HORIZON2020 FRAMEWORK PROGRAMME

TOPIC EUK-03-2016

"Federated Cloud resource brokerage for mobile cloud services"

D2.3
Global Architecture Design

Project acronym: BASMATI

Project full title: *Cloud Brokerage Across Borders for Mobile Users and Applications*

Contract no.: 723131

Workpackage:	2	
Editor:	Emanuele Carlini	CNR
Author(s):	Emanuele Carlini, Massimo Coppola, Patrizio Dazzi, Vinicius M.De Lira	CNR
	Antonia Schwichtenberg, Richard Wacker, Corinna Lechler	CAS
	Myoungjin Kim, David Lee	INNO
	Jörn Altmann, Netsanet Haile, Baseem Al-athwari	SNU
	Konstantinos Tserpes, John Violos, Vaggelis Psomakelis	ICCS
	Jamie Marshall	AMEN
	Young-Woo Jung, DongJae Kang, Sunwook Kim, Ganis Zulfa Santoso	ETRI
	Enric Pages, Ana Juan Ferrer	ATOS
Authorized by	Konstantinos Tserpes	ICCS
Doc Ref:	D2.3	
Reviewer	Konstantinos Tserpes	ICCS
Dissemination Level	Public	

Document History

Version	Date	Changes	Author/Affiliation
v.0.1	23-09-2016	Created ToC	Patrizio Dazzi / CNR
v.0.2	27-09-2016	Added architecture component diagram	Emanuele Carlini / CNR
v.0.3	08-10-2016	Added sections from the draft document and some output of telcos.	Emanuele Carlini, Massimo Coppola / CNR
v.0.4	06-12-2016	Modifed Acronyms, ToC, added explanations of contents for some sections	Massimo Coppola, Emanuele Carlini / CNR
V0.5	01-01-2017	Add user modeling	Antonia Schwichtenberg / CAS
V0.6	23-01-2017	Reworked ToC. Updated with the last description for each component. Description of the logical diagram of the architecture.	Emanuele Carlini, Massimo Coppola, Patrizio Dazzi / CNR
V0.7	25-01-2017	Monitoring, Integration Strategies, Application and User data collector	/INNO
V0.8	25-01-2017	SLA Violation flow, other fixes	/CNR
V0.9	26-01-2017	Decision Maker, Resources Broker & Edge Execution	/SNU
V10	26-01-2017	Application Controller, resource brokering, provider deployment	/ETRI
V11	26-01-2017	Federation Data Management	/ICCS
V12	26-01-2017	BASMATI Applications	/CNR
V13	31-01-2017	Knowledge Extractor Monitoring	/ICCS /INNO
V14	31-01-2017	Federation Data Management Knowledge Extractor	/ICCS /CAS
V15		Application deployment flow, Decision Maker	/CNR /SNU /CAS
V16	06-02-2017	Edge SLA Manager	/ATOS
V17-18	01-03-2017	Document finalization	/CNR
V21	28-11-2017	Compliance review	K. Tserpes/ICCS
V22	27-11-2018	Revised version to address reviewers' comments	K. Tserpes/ICCS, Patrizio Dazzi, Emanuele Carlini/CNR, Jamie Marshall/AMEN

BASMATI Glossary

Term/Acronym	Definition
Mobile cloud services	Online services offered by cloud resources to support mobile apps. The backend of the mobile apps.
CP	Cloud Provider. The actor that provides the cloud infrastructure/resources, such as VMs.
CSP	Cloud Service Provider. The actor that provides cloud services on top of a rent infrastructure from a CP.
Cloudlet	Limited capacity infrastructures with virtualization capabilities, often used to support a limited amount of users or perform a limited set of operations on behalf of the central cloud infrastructure that hosts the complete application
Edge resources	Resources aimed to operate specialized functionality, located at the "edge" of the network infrastructure, thus, closer to the end users. Examples are (clusters of) RaspberryPis or cloudlets
BUDaMaF	BASMATI Unified Data Management Framework
KE	Knowledge Extractor
DM	Decision Maker
RB	Resource Broker
MVD	Mobile Virtual Desktop
DASFEST	An 3-day long music festival taking place in Karlsruhe, Germany every July
ACE	Amenesik Cloud Engine. The cloud service deployment tool through which actual federation is achieved
BEAM	BASMATI Enhanced Application Model. An extension of the TOSCA specification
ASP	Application Service Provider. A Federation user that rents resource services in order to provide an Application services to End-users
Brokering	The matchmaking support provided by BASMATI platform to decide about the best cloud resources to exploit for the execution of the back-end of BASMATI applications. This activity regards the placement of the services or data on computational resources and storages belonging to the cloud datacentre and the cloudlets within the federation.
End user	A user who benefits the various application and infrastructure services provided by the Cloud. Within BASMATI, the most typical example is exploiting the Cloud federation via a mobile device (possibly a laptop) using specialized apps or a web browser.
Offloading	The ability of BASMATI platform supporting the runtime placement of the components composing the front-end of BASMATI applications on edge resources available nearby the end user. This activity takes place both when edge and mobiles exchange one each other their own workload or when such devices transfer some workload to the clouds or cloudlets. In BASMATI we often distinguish Front-end offloading, related to the mobile part of application, from Back-end offloading, concerning the server side of

