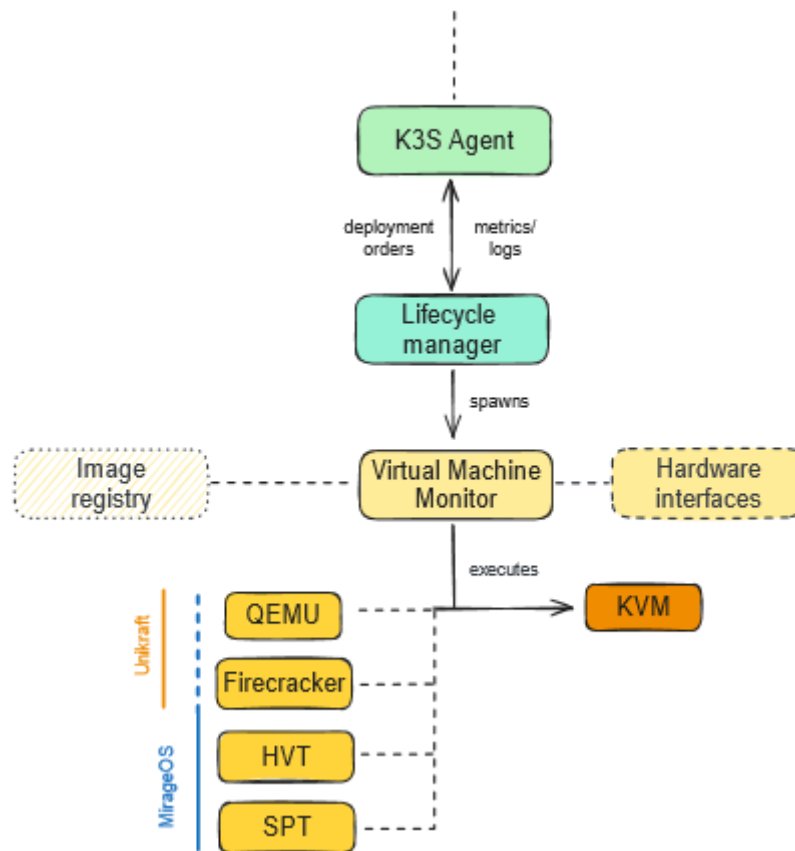


Figure 12. Runtime deployment of applications

### 5.1.2 Automatic build

Based on the information gathered beforehand, the SDK allows developers to build their application to obtain intermediate build artefacts (before packaging at the next step). It is expected that this build step should be available locally for developers to iterate on their design, as well as in a hosted environment controlled by the service provider so the output can immediately be packaged and stored in the registry. Hence, the interaction with the build system will be both from the same HMI as last step, as well as options for integration within CI/CD environments.

The build itself can be entirely automatic if enough information has been gathered beforehand, and consists in:

1. Selecting the unikernel toolchain (MirageOS or Unikraft)
2. Configuring its environment (architecture, platform, enabled kernel features)
3. Retrieving and building the application dependencies (e.g. fetching Python packages)
4. Injecting code for hardware interaction (if the application expects a POSIX interface, the calls must be translated to match ORCHIDE conventions; accelerated hardware interfacing must be set up with vAccel (see state of the art))
5. Building and creating the unikernel binary

The artefacts (unikernels, bitstreams, configuration data, secrets, etc.) are then ready to be packaged.

### 5.1.3 Packaging for ORCHIDE

After the build step, the SDK can seamlessly package the intermediate artefacts into an ORCHIDE application format. That format is expected to be self-contained, so that a registered package doesn't need any external dependencies to be dispatched and executed on board of a satellite.

**PUBLIC**

The goal is to use, as much as possible, standard and existing formats for this step. Notably, the OCI Image Format is a good candidate, as it allows to store both the binary, but also file system information and other metadata, in practice enforcing conventions around I/O. To answer user requirements, the resulting package should be hardware-agnostic, as much as possible. For example, the package format could be multi-architecture, embedding artefacts for all possible platforms.

