TRANSPARENT APPLICATION DEPLOYMENT IN A SECURE, ACCELERATED AND COGNITIVE CLOUD CONTINUUM

Grant Agreement no. 101017168

Deliverable D5.1
Abstraction models and intelligent service orchestration

Programme:	H2020-ICT-2020-2
Project number:	101017168
Project acronym:	SERRANO
Start/End date:	01/01/2021 – 31/12/2023

Deliverable type:	Report
Related WP:	WP5
Responsible Editor:	INNOV
Due date:	31/03/2022
Actual submission date:	31/03/2022

Dissemination level:	Public
Revision:	FINAL

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101017168

Revision History

Date	Editor	Status	Version	Changes
29.11.21	INNOV	Draft	0.1	Initial ToC
15.12.22	INNOV	Draft	0.2	Sections 2, 3.1, 4
21.01.22	INNOV	Draft	0.3	Revision and Update
04.03.22	UVT	Draft	0.3.1	Resource Model
04.03.22	NBFC	Draft	0.3.2	Telemetry Data Model draft
08.03.22	INNOV	Draft	0.4	Integrated Version
13.03.22	NBFC	Draft	0.4.1	Telemetry Data Model Updated
15.03.22	INNOV	Draft	0.5	Document Revision and Update
19.03.22	ICCS	Draft	0.5.1	Review Comments
21.03.22	CC	Draft	0.5.2	Review Comments
22.03.22	INNOV	Draft	0.6	Integrated Version with comments
25.03.22	UVT	Draft	0.6.1	Section 3.3 Update
28.03.22	NBFC	Draft	0.6.2	Section 3.4 Update
28.03.22	INNOV	Draft	0.7	Integrated Version with updates
29.03.22	INNOV	Draft	0.8	Revised Document
31.03.22	INNOV	Final	1.0	Final Document

Author List

Organization	Author
INNOV	Stelios Pantelopoulos, Filia Filippou, Andreas Litke
UVT	Adrian Spătaru, Silviu Panica, Ioan Dragan
NBFC	Anastassios Nanos

Internal Reviewers

Panagiotis Kokkinos, ICCS

Márton Sipos, CC

Abstract:

This deliverable (D5.1) presents the outcomes of *Task 5.1 - Abstraction Models and AI-enhanced Service Orchestration* of *Work Package 5 - Intelligence Service and Resource Orchestration* of the SERRANO project during the first iteration of its incremental implementation plan. The deliverable includes the description of the ARDIA (A Resource reference model for Data-Intensive Applications) modelling framework and its three main models for capturing the different aspects of applications and their unique needs, resources and telemetry information in order to facilitate the deployment and execution of the applications in the SERRANO platform. It further includes detailed information about the AI-enhanced Service Orchestrator component of the SERRANO platform and its main background mechanisms that utilise AI/ML techniques for the decomposition of high-level requirements into more specific deployment scenarios for further aiding the deployment of the application in the appropriate infrastructure resources.

Keywords:

ARDIA framework models, Application model, Resource model, Telemetry model, AI-enhanced Service Orchestrator component, background mechanisms, SERRANO platform, usage example

Disclaimer: *The information, documentation and figures available in this deliverable are written by the SERRANO Consortium partners under EC co-financing (project H2020-ICT-101017168) and do not necessarily reflect the view of the European Commission. The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The reader uses the information at his/her sole risk and liability.*

Table of Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
AISO	AI-enhanced Service Orchestrator
ARDD	Application Resource Deployment Description
ARDIA	A Resource reference model for Data-Intensive Applications
CC	Chocolate Cloud ApS
CPU	Central Processing Unit
D	Deliverable
DBSCAN	Density-based spatial clustering of applications with noise
DoW	Description of Work
DPUs	Data Processing Units
EC	European Commission
FPGA	Field-programmable Gate Array
GPU	Graphics Processing Unit
HAC	Hierarchical Agglomerative Clustering
HPC	High Performance Computing
ICCS	Institute of Communication and Computer Systems
INNOV	Innov-acts Ltd
LSTM	Long-short-term memory
M	Month
ML	Machine Learning
NBFC	Nubificus Ltd
NUMA	Non-Uniform Memory Access
PM	Project Manager
PO	Project Officer
REST	REpresentational State Transfer
SSH	Secure Shell Protocol
TOSCA	Topology and Orchestration Specification for Cloud Applications
UC	Use Case
UVT	Universitatea de Vest din Timișoara / West University of Timișoara
VM	Virtual Machine

1 Executive Summary

The deliverable is divided in seven main sections. Section 1 is this executive summary.

Section 2 serves as an introduction to the document. It provides information about the document's purpose, including references to the relevant SERRANO project Work Packages, Tasks and Deliverables, its structure and the audience to which it is mainly addressed.

Section 3 presents in detail the first version of the ARDIA Framework and its three main models, namely the Application Model, the Resource Model and the Telemetry Data Model. For each of these models, their scope, design approach and main characteristics are presented along with their attributes and values organised in categories, (sub-)/classes and parameters/fields. Finally, the model constraints are explained.

Section 4 presents the development efforts of the AI-enhanced Service Orchestration and the resulting component of the SERRANO platform at the end of the first iteration of work. The scope, main functionality and dependencies of the component with other parts of the SERRANO platform are detailed. Its architecture is briefly recapped and in-depth information is provided regarding the background mechanisms enabling its functionality, such as the user-defined mapping rules used to express correspondences among the various ARDIA Framework models and ML techniques used to automatically detect further correspondences based on the telemetry data collected from the execution of the applications in the SERRANO platform.

Section 5 provides a usage example showcasing the service orchestration functionality in the SERRANO platform, which starts with the definition of the application profile and continues with the invocation of the AI-Enhanced Service Orchestrator for the formation of potential deployment scenarios that are the output of the service orchestration procedure. These scenarios are then fed to the SERRANO Resource Orchestrator for aiding the deployment of the application in the appropriate infrastructure resources.

Section 6 concludes the deliverable. It discusses the main findings and future work to be undertaken in the SERRANO project and the second iteration of work.

Section 7 contains information about other sources that have been referred to in the various sections of the document.

2 Introduction

2.1 Purpose of this document

This deliverable presents the outcomes of the *Task 5.1 - Abstraction Models and AI-enhanced Service Orchestration* of the SERRANO project during its first iteration. This task involves the development of a series of abstraction models for representing and describing resources, services and applications and additionally telemetry data. These models are the building blocks of the ARDIA modelling framework that will be used for the service and resource orchestration.

It further includes the development of the application description and profiling mechanisms that fulfil the description and definition of the unique needs of applications in an abstract infrastructure-independent way. These include the goal of the application provider (i.e., intent), such as data-intensity, safety-criticality and others, as well as the particular application requirements and/or prerequisites that are necessary for the deployment and proper execution of the internal application components.

The conducted work also includes the development of the SERRANO AI-enhanced Service Orchestrator (AISO) that utilises AI/ML techniques and the SERRANO service assurance mechanisms in order to form a set of deployment scenarios that facilitate each specific application. These scenarios match the application components to a list of resource types to aid the SERRANO Resource Orchestrator and enable the transparent deployment of applications and the optimal allocation of available resources.

The starting point for this work was the detailed description of the three SERRANO Use Cases, based on which the main service orchestration requirements were elicited and detailed in deliverable *D2.2 - SERRANO use cases, platform requirements and KPIs analysis*. Further source of important input was deliverable *D2.3 - SERRANO architecture* where the whole SERRANO platform architecture and its building blocks were detailed. In that document, the design details of the ARDIA Framework and the AISO were presented along with their main components and their interactions with other components of the platform, while the information flows between the various components regarding orchestration purposes were defined. Deliverable *D2.1 - State of the art analysis report*, in which State of the Art technologies for Intent-based Service Orchestration including Machine Learning (ML) techniques were presented among others, was also carefully consulted for the scopes of this work.

The content of this document will be updated and presented as part of deliverable *D5.4 - Intelligent service and resource orchestration mechanisms* in M31 of the project timeline, after the second iteration of work has been completed.

2.2 Document structure

The present deliverable is split into seven major sections:

- Executive Summary
- Introduction
- The ARDIA Framework
- AI-enhanced Service Orchestration
- Usage Example
- Conclusions

2.3 Audience

This document is publicly available and should be of use to anyone interested in the detailed specification of the ARDIA Framework and the AISO components of the SERRANO platform.