	applications. The latter roughly translates to the known concept of Cloudbursting.
QoE	Quality of experience. It is a measure of a customer's experiences with a service. It may be related to some aspects of the QoS and QoP but can also take into account other metrics.
Service handover	Service handover refers to the activity of transferring an active service between two computational resources (e.g. Cloudlets) with minimal or no disruption on the availability of the service. Ideally, service handover is transparent with respect to the user.
Situational Awareness	The ability of the BASMATI platform to recognise the "situation" characterising the actual combined status of users, applications and resources, aimed at achieving an effective and efficient management of applications and resources.

Executive Summary

The ultimate goal of the architecture work package of BASMATI is to ensure that all consortium members have a common vision of the global architecture of the system and that all developers are aware of the interfaces exported by any architecture component to others. This deliverable defines the architecture of the BASMATI platform.

Overall, the objective of the BASMATI project is to design, implement and evaluate a dynamic and integrated brokerage platform targeting federated clouds that supports the dynamic needs of mobile applications and users. To this end, the BASMATI architecture provides a common ground to address key technological and research challenges in different research fields. These challenges mainly focus on three core aspects: (i) User, application and situation modelling and understanding to drive application placement; (ii) Runtime adaptivity and reconfiguration; (iii) Brokering and Offloading of application and services.

The resulting architecture is divided into layers, which stem from several choices taken at design time. The first choice is the separation in the management of those services that natively run on the server side from those that run on a client device. The second choice is to separate the computational plane (how computation is organized, what are the functional dependencies from the services composing the application) from the data plane, in order to foster advanced computation and data orchestration techniques. The last, and probably the most important choice, is the definition of a specific application model (which we refer to as BEAM) that encompasses all the many views of the application within the platform.

Finally, the architecture defined in this document has also been designed to support the requirements identified at the beginning of the project, from the use cases, in accordance with both the joint Korean and European use cases' needs. A final version of this document was created at M18.

Table of Contents

The aim of this document is to define the overall architecture of the BASMATI platform, providing a general picture and framing the overall architectural model. This document deliberately avoids deep technical details, aiming at being easy to read, comment and understand each of its parts.

The document is structured as follows. In section 1 we summarize the core concepts in the BASMATI ecosystem and the motivations for BASMATI; to this purpose we provide a description of the basic concepts, a short definition of the terms which are core concepts for BASMATI, as well as terms which have a specific meaning within the project, also putting them in context with the main pillars of the BASMATI environment. In particular, Section 1.1.2, documents the structure of BASMATI applications,

Section 3 reports about the positioning of BASMATI with respect to Cloud providers and the cloud software stack.

In Section 4 we present the abstract, high-level view of the BASMATI platform. We provide a preliminary view of the BASMATI architecture and its components as a component diagram, as well as an analysis of how requirements from use cases are covered by the architecture. In Section 4, we then provide a high-level description of all components of each architectural layer, including their management of data, their interaction with other component interfaces, and their deployment. Section 5 reports behavioral and sequence diagrams for core use cases of the platform, such as application back-end deployment and runtime reconfiguration.

1 BASMATI concepts and positioning

The mobile apps market displays a staggering growth during the past few years. Whether it is examined from the point of view of the number of applications developed, users, or revenue, there is a market share for practically every involved stake-holders. But, in order for the application developers to bite their share of this market, there is a catch: keep the end users happy. The mobile app users essentially expect no less than top-notch quality no matter where and when they decide to use the app. The mobile app developers need to cope with these requirements otherwise they will get “punished” by the users with low rankings and bad reputation; a nearly catastrophic event for a small mobile app developer.