Additionally, the application format must support metadata that will be useful for both developers and service providers to effectively communicate about these applications: versioning data, metadata enabling incremental updates, metadata specifying potential dependencies on (or advertising support of) accelerated hardware.

The SDK, both in its HMI and command-line tools, must offer an option to upload the resulting packages to the application registry of the service provider. Independently of registering, the SDK must as well allow to quickly validate that the application has been correctly built and packaged, by testing that it boots correctly within the supported environments (but the actual feature testing is left to the developer).

## 5.2 Workflow Builder

While the application builder described in last section allows for efficiently building and packaging single applications, a complete service is a workflow staging multiple applications. Hence, an orthogonal role of the SDK is to allow a service provider to easily express a service plan as ORCHIDE-compatible workflows, referring to applications built via that same SDK.

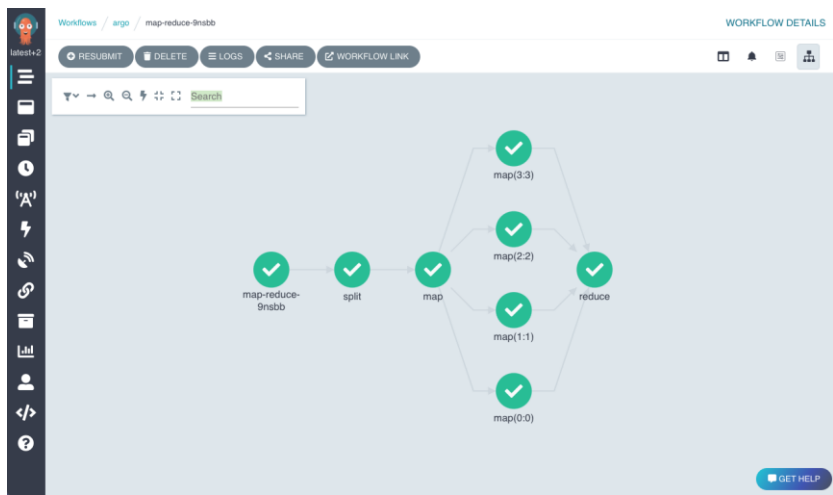


Figure 13. Example visual structure of a workflow

That interface (designated as IF SP\_WF\_CREATE in earlier sections) must allow easy description of pipelines (steps, conditions, output gathering), as well as composition of thereof, to allow ease of reuse as well as parallelization of tasks. It is expected that the service provider can manipulate the workflow in a visual way via a graph editor, allowing specification of further metadata that maps to the chosen workflow manager (such as Argo), as illustrated in Figure 13.

The output of this builder, which depends on the choice of workflow manager, consists in complete configuration files compatible with the workflow manager (e.g. YAML files) that can be executed on ORCHIDE and triggered by a mission plan. Together with the application registry from which it will fetch application packages, this constitutes the whole output of the SDK that is uploaded on board of a satellite for it to function autonomously.

While ORCHIDE sets some conventions for communication between the application and the environment (as seen previously), the application developers are generally free to decide how their application should take input and give output (via the command-line, a file system, standard input/output, etc.). Hence developers and service providers must be able to easily configure how application data is transmitted and captured within each step: in particular, defining volumes and assigning them to various steps.

**PUBLIC**

Along with the core creation and edition tools for workflows, the SDK will also provide diverse example and template workflows for developers to get started, as well as comprehensive documentation.

**PUBLIC**

This document is not to be reproduced, modified, adapted, published, translated in any material form in whole or in part nor disclosed to any third party without the prior written permission of THALES ALENIA SPACE.



Co-Funded by the European Union Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.