3 The ARDIA Framework

3.1 Overview

The ARDIA Framework and in particular the Abstractions Models that are being presented in this work enable users to describe different aspects of their application in a machine processable format so that it can be then used to facilitate their deployment and execution through the SERRANO platform. For this purpose three different conceptual models have been developed in order to capture several parameters of particular interest for UC (and other) applications that have been directly provided by the application owners or indirectly collected by the SERRANO platform infrastructure. For the description of each application (i.e., provision of application-specific metadata) a model named **Application Model** has been developed with the intention to capture the overall goal of the application provider (i.e., intent) *in an abstract infrastructure-independent way* as well as the particular application requirements and/or prerequisites that are necessary for the deployment and proper execution of the internal application components (e.g., abstract description of the storage policy and/or concrete parameters). The starting point for the design of this model was the *description of the three SERRANO Use Cases* in deliverable D2.2. Its design was further driven by the actual needs for expressivity of the aforementioned application characteristics, as established and refined through discussions with the UC providers.

The metadata provided via this model need to be then expressed in a more concrete way so that they can be used by the Resource Orchestrator for the actual deployment and execution of the application services. For this purpose another model is needed named **Resource Model** with all those parameters being necessary for the deployment of the given application/services to the appropriate infrastructure by the Resource Orchestrator. The design of this model was driven by the parameters that can be specified to best describe each platform type of the cloud continuum. Nevertheless, the concrete values for those parameters may be further adjusted by the Resource Orchestrator. The data collected from the deployment and execution of the application in the SERRANO platform can be expressed using the elements of another model developed, namely the **Telemetry Data model**. This model encompasses several parameters that stem from the execution of an application and facilitate the work of the Service and Resource Orchestrator. Hence, the design of this model was driven by the particular platforms linked with the SERRANO infrastructure and the monitoring data available by each one of them as well as the particular needs of the aforementioned SERRANO software components. For ensuring that the overall goal of the end user is accomplished, the model is expressive enough so that it can capture the outcome of the automatic evaluation process (e.g., parameters exceeding desirable limits during the execution of an application at a particular resource) as well as the quality of experience from this process and especially the feedback that could be provided by the end users, if such a feature is required.

In this work, we use the term “end user” (or just “user”) for referring to the “application owner” who is responsible for the proper deployment and execution and hence it can affect the functionality provided based on their own goals and/or intents. Hence these terms will be used interchangeably in the rest of this document.

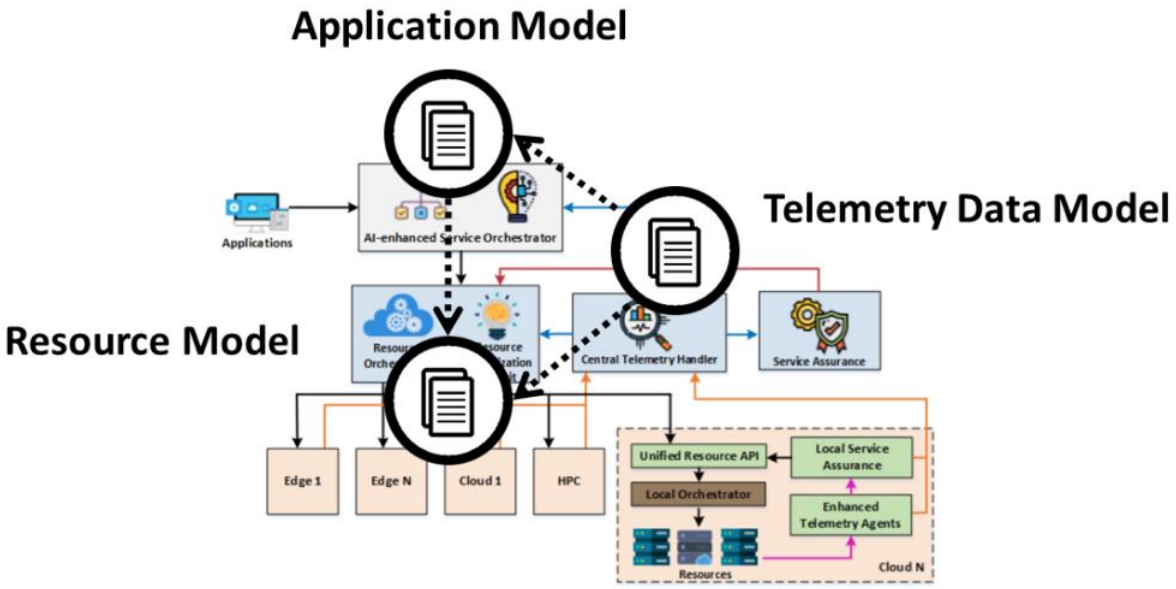

Figure 1: Interaction among SERRANO platform entities and the three Abstraction models

The three aforementioned ARDIA models are depicted in the above figure along with the relevant SERRANO components (i.e., the ones consuming the messages expressed using the elements of the models). The application metadata that are initially expressed using the elements of the Application Model should be accordingly expressed using the elements of the Resource Model (aka mapping or translation process) before their usage for resource orchestration purposes. This process should be driven by rules, ML techniques and forecasting mechanisms that intend to translate the initial abstract parameters to an intermediate or lower level so that they can be finally used for the application deployment, execution and monitoring by the Resource Orchestrator. Due to the high and complex correlation among the elements of the three different models being described in this work we will initially focus on mapping rules that unambiguously specify the relation among the elements. For identifying more complicated relations among the terms of the models, especially for the translation of applications’ description to the appropriate constraints to resources, a post-execution analysis will follow based on the telemetry data that would be collected through the usage of state of the art ML techniques. The knowledge that would be acquired through this process will be incorporated the AI-enhanced Service Orchestration (AISO) mechanism.

3.2 Application Model

3.2.1 Description

The Application Model provides the terminology required for the formal description of a software application from a user's point of view including the particular requirements that the application should satisfy as well as the users' goals and/or intents. Taking into account the aforementioned data, the other software components of the SERRANO platform can reserve the appropriate resources from those ones linked with the platform for the proper execution of the application. Hence, the application model includes several parameters about the input and output of each application as well as the process being internally followed including the compute resources that are directly or indirectly used by each application for accomplishing its purpose, which will be described in detail in the following sections.

3.2.1.1 Application Model Design

Before proceeding to the presentation of the Application Model, we present some parameters taken into consideration during its design.

- *Model Parameters & Granularity of Terms*

The Application Model *should be expressive enough so that it can cover many different applications* including the ones required by the three SERRANO use cases as described in the deliverable D2.2. Taking into account the particular needs of each one of these applications as well as the different domains/fields they come from, some parameters are much more important for the description of an application than the others (e.g., data storage policies are of primary concern for the secure data storage in the UC1, whereas the above parameters are not so important for the analysis of sensor data coming from the ball screws in the UC3). Also, the *granularity of terms* required for the adequate description of the respective constraints (i.e., restrictions over the values of model parameters) may be different (e.g., specification of the exact value of a parameter or the broader category it belongs to such as a set or range of values or even another term with particular meaning). In fact, in some cases, the data provided should be quite abstract (e.g., long or short-term data storage) whereas in some other cases the users should provide the concrete set or range of values in which the respective parameters should belong (e.g., data should be stored for 10 years)

- *Application Formal Description*

For the formal expression of the particular constraints that an application should satisfy, a model-based approach can be followed, according to which several constraints should be specified based on the parameters included in the application model and the possible values of each one of them. Hence, the application model should provide the appropriate terms to be used at a later stage for the introduction of the appropriate constraints. In the following paragraphs, the terms "condition", "constraint" and "restriction" are used interchangeably

for expressing that the value of parameter should belong to the particular set or range of values specified by the end user. The document that includes all the constraints specified about a particular application will be called the application's profile.

- *User's Intent Definition and Formal Expression*

The user's intent expresses the particular goals or intentions that the owner of an application has when using the SERRANO platform for the deployment and execution of his/her own application. From a technical perspective, the user's intent would simply be one or more additional constraints that intent to further restrict the acceptable set or range of values of the parameters of interest. Hence, the user's intent may be directly or indirectly linked with the parameters specified. In the former case, the user's intent refers to existing parameters and may be expressed using the same terminology or a different one (e.g., storage cost being low or less than a particular amount). In the latter case, it refers to additional parameters introduced in the model especially for capturing the user's intent (e.g., low overall cost) and hence these parameters should be further clarified in terms of how they should be linked with the other elements of the application or resource model so that they could be used for driving the deployment decisions.

- *Application Mandatory & Optional Constraints*

The restrictions specified by the end users of a particular application should enable the SERRANO components to employ the appropriate resources for the proper execution of the given application. However, the restrictions specified may not be all equally important. Some of the restrictions should be definitely satisfied (e.g., data should remain within the EU territory otherwise there would be legal implications) and hence such restrictions should be considered as mandatory (aka hard restrictions) whereas the others as optional or nice to have (aka soft restrictions).