The two main challenges for meeting those requirements are latency and load distribution. The questions are “how can the application provider continue to meet the requirements when”: a) the user is moving away from the mobile app infrastructure and b) when a large concentration of users appears in a confined space one day and disappears the other.

These sorts of challenges were tackled by the big application and infrastructure providers such as Facebook, Google, Amazon, etc. through redundancy and at a great infrastructure cost. Currently, there is a datacenter proactively meeting the requirements of mobile end users in most of the regions of the world. However, this approach is not viable for SMEs who cannot

afford the costs of such infrastructure developments. As a result, the SMEs also turn to the big infrastructure vendors and lease their operations in order to be able to provide their mobile applications while meeting the end users' requirements. Nearly 80% of the total effort in developing a mobile app goes to the backend, which in turn is controlled almost invariably by the infrastructure providers. This control is further enhanced with the advent of serverless computing and functions as a service (FaaS). Inevitably, the small players in the mobile development market become more or less dependent on the big service providers.

This fact fueled the motivation behind BASMATI. BASMATI sought to provide a financially viable way for the SMEs to provide the infrastructure that will allow them to host their backend mobile app services so as to meet the end user requirements. Considering the dependency of the EU and Korean market on the SMEs, there is one more dimension added to this endeavor: BASMATI can assist in securing the robustness of EU and Korean small mobile app developers (like INNO and YellowMap) by intercepting the flow of revenue that is directed towards non-EU and non-Korean companies.

Besides the optimistic nature of the vision, the BASMATI approach is realistic in the sense that it does not seek to radically change the way things operate, but rather to intelligently utilize the existing tools on behalf of the small mobile app developers. The key value proposition is to intelligently leverage on the existing infrastructure and business models to allow mobile app developers to use only what is needed, when it is needed and to benefit from the possible combinations of cloud service providers' offerings in situations where the competition between them leads to new and unique opportunities.

As such, BASMATI set as its objectives to enable SMEs to predict resource demand and proactively deploy application components to cloud-cloud and edge-cloud federations based on cost-related utility functions. To deal with the interoperability issues, that rise as a result of the dynamic selection from a pool of heterogeneous resources, but also ease-of-use, BASMATI presented a federation mechanism as well as the application description model, entitled BEAM.

The last -and most visionary- objective of BASMATI, is to provide the necessary incentives to the small mobile app developers to form coalitions through multi-cloud or edge-cloud federations, alleviating each another from the extra costs and overheads imposed by the big cloud providers when one is seeking to extend the leased infrastructures for the application purposes.

BASMATI aims at the development of an innovative brokerage platform targeting scalable management of heterogeneous distributed and federated resources in order to support mobile cloud applications in challenging scenarios, such as large entertainment events (e.g. concerts or sport events) with a specific focus on nomadic users that interact with their applications when travelling across regional borders.

1.1 BASMATI key entities

The BASMATI platform revolves around three main kinds of concepts: Cloud Resources, applications and users. It is necessary to provide a definition of such concepts in the context and perspective of BASMATI.

1.1.1 Cloud Resources

A **Cloud** is a provider of services and resources large enough to support the illusion of infinite resources (although the federation of Clouds is motivated by the fact that the illusion may not be supported with a single datacenter). As we aim to avoid user lock-in, we place no strong constraint on the technology used in specific Clouds that wish to merge with the BASMATI federation. Hence the BASMATI platform needs to include protocol and API adapters in order to interoperate with diverse Cloud provider technologies. Specific components in the platform architecture are the place for such adapters. As a notable example, we explicitly mention that a full SLA manager may need to be deployed in order to augment specific providers technologies with full support of BASMATI SLAs, allowing integrating resources from that provider into the SLA management hierarchy of BASMATI.

A **Cloudlet** is a smaller Cloud (possibly a single cluster) enrolled in the federation. A Cloudlet is typically placed in the proximity of the end users (i.e. squares of smart cities, or inside shopping malls), and is capable of running applications. Within BASMATI a Cloudlet *may* provide the full set of services offered by a Cloud data center. However, as Cloudlets are small resource pools typically managed by local entities, Cloudlet cannot be expected to provide the wide elastic ranges of resources, as we can expect form a Cloud.