## A.1 Annex

### A.1.1 Images

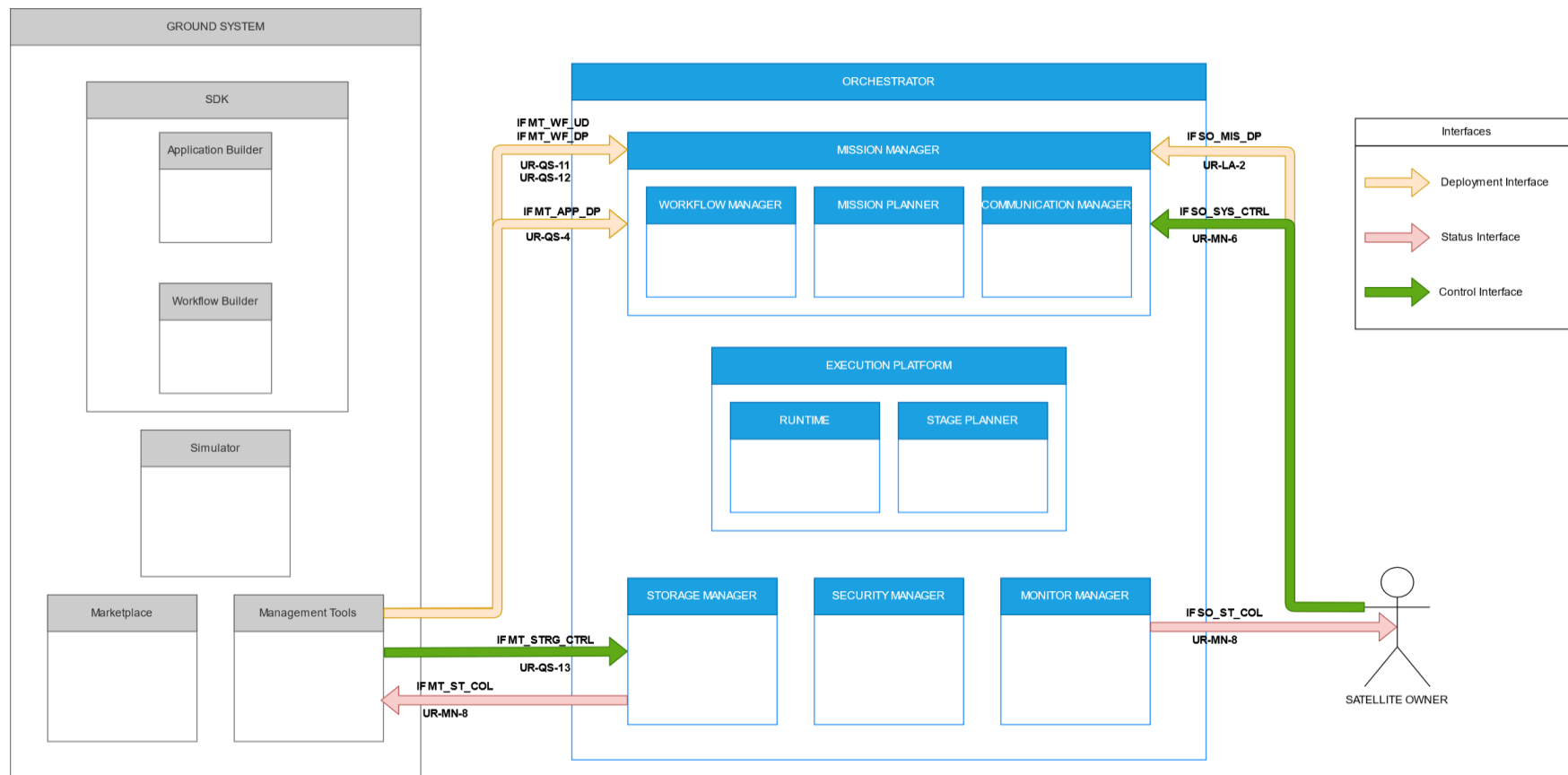


Figure 14. ORCHIDE Logical Overview with External Interface

**PUBLIC**





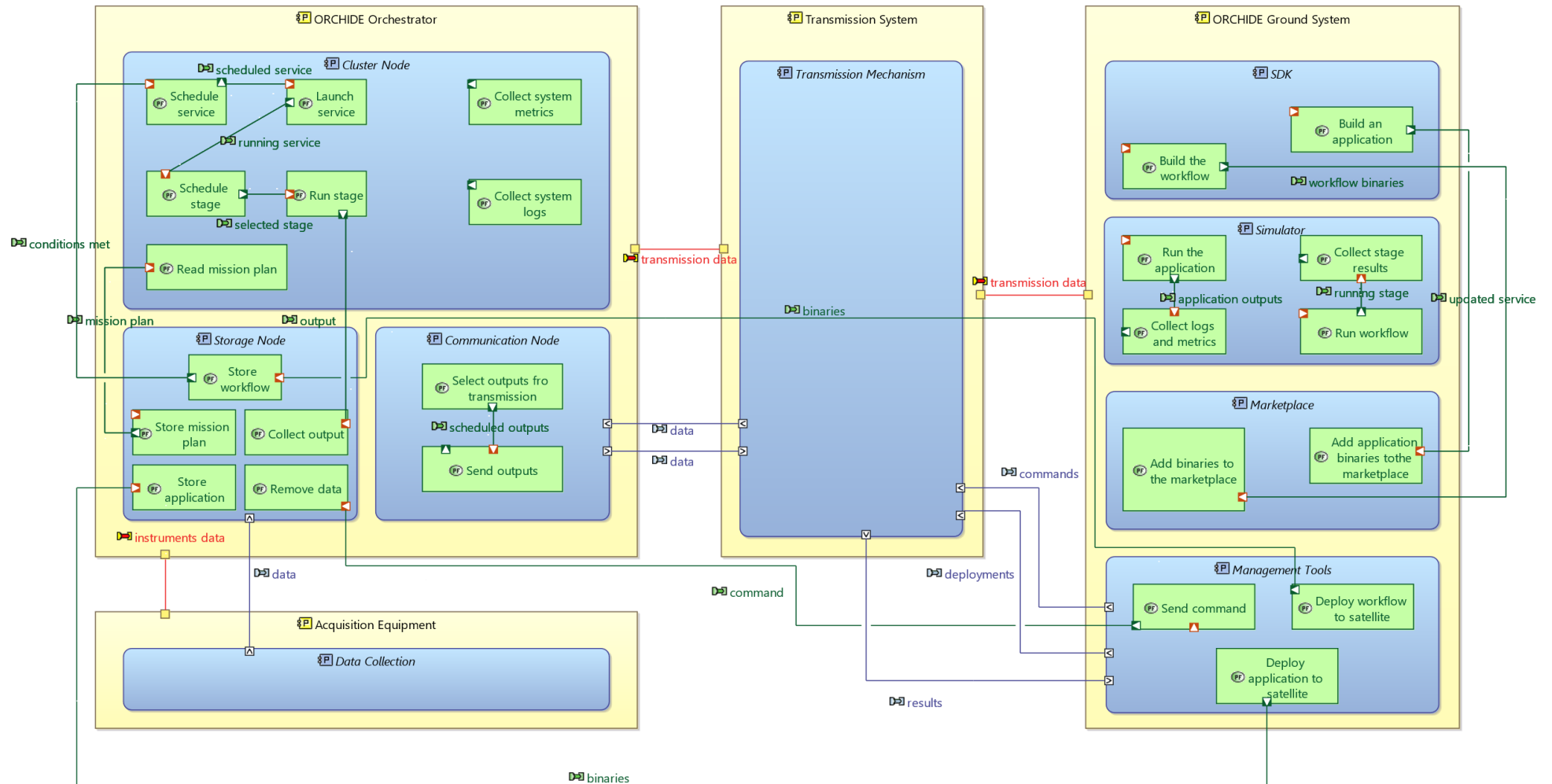


Figure 15. ORCHIDE Physical Components

PUBLIC



END OF DOCUMENT

**PUBLIC**

This document is not to be reproduced, modified, adapted, published, translated in any material form in whole or in part nor disclosed to any third party without the prior written permission of THALES ALENIA SPACE.



Co-Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the granting authority can be held responsible for them.