3.2.2 Attributes & Values

The following table presents the Application model parameters that can be used for the adequate description of applications and accordingly utilized by the SERRANO platform components for their proper deployment. The parameters are grouped in categories and classes of data. The classification of parameters took place based on the key characteristics of a software application. Nevertheless, some of these parameters may affect different aspects of a software application (e.g., Data Encryption is relevant with both data transmission and data storage while it is also linked with data fragmentation and redundancy level). Regarding their values, in almost all cases the value is a (plain) number (e.g., an integer, such as the number of users, or a percentage, such as the up time) or an amount (i.e., a number followed by a unit of measurement) such as the storage volume. Nevertheless, in some cases, their values can be Boolean, such as whether some parts of the application can be executed in parallel, or even a Term, such as the type of data stored or the programming language used.

Table 1: Application Model Parameters

Category	Class	Sub-Class / Parameter	Parameter	Type / Values
Usage Demand	Number of Users	Number of application Users per Period/ timeframe	Total Number of Users	Number
			Number of Concurrent Users	Number
	Frequency of Use	Requests per Period of Time	Total Number of Requests	Amount
			Number of Concurrent Requests	Number
	Scaling	Number of Instances		Number
Service Level	Availability	Up-time		Number
		Mean-time to Recovery		Number
	Reliability	Reliability Overall		Number
		Failure Rate		Amount
	Accuracy	Acceptable Approximation	Error Margin	Number
			Confidence Level	Term
Application Performance	Execution	Total Execution Time		Amount
		Response Latency		Amount
	Hardware Acceleration	Hardware Type	Graphics Processing Unit (GPU)	Term that indicates the Brand / Model, Core Clock Speed, Memory Type, Size and Bandwidth
			Field-Programmable Gate Array (FPGA)	Term that indicates Vendors, IO count, Gates, Operating frequency, External Interfaces
	Parallelization	Parallelized		Boolean
Data Storage	Data Type (or Category)			Term such as System/Log Files, External/User Files
	Volume	Volume of Storage Data		Amount
	Data Latency	Data Storage Latency	Acceptable Latency	Amount
		Data Retrieval Latency	Acceptable Latency	Amount
	Duration	Storage Duration		Amount or Term(e.g., Long/Short Term)
		Period of Time	Start and/or End Date	Amount(s)

	Location	Proximity to User		Boolean
		Geographical Location		Term (e.g., Country or EU)
Data Transfer	Input Data	Data Type (or Category)	Type of Input Data	Term
		Volume	Volume of Input Data	Amount
	Output Data	Data Type (or Category)	Type of Output Data	Term
		Volume	Volume of Output Data	Amount
Network	Network Capacity	Network Bandwidth	Network Upload Bandwidth	Amount
			Network Download Bandwidth	Amount
		Network Latency		Amount
Security and Privacy	Security	Data Encryption	Data Encryption (Parameters)	Term(s)
			Data Encryption Algorithm	Term
		Erasure Coding	Erasure Coding Scheme	Term
			Erasure Coding Stripe Size	Number
			Level of Redundancy	Number (percentage)
	Privacy	Data Nature		Term e.g. Personal Data, Sensitive Data, IPR, other, none
	Legal Framework			Term e.g., aligned with National, EU/GDPR
Energy and Cost	Energy	Energy Consumption		Amount
	Cost	Overall Cost		Amount
		Data Storage Cost		Amount
		Data Access Cost		Amount
Timeframe	Deployment Duration	Overall	Period (Start and/or End Date)	Amount(s)
		Scheduling	Period (Start and/or End Date and Time)	Amount(s), multiple instances possible
Other Technical Details	Development	Programming Language		Term (e.g., Python or Java)
	Execution Environment	Operating System		Term
		Software Environment	Runtime/Execution Environment	Term
		Software Dependencies	Libraries/Frameworks	Term(s)

3.2.3 Usage Details

An Application Profile is a document in a machine processable format (e.g., JSON) developed based on the elements included in the Application Model. This document may contain several restrictions that depict different aspects that should be taken into account during the orchestration process for the proper deployment and execution of the application. For each restriction specified by the end user, the following elements should be included:

- The parameter of particular interest.
    - Example: Network Download Bandwidth.
- The desirable set or range of values that the value of the aforementioned parameter should belong to. In some cases, the set or range of values can be specified explicitly (e.g., the value should be less than a particular number) or implicitly (e.g., the values should be Low) by the end user.
    - Example: Greater than 1 Gbps (range of values for the above parameter).
- Additional data that should be taken in consideration, such as whether this constraint is mandatory or not or its relevant importance in comparison with the other ones specified.
    - Example: Normal Priority (for the above constraint) or Priority=5 in an integer range [0, 10] – the greater this number is the higher priority the constraint has.

The last bullet is rather important when expressing the user's intent, since it can be used for distinguishing one or more constraints of greater importance for a particular application from the rest of constraints set. This can, in turn, affect how orchestration is performed and alter the possible deployment options. For instance, the user specifies that the data storage size should be greater than a particular value so that the application can work properly (minimum volume of data storage specified). However, if the user also desires that the application responds instantly (i.e., maximum response latency specified) and this constraint is set to have greater importance than the rest, it should be given the highest priority.

The priority of a constraint can consequently restrict the range of acceptable values of the other parameters even more than already restricted due to the specification of other constraints (e.g., data storage volume, among others, should probably be increased even more so that the application can initially allocate more storage space and not have to perform multiple data transfers in order to be able to respond more quickly). It should be further noted that in a different application (or scenario) the same parameter (i.e., response latency) could be set to have the same range but a normal priority, since the end user may want to focus on other priorities (e.g., low energy consumption).

Apart from the aforementioned application model constraints, additional or different data may be required, including lower-level technical details about the application's internal components, the interactions among them and the endpoints of external services being used (e.g., database name, port, etc.), which can be directly consumed by the Resource

Orchestrator component for deployment or other purposes. This information (aka Application Resource Deployment Description - ARDD) should be provided by the use case provider and would contain parameters that cannot be inferred by the given constraints. Additionally, the terminology used in the ARDD document should be also aligned with the Resource Model terms so that the translation of the Application Model constraints to Resource Model terms can be used for revising and/or enriching the values in the ARDD. The data provided in the two aforementioned documents should be enough so that the Recourse Orchestrator can allocate the appropriate resources and deploy the given application. The relation of the Application and Resource Model Constraints with the Deployment Description and the respective SERRANO platform components is depicted in Figure 2. Additional information regarding the Resource Model and its usage can be found in Section 3.3.

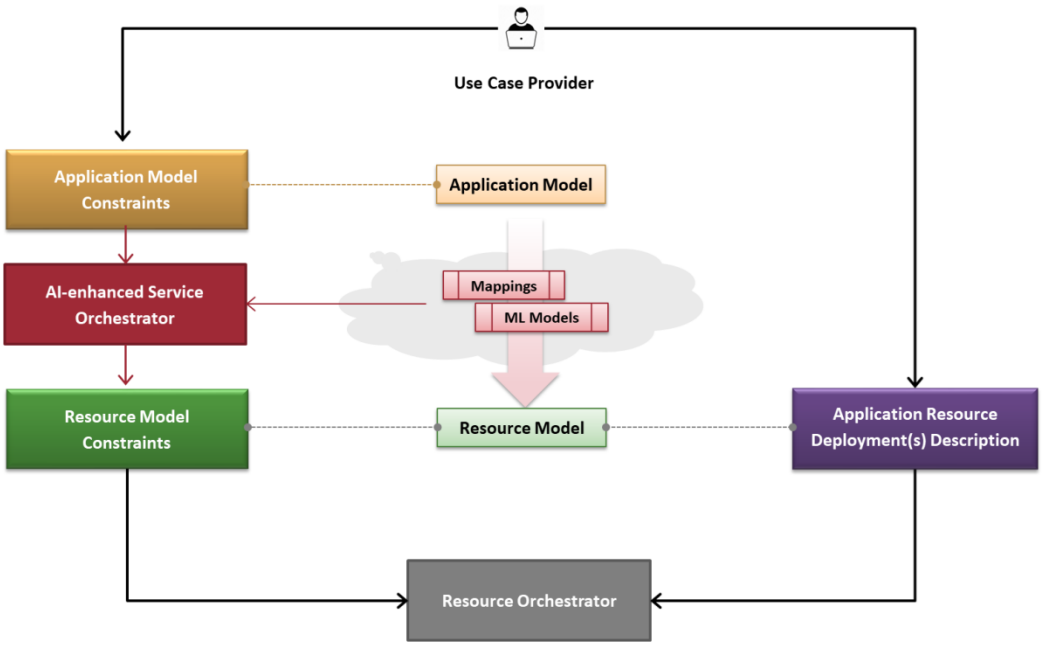

Figure 2: Application and Resource Model Constraints and Deployment Description

More information about the aforementioned process (especially the translation of application to resource model constraints) is provided in Sections 4 (AI-Enhanced Service Orchestrator) and 5 (Usage Example) of this document.