An **Edge resource** is a small computing device integrated within the network hierarchy. They are simple and cheap computational resources, low-performance and IoT-like systems. Edge resources can include e.g. small ARM servers, last-mile systems, routers, as well as laptops and desktops. Edge devices do not support full Cloud features in BASMATI. Being the closest resources to the end-user, edge units are ideal for offloading well-defined computational tasks from mobile resources, for the sake of improved performance or reduced power consumption. The management of the edge part of the platform needs to be decentralized as much as possible, to (i) avoid centralization bottlenecks in resource management and (ii) allow full exploitation of the low-latency, high bandwidth links available with both the end-user devices and the Cloud.

Mobile devices are those typically owned and used by the end-users of Cloud services: laptops, mobile phones, tablets or smart watches are all valid examples. They are all recent mobile devices (GPS, touchscreen), with limited battery/computational power, storage space and constrained connectivity. More powerful hardware items (i.e. laptops) may be considered mobile devices depending on the specific use case.

Resource Type	*Hard Power constr.*	*Wired Network Bandwidth constr.*	*Typical Network Latency to End-User*	*Provider SLA manager.*
Cloud	No	No	High	Yes
Cloudlet	No	No	Medium-High	May provide
Edge		1-10 GB/s	Small	Fixed SLA offers (pre-negotiated)
Mobile	Yes	N/A	0	Self-managed QoE

Table 1. Comparison among Cloud, Cloudlets, edge and mobile resources, with specific constraints of each resource kind outlined; possibly Cloudlet for Back end and Cloudlet for Front-end need to be distinguished.

1.1.2 Applications

The BASMATI platform is aimed at managing and provisioning cloud applications exploiting the resources belonging to the BASMATI cloud federation. BASMATI applications are composed by a **set of interacting services**, materialized either as VMs or containers.

Specifically, in BASMATI, we see the application as a collection of *N* components. A component represents a part of a service, which realizes functionality within the workflow of the application. In order to orchestrate the deployment, in BASMATI we perform an additional step and organize the components into partitions. Each component belongs to only one partition, and each partition can have $1 \leq K \leq N$ components. Ultimately, BASMATI will deal with the deployment of the partitions.

The above representation provides a model of the application in terms of functional features of the application. In addition, BASMATI extends such a model with other pieces of information that drive the placement of the application on cloud resources. We call this extended model the Basmati Enhance Application Model (BEAM). From a practical point of view, a BEAM is a collection of information, materialized as documents, that describe the application as a collection of services and defines rules concerning the selection of resources for deployment.

In particular, the deployment document can be seen as a "colored mask" overlaying the application representation. These documents drive the allocation of the services composing the application by specifying, for each cloud provider, which part(s) of the original application is/are

expected to execute. These documents are also aimed at defining the contextual information associated with the different modules composing the applications (i.e. partitions), to be exploited during the deployment phase. These documents are stored and managed by the Application Repository, which is described in Section 4.1.4.

Furthermore, BASMATI applications assume a clear decomposition of computational resources and data items. Within BASMATI, application data is stored within the BASMATI data plane. The BEAM data plane will need to support an abstraction of data sources that is similar to that of the algorithmic skeletons, allowing the specification of the program semantics, concerning the data, via common patterns of data access.

The Data Plane patterns shall: (i) represent common data access solutions employed within Cloud applications and (ii) allow elastic and dynamic transformation of the application to be performed, to modify resource allocation while preserving application semantics.

It is of paramount importance to assert the goal of providing a templated definition of both computation and data: the deployment system shall have the tools to flexibly *resize* the underlying resource set and application, as well as the knowledge needed to take into account the effect of this *resizing* not only on the performance of each application part but also of the application as a whole. Please note that in this context resizing means the activity devoted to the modification of resource allocation with respect to the default one, either at initial deployment time or dynamically at run-time, according to (and in order to improve) the forecast on execution metrics that are computed by the BASMATI platform.

1.1.3 Users

When discussing about user classes and characteristics for the BASMATI federation, a first definition to be made is that of Federated User.

- A **Federated User** is any user whose identity is granted and recognized by the BASMATI identity services. These in turn may rely on identity providers federated within the BASMATI platform, but technical implications are not pertinent here. For what concerns BASMATI, a user whose identity is not tied to any BASMATI recognized identity service can be regarded as an Anonymous user.

A common issue with Clouds is that service users are typically also service providers, typically for services of a different level (e.g. IaaS users will often providers of PaaS or SaaS services). Within this document and for most of the work done in project deliverables, we will need to consider two classes of user as defined below:

- The **End-user** is the final user of a Cloud application or an Application service. Within BASMATI the end-user is almost always accessing services via a mobile device, so they are a mobile user both at the micro level (local mobility) and at a larger scale (geographic mobility, nomadic behavior). We can assume one or more mobile devices are owned by each end-user and will be used to access different kinds of services (e.g.

Application, Storage, Social networks, Communications, Streamed media). End-users are nevertheless not limited to mobile computing, they will also be able to exploit BASMATI to access Cloud services that they own and control in full (the project use case on Virtual Desktops is an example). An end-user *may* be a federated user.

- The **Cloud User** is the owner of an application, that is, they control an application that provides a service at the PaaS or SaaS level to a set of end-users. A Cloud user is a federated user. A Cloud user exploits the BASMATI platform to run their application service backend on IaaS resources from the federation. The service provider will use one or more cloud providers within the BASMATI federation. The service provider submits a description of its application (intended as a collection of services, with their functional specification) and a QoS that elaborates the non-functional specifications of the application.

We shall underline that we place no strict identity constraints on end-users, that is, an end-user is not necessarily a Federated user. Such a constraint may be enforced when mandated by a specific use case. Whenever such a constraint is needed (e.g. when an End-User is also a Cloud User) we will underline the fact in the description of the application or use case.

1.1.4 Application Backend and Frontend

Within BASMATI we will distinguish applications in two parts:

- *Application Front-End* (FE), executing within a mobile device environment and resources;
- *Application Back-End* (BE), which is a composition of services deployed in the Cloud federation;

This distinction also applies for the offloading concept. Offloading means “moving part of the service execution to a different resource”, in order to improve several (QoS and QoE) metrics of the overall computation, like performance, latency, economic or power cost.

By FE offloading we will mean the delegation of some computational task from a mobile device to a resource in the Federation. FE offloading target resources can belong to a Cloudlet (i.e. a small nearby accessible Cloud, possibly with simplified SLA management) or an edge device. Mobile devices are characterized by limited resources, energy and memory. The offloading support will enable the transparent and efficient exploitation of the networked computational and storage resources available nearby the users to relieve mobile devices from (part of) their workloads. By BE offloading we will instead mean “the action of deploying, migrating and elastically (up/down) scaling service instances across different Clouds”. Here the targets can be fully-fledged Clouds within the BASMATI federation or Cloudlets that expose an SLA management.

The BASMATI architecture provides specific support for both FE and BE offloading, but several issues need to be addressed in order to effectively achieve offloading on very dispersed and

heterogeneous resources. Thus, many components of the BASMATI architecture described in the following, have to deal with one or both forms of offloading.

2 BASMATI and Cloud Resource Providers

The BASMATI brokerage platform is organized and provided as a **collection of cooperating services**. Each service implements a specific feature, as presented in Section 4. Services, by means of their cooperation, provide the scalable management of heterogeneous distributed and federated resources, belonging to different cloud providers, in order to support mobile cloud applications in challenging scenarios supporting nomadic users that interact with their applications when travelling across regional borders.

These services, realizing the BASMATI platform, can be deployed independently; Services use different programming languages and different technologies, their interaction happens by means of well-defined **RESTful APIs**, enabling their interplay, both when deployed within the same cloud provider and when running in different providers. Each service can be either deployed on a cloud or on private, ad-hoc resources as on-premises software.

In fact, BASMATI platform does not pose any specific requirement to cloud providers beyond the ability of providing resources at IaaS level that may be accessed and used **programmatically** (e.g., via RESTful interfaces or other means supporting machine-to-machine interactions). Indeed, all the services realizing the BASMATI platform can be run by using off-the-shelf existing IaaS cloud technologies.

As will be described later in this document, the BASMATI platform embeds adapters specifically devoted to the enablement of interactions with cloud providers for the deployment and execution of BASMATI applications that are structured according to the BASMATI Enhanced Application Model (BEAM) format. Such a format encompasses both the IaaS-level description of the application, the SLA agreements and a few additional documents supporting the deployment and enhancement of the application.