The above constraints refer to the whole application rather than the particular components that may be an internal part of the application for accomplishing its purpose. In the latter case, not only the particular requirements for each component should be specified, but also their order of execution (i.e., the workflow) along with any other preparatory step that should take place (e.g., transfer of relevant input data to the location where the application will be actually run). More focus on these areas will be given in the second iteration of the Task 5.1.

3.3 Resource Model

This chapter introduces an abstraction model for representing the hardware and software resources that will be part of the SERRANO resource fabric. The model is extensible and the entities can be extended to incorporate new hardware types in the future.

3.3.1 Description

The resource model identifies both hardware attributes and definitions for interacting with the respective entity. The scope of this model is not restricted to the internal components that manage SERRANO resources, but includes the Application Developer's perspective, which can impose requirements over the attributes defined in the resource model. The end user executing SERRANO applications may not need to be aware of all resource attributes.

Figure 3 presents the resource model using a class diagram. A SERRANO **Resource** is the base type, only considering a string identifier that must be unique within the resource fabric. Next, we consider two types of SERRANO resources: standalone and attached. **Standalone** resources are resources that have computational capabilities and can be accessed externally via some interface (e.g. SSH, REST).

We consider three interfaces that define the interaction with a standalone resource: *telemetry* (tele), *configuration* (conf), and *execution* (exec). All three interfaces are intended to be used by the Resource Orchestrator for different functionalities. The telemetry interface is used to assess the state of a resource, the configuration interface is used to install software, or maintain the lifecycle of software installed on a resource, while the execution interface is used to execute a computational or storage task, depending on the concrete resource type (execution node, execution service, or storage service).

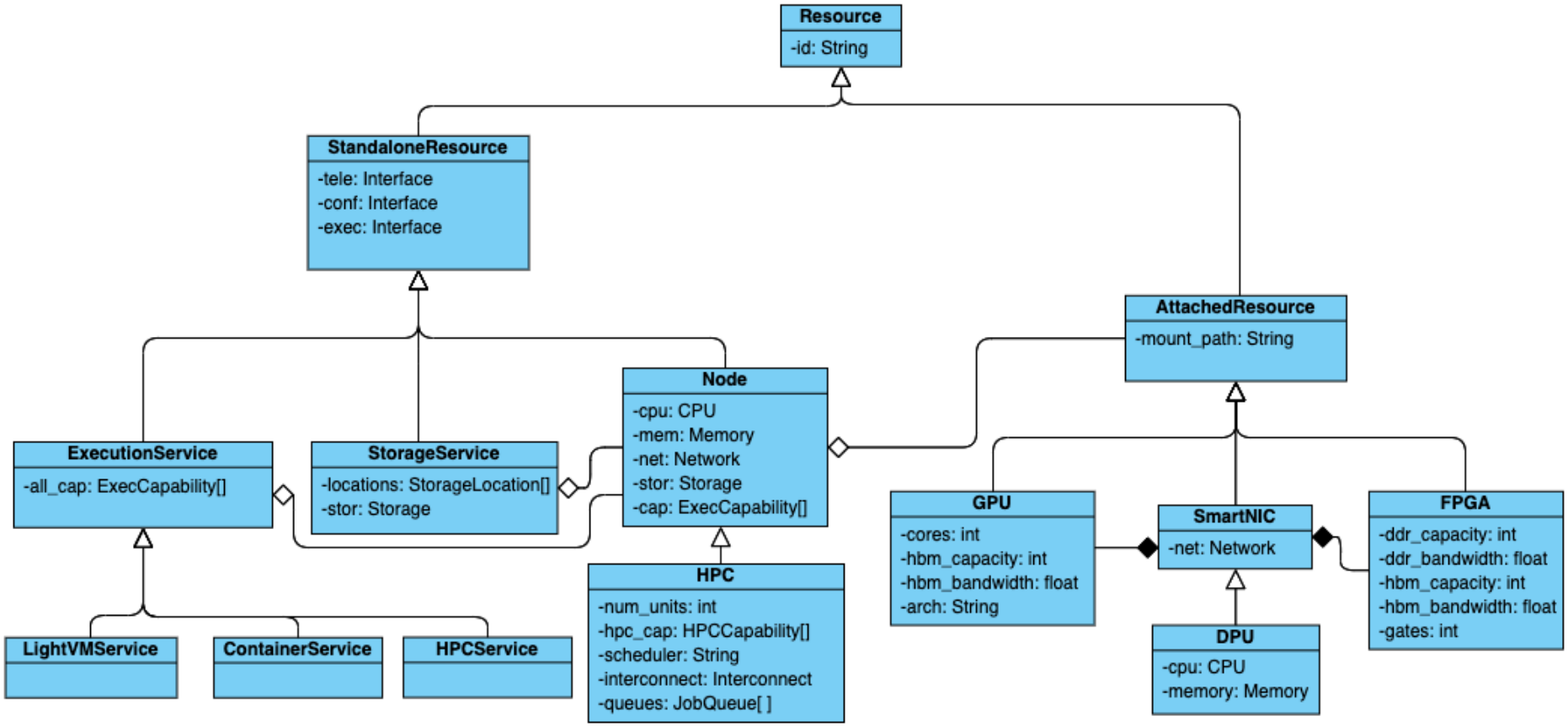

Figure 3: Class diagram representing main entities of the Resource Model

Standalone resources encompass both hardware and software resources. The base hardware resource type is modelled using the concept of a Node. A **Node** can represent an Edge Device, a Virtual Machine, or even Bare Metal servers ready to be aggregated under an execution or storage service. A Node has the following attributes: CPU, Memory, Network, and Storage, all of which are complex types, defining attributes of their own and are presented in Figure 4. Additionally, a Node has an array of Execution Capabilities such as trusted execution enclaves, vector extensions, streaming extensions, and software extensions (e.g. vAccel [2]). Finally, a Node can have one or more Attached Resources such as GPUs, FPGAs, SmartNICs (smart network interface cards) [3,4,5] or DPUs (data processing units) [6]. In our model, we consider that a DPU is a kind of SmartNIC, with more computational capabilities, adding CPU and Memory attributes to represent the on-board memory and CPU.

The **HPC** type models a homogeneous cluster of physical machines such as Supercomputers. The HPC type does not model a collection of Nodes, but instead is considered a special kind of Node, on the grounds of homogeneity along the computational units of the cluster. For this type, most attributes inherited from the Node type will describe a single unit within the cluster (for example a blade) and a new attribute named *num_units* is introduced to represent the amount of worker units in the cluster. The exception is the *network* attribute, which will describe the cluster's connection with the outside world. To describe the link between compute units we introduce the *interconnect* attribute, an extension of Network that additionally defines the topology (e.g. 9d-hypercube). Finally, the HPC Node defines an array of HPCCapabilities (e.g. MPI [7], OpenMP [8]), the scheduler type and an array of JobQueue definitions.

Software resources aggregate hardware resources and offer functional facilities on top. A **Storage Service** handles multiple Storage Locations, each having a cost, and a cost unit, and estimated storage amount, depending on where the data is stored (cloud, edge) and reliability requirements. Additionally, a Storage Location can provide data protection using erasure coding. The execution interface of the Storage Service inherited from Standalone Resource should define the API for storing data using this service.

An **Execution Service** aggregates hardware resources and may use different deployment models to execute applications on these nodes. We distinguish between Lightweight Virtual Machines, Containers, and HPC, each of which will define the interaction using the interfaces defined for a Standalone Resource.