To ensure a proper level of scalability, one or more services contributing to the BASMATI platform can be deployed as multiple instances. Potential issues related to data synchronization and coherence, as well as the management of race conditions, in private service data are implementation dependent and are the responsibility of the service developers. Instead, the management of synchronization related issues are managed by leveraging the **application repository**, a service of the BASMATI platform acting as centralized storage to support the storing and tracking of BASMATI applications.

2.1.1 BASMATI interaction with cloud providers

To realize a cloud federation, the BASMATI Platform has been carefully instrumented with solutions that are able to interact with several different cloud provider interfaces, by means of a flexible and generic provisioning and management sub-system, that is accessible for use by

other services composing the BASMATI platform. Each aspect, that characterizes a cloud provider, is represented by a specific OCCI category allowing discrete and standardized access for use by all higher-level components of the system.

Each BASMATI application is instantiated on cloud providers as a composition of cloud service instances by leveraging OCCI interfaces. These interfaces allow the instantiation of service instances on the different types of providers managed by the BASMATI platform. In the remaining of this section is described the instantiation process.

Cloud service instances are managed by leveraging a combination of information about the service, contract, provision. Such information results from the processing of a customer facing service level agreements that describes the technical, commercial and operational details of the required service. The technical details of the service are provided by the technical manifest and its collection of nodes and configuration actions.

The service level agreement and manifests structures result from the processing of the BASMATI Enhanced Application Model (BEAM) document that was initiated for the construction of an instance of service. The provider information, taken from the service level agreement will be used for the placement of the resources required for the delivery of the final service. A contract instance will be negotiated, for each of the nodes in the manifest, as described by the provider and region terms, and combined with the service instance. The configuration actions will be processed to create installation instructions that will be fused with the corresponding contract.

Using this information, the BASMATI sub-system aimed at interacting with cloud providers, can initiate and manage the complete deployment process.

2.1.2 BASMATI enabling a Multi-cloud environment

By means of the afore mentioned OCCI interfaces, BASMATI is able to support the instantiation of different cloud services on multiple cloud providers, even when are part of the same application.

Currently, the list of the supported providers is the following: Amazon EC2, Amazon ECS, Amazon RDS, Amazon Elastic Beanstalk, Amazon R53, Amazon S3, Google Compute Engine, Google Container Engine, OpenStack NOVA, OpenStack Neutron, OpenStack Glance, Windows Azure, Cloud Sigma, IBM SoftLayer, OpenNebula, SCALR, Compute Next, On APP, Cloud Foundry, Open Shift and Eucalyptus.

As matter of fact, the key core of BASMATI is the union of the brokering and selection subsystem with the provisioning-and-deployment one. Once the brokering subsystem provides the results of the matchmaking process, involving cloud resources and application instances, the deployment and provisioning subsystem is informed via BEAM. Cloud Provider information taken from the deployment service level agreement will be used to drive the placement and subsequent provisioning of the required resources unless overloaded by cloud provider specific

information. In this way the BASMATI platform is able to run applications on multiple clouds enacting the decisions taken by the brokering subsystem.

The provisioning and deployment layer takes on input the deployment documents, as specified in the previous section, and actually performs the deployment process relying on the specific technologies available at each provider. At runtime, the BASMATI platform then keeps track of all the deployments performed, manages and monitors all the cloud services instances deployed on the different providers to take decisions about migration, replication and all the operations that could be requested to be performed in support of compliancy with the terms of the SLAs. All the details about this process will be described later in this document.

The flexible and generic nature of the management structures of the BASMATI platform is perfectly suited to the management not only of multi-cloud provisioning but also multi-paradigm provisioning.

The generic nature of the service provisioning management structure allows resource provisioning to be described and performed, other than the standard Infrastructure as a Service most frequently associated with cloud provisioning. Platform as a Service and Software as a Service provisioning interfaces can be used and selected in this way allowing not only for multi cloud provisioning but also for the provisioning of heterogeneous resource types.

3 BASMATI: Logical architecture

The architecture of the BASMATI platform is organized into four main layers (see Figure 1 as reference) that are here presented from top to bottom.