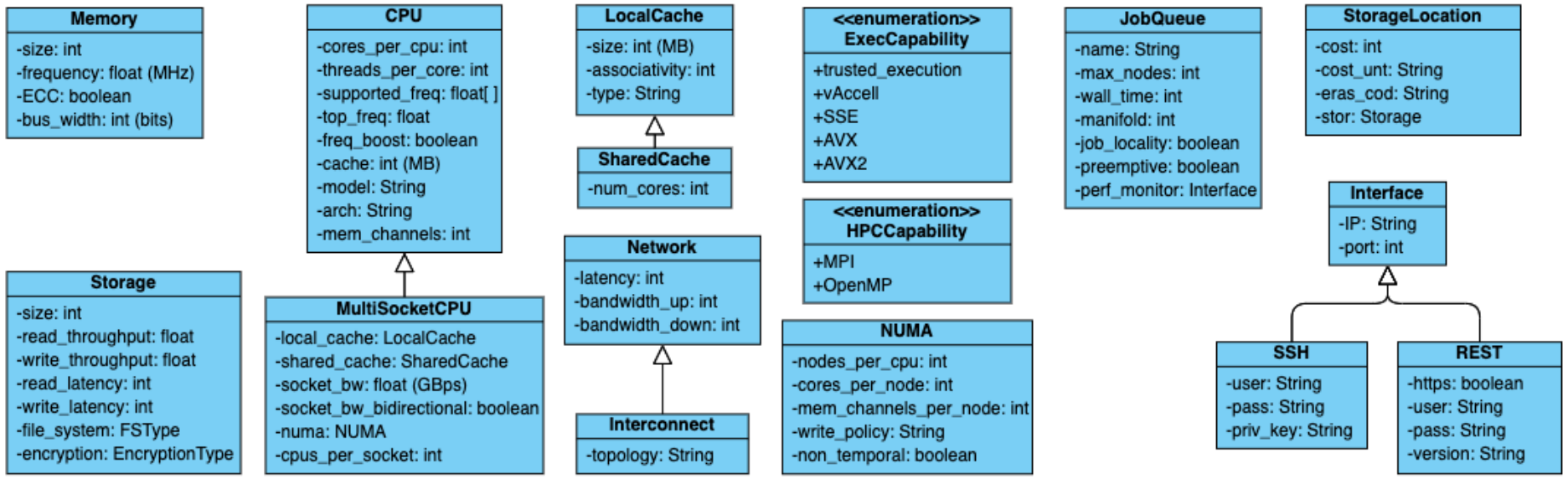

Figure 4: Class diagram representing attributes for resources

3.3.2 Attributes & Values

CPU defines several properties such as number of cores and number of threads, frequency, model and architecture. A MultiSocketCPU is a kind of CPU that we find on server blades which has some additional attributes such as shared cache, socket transfer bandwidth, and NUMA attributes. Memory, Storage, LocalCache, SharedCache, Network, and NUMA are standalone types defining size and performance attributes. A JobQueue will define the maximum number of nodes that can be assigned to one single job, the wall time, job properties as well as a performance monitoring interface. Details for all attributes are shown in Table 2.

Table 2: Details on attributes for resources

Class	Field	Type	Unit	Description
CPU	Cores	int		Number of cores in a CPU unit
	ThreadsPerCore	int		Number of hyperthreads per core
	Frequency	float	GHz	CPU frequency
	Cache	int	MB	CPU cache level size
	Architecture	string		CPU architecture (x86_64, amd64)
	Model	string		CPU Model
	SupportedFrequencies	float[ ]	GHz	Supported CPU Frequency as an array of floats
	FrequencyBoost	boolean		possibility to use frequency boosting
	MemoryChannels	int		Number of memory channels per cpu
Memory	Size	int	GB	RAM memory size
	Frequency	int	MHz	RAM memory frequency
	ECC	boolean		RAM memory support for ECC (error-correcting code)
	BusWidth	int		Bus width per memory channel in bits
Network	UploadBandwidth	int	Mbps	Network communication upload bandwidth
	DownloadBandwidth	int	Mbps	Network communication download bandwidth
	Latency	int	ms	Network communication latency

Storage	Size	int	GB	Storage size in
	ReadThroughput	float	Gbps	Storage performance read throughput
	WriteThroughput	float	Gbps	Storage performance write throughput
	ReadLatency	int	ms	Storage performance read latency
	WriteLatency	int	ms	Storage performance write latency
	FSType	string		Storage file system type (ext4,ext3,xfs,btrfs)
	Encryption	String		Storage encryption algorithm to be used
MultiSocketCPU	NumCPUs	int		Number of CPU units on motherboard
	SocketsBandwidth	float	Gbps	Total bandwidth between two sockets
	SocketsBandwidthBiDirectional	boolean		Is bandwidth between cpus bi-directional, if yes half of total bandwidth in each direction
	LocalCache::Size	int	MB	Size of local cache per core
	LocalCache::Associativity	int		Local cache associativity
	LocalCache::Type	string		Type of local cache
	SharedCache::Size	int	MB	Size of shared cache
	SharedCache::Associativity	int		Shared cache associativity
	SharedCache::Type	string		Type of shared cache
	SharedCache::NumCores	int		Number of cores sharing cache
NUMA	NumaNodes	int		Number of numa nodes per cpu
	CoresPerNuma	int		Number of cores per numa node
	MemoryChannelsPerNuma	int		Number of numa nodes per cpu
	MemoryWritePolicy	string		write-through,write-back=write-back
	MemorySupportNonTemporal	boolean		Support for non-temporal memory access (write)
Interconnect extends Network	Topology	string		e.g., 9D-hypercube

JobQueue	Name	string		name of the queue, used when communication with scheduler
	MaxNodes	int		maximum number of nodes that a job can use
	WallTime	int	seconds	maximum run time for a job
	Manifold	int		interconnect topology may need jobs to request a number of nodes that is divisible by the manifold.
	JobLocality	boolean		support for locality aware scheduling
	Preemptive	boolean		jobs running in an preemptive queue can be shut down in favour of critical jobs.
	PerformanceMonitor	Interface		interface for collecting information about task execution.
Interface	IP	string		IP address
	port	int		IP port
SSH	user	string		username for accessing the server using SSH
	pass	string		password for accessing the server using SSH
	priv_key	string		A private key generated for accessing the server using SSH. The key should be associated with a user with specific and minimal access rights.
REST	https	boolean		Support for secure HTTP requests
	User	string		username for accessing the REST API
	Pass	string		password for accessing the REST API
	Version	string		A string identifying the API and version
ExecCapability	Enumeration	TEE, vAccell, AVX, AVX2, SSE		Capabilities exposed by resources such as trusted execution, vector extensions, and streaming extensions.
HPCCapability	Enumeration	MPI, OpenMP		Capabilities exposed by HPC resources, such as MPI or OpenMp

3.3.3 Usage Details

The model has two facets: the system perspective and the application perspective. An application imposes requirements over hardware capacities such as number of cores, RAM size, etc., but the access interface for managing the selected node is out of the scope of the application. Therefore, the application facet will contain a subset of the attributes presented throughout the section, representing the system facet. Both models will be implemented in machine-readable formats.

The system model will be used when generating the specifications for a hardware resource that is added to the resource fabric. An internal schema can enforce the interoperability of the components, through the exchange of JSON representation of the model.

The application facet of the model will be used to define the requirements of the services composing an application. For example, if TOSCA [1] is used for modelling the application topology, then the corresponding resources will need to be modelled in this language. The current class diagrams are in line with the TOSCA specification, allowing for easily extending the TOSCA base types to provide added functionality.

3.4 Telemetry Data Model

3.4.1 Description

The SERRANO telemetry data model captures the infrastructure runtime measurements of the various components that comprise the SERRANO platform. Specifically it builds on streams of time series that refer to the current state of the hardware and low-level software resources available in the SERRANO platform.

We have identified five basic categories of telemetry types: *compute*, *memory*, *disk*, *network* and *hardware info*, which we further expand into sub-categories to adequately describe the state of the hardware and software components available for allocation and usage. The categories are shown in Table 3.

Table 3: Telemetry Data Categories

Category	Description
compute	Captures compute-related statistics
memory	Captures memory-related statistics
disk	Gathers storage / I/O related statistics such as disk space used, filesystem metrics etc.
network	Exposes network interface statistics such as bytes transferred, packets received, lost, errors etc.
hwinfo	Exposes hardware-sensor statistics like temperature, faults etc.

Table 4 presents the relevant sub-categories, based on the data collected from the SERRANO platform hardware.

Table 4: Telemetry Data Sub-categories

Name	Description	Category
cpu	Exposes CPU statistics	**compute**
loadavg	Exposes load average.	**compute**
vmstat	Exposes statistics from `/proc/vmstat`.	**memory**
meminfo	Exposes memory statistics.	**memory**
filesystem	Exposes filesystem statistics, such as disk space used.	**disk**
diskstats	Exposes disk I/O statistics.	**disk**
netdev	Exposes network interface statistics such as bytes transferred.	**network**
uname	Exposes system information as provided by the uname system call.	**compute**
os	Expose OS release info from `/etc/os-release` or `/usr/lib/os-release`	**compute**
entropy	Exposes available entropy.	**compute**
thermal	Exposes thermal statistics like `pmset -g therm`.	**Hwinfo**

Essentially, the Telemetry Data model acts as the runtime input to the Application and Resource Models by feeding them metrics that describe the current state of the infrastructure, enabling the system to act on various resources (e.g., Deploy an application on a specific hardware resource, or stop a workload and re-spawn elsewhere etc.).

3.4.2 Attributes & Values

In this section, we present the attributes of the measurements captured in the SERRANO platform.