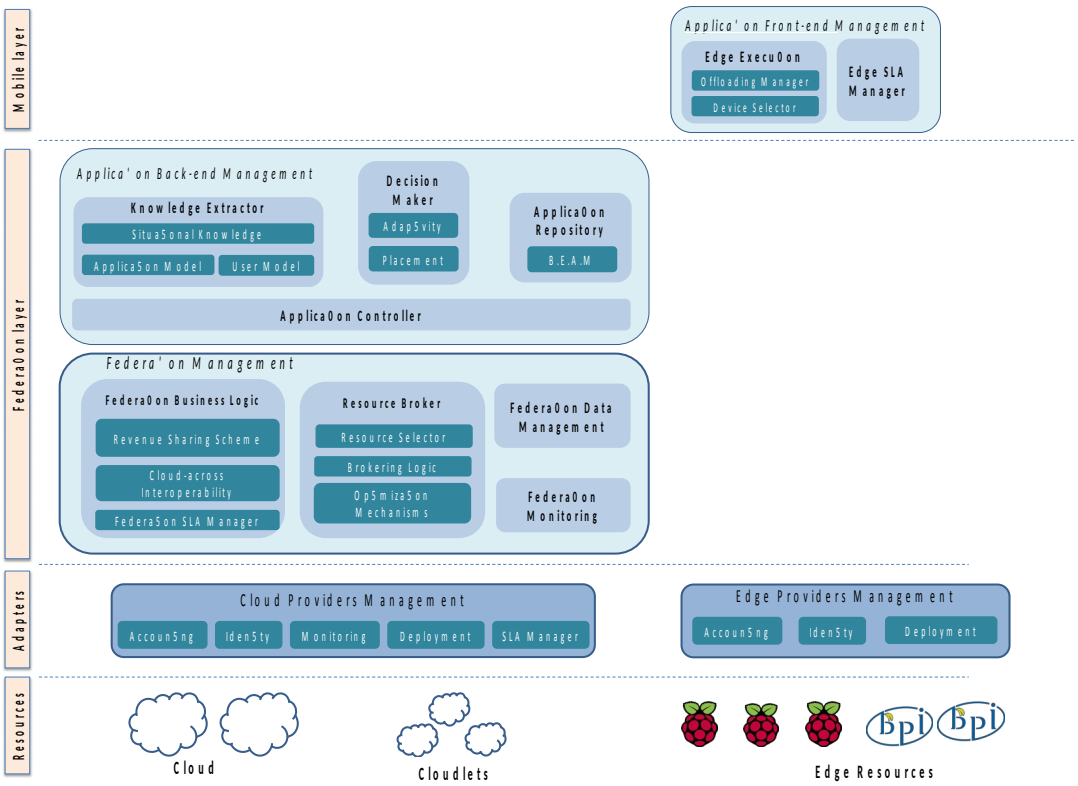


Figure 1. BASMATI high level logical architecture

Mobile layer. It contains components and functionalities that are expected to run on the end-user mobile devices. They interface with the lower layers of the BASMATI platform. It shall be explicitly noted that the mobile layer modules can directly interact with local edge resources.

Federation Layer. Represents the main part concerned with federation support, its modules are expected to run on federation resources. It is further decomposed in the two following logical layers:

Application Back-End Management. These modules manage the back end of a BASMATI application; the services provided are used to support application management directly. It uses the underlying sub-layer as a support for common functionalities.

Federation Management. Provides the specific features of BASMATI federations by building on top of standard Cloud features.

Adapter layer. Software modules here are designed to interface a BASMATI federation with different Cloud providers. They may run on federation resources or in those of the Providers.

Federation Resources layer. Groups together actual resources and services from different Cloud providers, Cloudlets and edge resources. While different resources are grouped here and we attempt to provide a homogeneous abstraction towards the upper layers, not all resources will provide all the features.

3.1 Requirements Coverage

The architecture of BASMATI, and its technologies, have been designed and chosen to cover the requirements identified in the state of the art and use case analysis. In the next tables, we present the list of identified requirements and, for each requirement, the associated components and technologies of the architecture that satisfy it. For a complete description of the requirements, please refer to the Deliverable D2.1 «State of the art and Requirements Analysis».