Table 5: CPU statistics

Type	Unit	Description
CPU::node_cpu_seconds_total	int:array of int	nr of seconds for each type of run mode
		user: The time spent in userland
		system: The time spent in the kernel
		iowait: Time spent waiting for I/O
		idle: Time the CPU had nothing to do
		irq&softirq: Time servicing interrupts
		guest: If you are running VMs, the CPU they use
		steal: If you are a VM, time other VMs "stole" from your CPUs

Table 6: Virtual Memory statistics

Type	Unit	Description
VMS::node_vmstat_oom_kill	int:nr_occur	Number of out-of-memory process kills in the system
VMS::node_vmstat_pgfault	int:nr_occur	Number of page fault handled when the relevant page is in memory but not allocated to the requesting process or not marked as present in the memory management unit.
VMS::node_vmstat_pgmajfault	int:nr_occur	Number of page faults handled when the page is not in memory.
VMS::node_vmstat_pgpgin	int:kbytes	Number of kilobytes the system has paged in from disk per second.
VMS::node_vmstat_pgpgout	int:kbytes	Number of kilobytes the system has paged out to disk per second.
VMS::node_vmstat_pswpin	int:kbytes	Number of kilobytes the system has swapped in from disk per second.
VMS::node_vmstat_pswpout	int:kbytes	Number of kilobytes the system has swapped out to disk per second.

Table 7: Operating system version and architecture statistics

Type	Unit	Description
INF::node_uname_info	string:dict	domainname="(none)",machine="x86_64",nodename=" ",release="5.4.0-96-generic",sysname="Linux",version="#109-Ubuntu SMP Wed Jan 12 16:49:16 UTC 2022"

Table 8: Load average statistics

Type	Unit	Description
LDA::node_load1	int:load	Load average (1min)
LDA::node_load15	int:load15	Load average (15min)
LDA::node_load5	int:load5	Load average (5min)

Table 9: Filesystem usage statistics

Type	Unit	Description
FS::node_filesystem_avail_bytes	int:array	Space available in bytes
FS::node_filesystem_device_error	bool:array	Boolean value reporting error in filesystem
FS::node_filesystem_files	int:array	Number of inodes used
FS::node_filesystem_files_free	int:array	Number of free inodes
FS::node_filesystem_free_bytes	int:array	Filesystem size available ignoring the reserved blocks.
FS::node_filesystem_readonly	int:array	Boolean value of read/write permissions
FS::node_filesystem_size_bytes	int:array	Total filesystem size

Table 10: Hardware sensor statistics

Type	Unit	Description
HWI::node_thermal_zone_temp	string:dict	type="chipset", zone="XX"

Table 11: Memory statistics

Type	Unit	Description
MEM::node_memory_PageTables_bytes	int:bytes	amount of memory dedicated to the lowest level of page tables
MEM::node_memory_VmallocUsed_bytes	int:bytes	amount of vmalloc area which is used
MEM::node_memory_Committed_AS_bytes	int:bytes	An estimate of how much RAM you would need to make a 99.99% guarantee that there never is OOM (out of memory) for this workload. Normally the kernel will overcommit memory. That means, say you do a 1GB malloc, nothing happens,really. Only when you start USING that malloc memory you will get real memory on demand, and just as much as you use. So you sort of take a mortgage and hope the bank doesn't go bust. Other cases might include when you mmap a file that's shared only when you write to it and you get a private copy of that data. While it normally is shared between processes. The Committed_AS is a guesstimate

		of how much RAM/swap you would need worst-case.
MEM::node_memory_Inactive_bytes	int:bytes	Assumed to be easily free-able. The kernel will try to keep some clean stuff around always to have a bit of breathing room.
MEM::node_memory_Dirty_bytes	int:bytes	Dirty means “might need writing to disk or swap.” Takes more work to free. Example might be files that have not been written to yet. They aren’t written to memory too soon in order to keep the I/O down. For instance, if you’re writing logs, it might be better to wait until you have a complete log ready before sending it to disk.
MEM::node_memory_Mapped_bytes	int:bytes	files which have been mapped, such as libraries
MEM::node_memory_Slab_bytes	int:bytes	in-kernel data structures cache
MEM::node_memory_MemFree_bytes	int:bytes	Is sum of LowFree+HighFree (overall stat)
MEM::node_memory_VmallocChunk_bytes	int:bytes	largest contigious block of vmalloc area which is free
MEM::node_memory_Buffers_bytes	int:bytes	Memory in buffer cache. mostly useless as metric nowadays Relatively temporary storage for raw disk blocks shouldn’t get tremendously large (20MB or so)
MEM::node_memory_Cached_bytes	int:bytes	Memory in the pagecache (diskcache) minus SwapCache, Doesn’t include SwapCached
MEM::node_memory_Active_bytes	int:bytes	Memory that has been used more recently and usually not reclaimed unless absolute necessary.
MEM::node_memory_SwapCached_bytes	int:bytes	Memory that once was swapped out, is swapped back in but still also is in the swapfile (if memory is needed it doesn’t need to be swapped out AGAIN because it is already in the swapfile. This saves I/O)
MEM::node_memory_Writeback_bytes	int:bytes	Memory which is actively being written back to the disk
MEM::node_memory_SwapTotal_bytes	int:bytes	Total amount of physical swap memory.
MEM::node_memory_Unevictable_bytes	int:bytes	Total amount of physical swap memory.
MEM::node_memory_SwapFree_bytes	int:bytes	Total amount of swap memory free. Memory which has been evicted from RAM, and is temporarily on the disk
MEM::node_memory_VmallocTotal_bytes	int:bytes	total size of vmalloc memory area
MEM::node_memory_MemTotal_bytes	int:bytes	Total usable ram (i.e. physical ram minus a few reserved bits and the kernel binary code)

Table 12: Disk statistics

Type	Unit	Description
DISK::node_disk_discard_time_seconds_total	int:array	number of disard time
DISK::node_disk_discarded_sectors_total	int:array	number of discarded sectors
DISK::node_disk_discards_completed_total	int:array	number of discards completed
DISK::node_disk_discards_merged_total	int:array	number of discards merged
DISK::node_disk_info	string:array	disk information
DISK::node_disk_io_now	int:array	I/O operations
DISK::node_disk_io_time_seconds_total	int:array	number of seconds spent on I/O
DISK::node_disk_io_time_weighted_seconds_total	int:array	number of weighted I/O seconds
DISK::node_disk_read_bytes_total	int:array	number of bytes read
DISK::node_disk_read_time_seconds_total	int:array	number of seconds spent reading bytes
DISK::node_disk_reads_completed_total	int:array	number of reads completed
DISK::node_disk_reads_merged_total	int:array	number of reads merged
DISK::node_disk_write_time_seconds_total	int:array	number of seconds spent on writing bytes
DISK::node_disk_writes_completed_total	int:array	number of completed writes
DISK::node_disk_writes_merged_total	int:array	number of merged writes
DISK::node_disk_written_bytes_total	int:array	number of bytes written

Table 13: Network device statistics

Type	Unit	Description
NET::node_network_address_assign_type	int:address_type	address type
NET::node_network_carrier	bool:carrier	carrier on or off
NET::node_network_carrier_changes_total	int:nr_changes	changes of carrier entry
NET::node_network_carrier_down_changes_total	int:nr_changes	number of times carrier was down
NET::node_network_carrier_up_changes_total	int:nr_changes	number of times carrier was up
NET::node_network_device_id	int:dev_id	device id
NET::node_network_dormant	bool:dormant	indication of a dormant interface
NET::node_network_flags	uint:flags	network device flags
NET::node_network_iface_id	int:iface_id	interface id
NET::node_network_iface_link	int:link_status	link status

NET::node_network_iface_link_mode	int:link_mode	link mode
NET::node_network_info	string:dict	address="02:42:50:1e:95:e4",broadcast="ff:ff:ff:ff:ff:ff",device="docker0",duplex="",ifalias="",operstate="down"
NET::node_network_mtu_bytes	int:bytes	number of Maximum Transfer Unit (MTU) in bytes
NET::node_network_name_assign_type	string:name	device name
NET::node_network_net_dev_group	int:net_group	group id
NET::node_network_protocol_type	int:ether_proto	protocol type
NET::node_network_receive_bytes_total	int:bytes	number of bytes received
NET::node_network_receive_compressed_total	int:packets	number of compressedbytes received
NET::node_network_receive_drop_total	int:packets	number of packets dropped
NET::node_network_receive_errs_total	int:packets	number of received errors
NET::node_network_receive_fifo_total	int:packets	number of fifo requests received
NET::node_network_receive_frame_total	int:bytes	number of frames received
NET::node_network_receive_multicast_total	int:bytes	number of multicast frames received
NET::node_network_receive_packets_total	int:packets	number packets received
NET::node_network_speed_bytes	int:bytes/sec	speed in bytes per second
NET::node_network_transmit_bytes_total	int:bytes	number of bytes transmitted
NET::node_network_transmit_carrier_total	int:packets	number of packets transmitted
NET::node_network_transmit_colls_total	int:packets	number of collisions
NET::node_network_transmit_compressed_total	int:packets	number of compressed packets transmitted
NET::node_network_transmit_drop_total	int:packets	number of packets dropped
NET::node_network_transmit_errs_total	int:packets	number of errors on transmitted packets
NET::node_network_transmit_fifo_total	int:packets	number of fifo packets transmitted
NET::node_network_transmit_packets_total	int:packets	number of transmitted packets
NET::node_network_transmit_queue_length	int:qlen	number of transmitted packets queued
NET::node_network_up	bool:carrier	indication of an active interface

3.4.3 Usage Details

Each metric presented in Section 3.4.2 is captured along with a specific timestamp. The metrics gathered are stored in a timeseries database (Prometheus) that provides querying capabilities for a specific time range.

4 AI-enhanced Service Orchestration

4.1 Overview

Successful deployment (user-wise) of applications and efficient usage of computer resources is a challenging task that depends on several parameters, including but not limited to the capabilities and available capacities of the set of cloud continuum resources that are linked at each given time with the SERRANO platform (e.g., CPU power of edge devices), the particular needs of each application (e.g., data storage volume) and the goals or intents of the users (e.g., application availability, security and energy consumption) as well as the priority assigned to each one of them. The Service Orchestrator undertakes the first part of the orchestration process and cooperates with the Resource Orchestrator for the proper deployment of applications in the SERRANO infrastructure. The Resource Orchestrator focuses on the deployment and execution of an application based on a set of provided parameters. However, an end user may need to a) specify these parameters in a higher level and b) further specify particular goals or intents, which can vary through time. These are not directly usable by the Resource Orchestrator. Hence, it is necessary to further examine these parameters provided by the end user and translate them to the appropriate constraints based on the parameters available for each resource so that they can be effectively used by the Resource Orchestrator. This would be of the primary concern by the AI-enhanced Service Orchestrator (AISO). This component will be equipped with AI techniques (including user defined rules and ML techniques) for assigning potential deployment scenarios and hence facilitate the deployment and execution of the application by the Resource Orchestrator.

A **deployment scenario** consists of a particular set of constraints expressed based on the elements of the Resource Model that determine the type and characteristics that the infrastructure should have. Since the relations among parameters are complicated and the constraints specified by the end user can be quite abstract there are often more than one deployment scenarios possible that satisfy these constraints.

4.2 Functionality and Dependencies

The AISO is responsible for the translation of the parameters specified by the end user using the elements of the Application Model to the appropriate constraints using the elements of the Resource Model. The output of this component is a list of potential deployment scenarios that are given to the Resource Orchestrator, which is responsible for the deployment and execution of the application, taking into account the restrictions provided and the computer resources linked with the SERRANO platform.

The AISO will be available as a REST service. The input of the AISO would be a JSON File that encompasses several constraints regarding the application andits execution (including user's intents and other application and infrastructure-related parameters of importance) that should be expressed based on the elements of the Application Model. The output of the

AISO would be another JSON File with the potential deployment scenarios that could take place that include the high-level restrictions that the respective resources should satisfy. The latter would be expressed based on the elements of the Resource Model and will be given to the Resource Orchestrator by invoking the corresponding REST service.

The data collected by the SERRANO platform regarding the execution of each application should be expressed using the elements of the Telemetry Data Model and stored in a predefined location so that they can be further analysed (either online or offline) for improving the decisions made by the AISO during the translation process.

4.3 Component Architecture

The basic components of the AISO has been described in the deliverable D2.3. Figure 5 presents the components of the AISO along with the interaction with the other entities of the SERRANO platform.

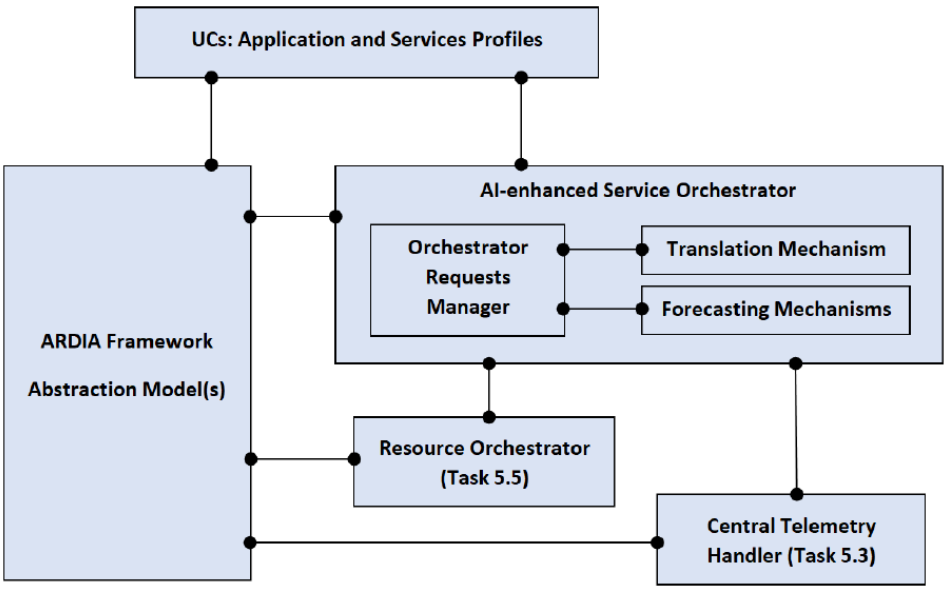

Figure 5: AI-Enhanced Service Orchestrator

The main component of the AISO is the Requests Managers, which is responsible to handle the data provided by the end user regarding the execution of an application as well as the Telemetry Data collected by the Central Telemetry Handler. This component provides to the Resource Orchestrator the possible deployment scenarios. More precisely, for each one of them, it produces a list of resource model constraints based on the application model constrains and the priority assigned to each one of them. For this purpose, it internally uses two different components. The first component (Translation Mechanism) uses the mapping rules specified (analytically described in the following subsection) for making the appropriate decisions regarding the deployment scenarios that can take place and translating the applications parameters to the appropriate resource constraints. The second component (Forecasting Mechanism) uses machine-learning techniques for the analysis of telemetry data and the prediction of the appropriate values or range of values of resource parameters

so that they can be accordingly used for the correct translation of the application constraints. For example, for enabling one hundred users to simultaneously access an application, the CPU (e.g., CPU cores and threads) should have certain characteristics. The background mechanisms used by these two components are described in the following section.

4.4 Background Mechanisms

4.4.1 User-defined Mapping Rules

The functionality provided by the AISO depends on the mapping rules specified that express the correspondence among the elements of the Application Model with those ones specified in the Resource Model. For this purpose, there was close cooperation among the technical experts of the SERRANO project in order to identify the dependencies (relevancies) among the Resource Model and Application Model parameters and define the exact relations. The first iteration of this process, which will be completed during the second iteration of work, indicated that, in some cases, there is a one-to-one correspondence among them such as the following mapping rule (Table 14). Nevertheless, in many cases, the relation among them is much more complicated and hence further investigation is necessary. For this purpose, analysis of data that would be collected from the execution of the same or similar applications is necessary. This is covered in the following sub-section.

Each one of the mapping rules has a declarative and a procedural part. The declarative part specifies the source and target elements from the application and resource models respectively and optionally the range of values of such parameters. The procedural part (if being necessary) specifies the process that should be followed for expressing the given data/constraints using the elements of the resource model. Additional information about each one of them could be also provided in metadata, including, but not limited to, the target infrastructure type (e.g., cloud) and its origin (e.g., manually specified or produced using ML techniques – see next section).

Table 14: An example of a user-defined mapping rule

Mapping Rule	Left Side	Right Side
Declarative Part	Application Model • Data Storage Volume	Resource Model • Storage Size
Procedural Part	Adjust Values based on the units of measurements	

Using the above elements, more complicated rules can be also specified. For instance, if an application should be almost always available (e.g., Availability or Up-Time greater than or equal to 99.999%) and instantly reply to the end user requests (e.g., Response Latency less than or equal to 1 ms) then it should be deployed to a cloud provider with adequate computational capacities (such as the number of cores, the number of threads per CPU, the

cache size, etc. Concrete details have been deliberately omitted from this example, as this should be the output of telemetry data analysis). In such a case, there is no need for a procedural part.

The AISO initially examines the parameters specified for each application and the mapping rules already specified (especially the left side of each mapping rule) in order to find and execute the appropriate ones. Based on the Mapping Rules specified, it identifies the different infrastructure types that can be used (e.g., cloud provider, edge device, HPC cluster) and the mapping rules that refer to each one. Then in applies the mapping rules one after the other. During this process, more than one deployment scenario (branches) can be produced, even for the same type of infrastructure, when the same parameters can be translated in more than one ways. The Mapping Rules with higher priority are applied at the end of this process, since they can modify the set or range of values of resource model parameters. Since a resource model constraint may be affected by more than one mapping rules, at the end of this process the system detects the constraints that apply to the same resource parameter and finds the appropriate subset or range of values so that all the constraints are satisfied. In case of a conflict (e.g., when there are contradictions in the generated constraints that arose from the application of different mapping rules) the user is being informed.

4.4.2 Automatically-detected Correspondences using ML Techniques

For identifying additional mapping rules among the elements of the Application and Resource model, Machine Learning techniques will be used (described in D2.1). More precisely, the telemetry data collected from the execution of each application through the SERRANO platform will be used in order to further study the relation among the elements of the two models taking into account the type of resource they come from (e.g., cloud provider and/or edge device) and the particular configuration being used (e.g., CPU and RAM limitations). For example, the data collected from the execution of the same or similar applications in an edge device could aid the analysis of the set restrictions set (i.e., constraints expressed using the Resource Model terms) and their impact in the rest parameters of each resource. During this process widely used data clustering techniques such as K-means, DBSCAN and HAC can be used for organizing the data collected in categories based on the values of the parameters recorded so far.

In case additional information is provided by the user about the execution of each application, such as whether they are satisfied with the decision made regarding execution (including the translation process), the data collected could be used for training a model that automatically suggests the appropriate restrictions based on the parameters (e.g., intents) specified by the end user. Taking into account that the telemetry data often capture the resource status at a particular point of time (or generally during a short period of time) we can use a Long-short-term memory (LSTM) neural network for this purpose. For example, we can train two different LSTM networks that predict the value of one or more resource parameters when the user is satisfied (LSTM-1) or not (LSTM-2) and utilize this information when the user would like to execute the same or a similar application for taking the

appropriate decisions (i.e., suggest values according to which the end user will be most probably satisfied rather not).

In the following iterations of this task, we are planning to examine two different approaches. In the first approach, we will focus on the data collected and train a deep neural network (e.g., LSTM) so that we can accordingly use it for making the appropriate decisions. The second approach would be to use reinforcement learning for making the appropriate adjustments to the translation mechanism (especially for intent based service execution) based on the feedback provided by the user during or after the execution of an application. In this case, the ML model will be an internal part of the overall procedure that makes the appropriate decisions during the translation process and utilizes the feedback received from the environment (in our case, SERRANO infrastructure) for its own improvement.

5 Usage Example

5.1 Usage Scenario

In this section, we briefly present the utilization of the models developed for the formal description of the main parameters of an application as well as the role of the AISO in this process.

For this purpose, we use an application from those already described in the second SERRANO Use Case (UC2 – Fintech) in the deliverable D2.2. More precisely, we assume an application about the analysis of portfolios (a component of the portfolio optimization process, which is part of this Use Case) that enables external users to analyse a specific investment portfolio with respect to the past data. This application takes as input an investment portfolio and historical price data (in the form of candlestick chart components [9]) about each investment instrument that composes the portfolio. The output is a brief analysis of the given portfolio that is being presented in a user friendly manner. The portfolio analysis process calculates various technical metrics about the investment, such as return, sharpe ratio, drawdown, etc. In addition to the analysis related to the past, the portfolio analysis may also include analysis related to the future, forecasting and projections regarding the expected values of these technical metrics at a specific moment in the future. The analysis internally uses historical data that can be obtained from a database, file or an online data source service (e.g., Bloomberg) that is being stored locally and reused for subsequent portfolios.

5.2 Application Profile

The application provider would like to make this application available to the public (or registered users) and hence this application should be offered as a service. Additionally, it should be always available and able to respond quickly to a large number of external users that can simultaneously use the application for the same purpose. For the temporal storage of application data at least 10 GB is needed. Also, for protecting sensitive user data from potential intruders or other threats, the user data should be encrypted prior to their storage. The application internally follows several steps some of which can be executed in parallel. Apart from this information, we assume that there is no other information about the exact process being followed and the interaction among the internal components of the application (black box assumption). Hence, using the elements of the Application Model the application provider can specify the following constraints:

Table 15: Application Constraints based on Application Model Parameters

Application Model Parameter	Set or Range of Values
Service Availability - Up-time	99.999%
Application Response Latency	maximum 1 second

Concurrent Number of Users	at least 100
Data Storage Volume	at least 10 GB
Security Encryption Algorithm	AES
Application Parallelization	YES

5.3 AI-Enhanced Service Orchestration and Deployment Scenarios

The AISO, based on the mapping rules already specified, produces possible deployment scenarios. As discussed, some of the parameters existing in the Application Model are directly linked with the ones existing in the Resource Model such as the data storage volume and encryption algorithms. In this case, the translation of the constraints specified is quite straightforward. Other parameters, such as the application response latency and the number of concurrent users, are linked to the Resource Model elements, such as the CPU cores/threads needed for the application execution. For this purpose, the analysis of past data of applications with similar characteristics is necessary since it may indicate specific range of values for the CPU cores and the threads required in order to satisfy future applications' requirements, as already described in section 4.4.2.

For example, let's assume that past data indicate that particular restrictions regarding the maximum response time and the minimum number of concurrent users that can be satisfied using (i) a minimum number of 8 CPU cores and a minimum number of 100 threads or (ii) using a minimum number of 16 CPU cores and a minimum number of 50 threads. Then, taking into account this information the AISO produces two different deployment scenario(s) like the following ones (Table 16).

Table 16: Application and Resource Model Constraints for a Cloud Provider

Resource Model Parameter	Set or Range of Values
Deployment Scenario 1:	
CPU cores	minimum 8
CPU threads	minimum 100
Memory Size	minimum 4 GB
Data Storage Size	at least 10 GB
Data Storage Encryption	AES
Deployment Scenario 2:	
CPU cores	minimum 16
CPU threads	minimum 50
Memory Size	minimum 4 GB
Data Storage Size	at least 10 GB
Data Storage Encryption	AES

Considering that the application should be offered as a service with high availability (up time: 99.999%), it should be deployed to a cloud infrastructure. Additionally, data storage size, encryption and memory should be within the range specified.

The deployment scenario(s) can aid the Resource Orchestrator allocate the most suitable infrastructure type and provider to deploy the application to. For the deployment of the application in a specific provider type (e.g., cloud provider, HPC infrastructure or edge device) additional information may be necessary that the application provider should provide in advance in the appropriate format (more information available at section 3.2.3). This kind of information has been deliberately omitted in the presentation of this example.

6 Conclusions

In this deliverable, we presented the first version of the ARDIA framework and the three abstraction models developed. The Application Model provides the terminology for expressing the main parameters of an application in a platform-independent way (aka Application Profile) whereas the Resource and Telemetry Data Model focus on the capabilities of the Resources and the data collected from different entities of the SERRANO platform respectively. The AISO facilitates the deployment and execution of each application taking into account the data provided by the user (i.e., the owner of an application) and their impact in the particular resources. In particular, this component can provide different deployment scenarios and the restrictions that the resources should satisfy so that the deployment that takes place is aligned with the user's needs, goals and intents.

In the first iteration of the Task 5.1 we have focused on the three abstraction models and the functionality that the AISO should provide. This component is currently based on user-defined rules that express the relations among a few parameters of the application model with the ones existing in the resource model. In general the relations among the terms of the two model are much more complicated and the user-provided data can be translated in more than one ways. For this purpose, in the second iteration of the Task 5.1 we plan to use ML techniques for learning from data collected from several applications and improving the system behaviour. Another feature to be considered for inclusion in the abstraction models and the respective SO components is the feedback provided by the users regarding the deployment of their own applications, which could further aid the improvement of the automatically detected translations.

7 References

[1] OASIS Topology and Orchestration Specification for Cloud Applications (TOSCA), available at https://www.oasis-open.org/committees/tosca/

[2] A. Nanos, B. Chalios and K. Papazafeiropoulos, "vAccel a lightweight framework to expose hardware acceleration functionality to VM tenants", *FOSDEM '21*, 2021, 23.06.2021, [online] Available: https://docs.vaccel.org/

[3] Netronome. Netronome Agilio SmartNIC. https:// www.netronome.com/products/agilio-cx/, 2018.

[4] Cavium. Cavium LiquidIO SmartNICs. https://cavium.com/pdfFiles/LiquidIO_II_CN78XX_Product_Brief-Rev1.pdf, 2018.

[5] Broadcom. Broadcom Stingray SmartNICs. https://www.broadcom.com/products/ethernet - connectivity/smartnic/ps225, 2018.

[6] Mellanox. Mellanox BlueField Platforms. http://www.mellanox.com/related-docs/npu-multicore - processors/PB_BlueField_Ref_Platform.pdf, 2018.

[7] Gabriel, Edgar, et al. "Open MPI: Goals, concept, and design of a next generation MPI implementation." *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, Berlin, Heidelberg, 2004.

[8] Chandra, Rohit, et al. *Parallel programming in OpenMP*. Morgan kaufmann, 2001.

[9] Introduction to Candlesticks, available at https://school.stockcharts.com/doku.php?id=chart_analysis:introduction_to_candlesticks