Module name	***Requirements ID***
Federated Data Management	[TB.SR.4,5,1]; [LE.BPR.3]; [MVD.SR.OMR.3]
Application Controller	[TB.SR.4]; [MVD.SR.SSR.6]; [MVD.SR.OMR.1,2]
Decision Maker	[TB.SR.2,3]; [TB.UR.3]; [LE.BPR.3,6]; [MVD.SR.SSR.3]; [MVD.UR.1]
Resource Broker	[TB.UR.3]; [LE.BPR.5]; [MVD.SR.SSR.3]; [MVD.SR.OMR.10]; [MVD.UR.1];
SLA Manager	[TB.UR.3]; [LE.BPR.3,6]; [MVD.SR.SSR.2,3,4]; [MVD.SR.OMR.2,10]; [MVD.UR.1,3]
Federation Monitoring	[TB.UR.2] [LE.BPR.8] [LE.BPR.2] [MVD.SR.OMR.4,5,7,9] [MVD.SR.OMR.3]
Knowledge Extractor	[TB.UR.1,2]; [LE.UR.1]
User Identity	[MVD.SR.SSR.1]; [MVD.UR.4]
Edge Providers Management (includes monitoring)	[TB.UR.2]; [LE.UR.2];
Front-end Management	[TB.BPR.1]; [LE.BPR.6]

Table 2: Requirements Coverage: components

Technology name	***Requirements ID***
TOSCA	[TB.SR.2,3,4,5]; [LE.BPR.4,6,7]; [MVD.SR.SSR.5]; [MVD.SR.OMR.6,8]; [MVD.UR.2]

Amenesik Cloud Engine	[TB.SR.1,2,3]; [TB.UR.3]; [LE.BPR.2,5,7]; [MVD.SR.SSR.1,2,4,6]; [MVD.SR.OMR.1,3];
Adaptivity Mechanisms	[TB.SR.2,3]
BEAM	[LE.BPR.3]; [MVD.UR.2,3]
OCCI	[MVD.SR.OMR.6]

Table 3: Requirements Coverage: technologies

4 Components and Interactions

This section provides an overview of the components comprising the BASMATI architecture. For each component, the following set of information is provided:

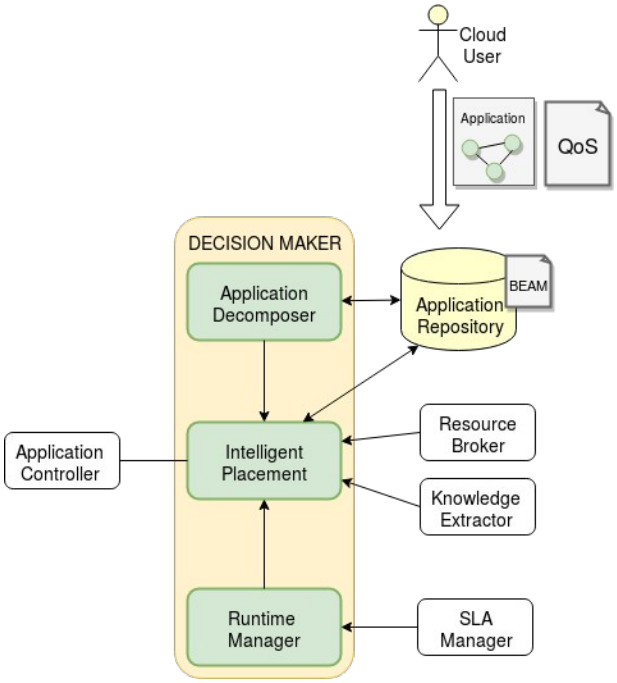
- A general description of the actions/tasks carried out by the component; Brief description of internal sub components (if applicable);
- The Deployment of the component (distributed, centralized), and possible specific hardware requested by the component;
- What kind of information the component is expected to access, manipulate in terms of size and frequency.
- Specific technologies, solutions, software, that will be used in the component.

The structure of the description is focused both on the description of the internal operations of the components, as well as the interactions between them. A high-level snapshot of the interactions can be seen in Figure 2. Details and specification for each interaction are illustrated in the next sections.

Figure 2. Component interactions overview

4.1 Application Back-end Management

4.1.1 Decision Maker



The aim of BASMATI is to provide a complete eco-system able to efficiently deploy and manage cloud applications, especially ones serving requests mainly originated from mobile devices. Consequently, the cloud platform should be able to react to events related to user mobility, such as users crossing national borders. To this end it is of paramount importance to exploit information about previsions on user mobility, as well as pertaining to application nature and structure, to be able to define and enact optimal application deployment. The Decision Maker module is responsible for taking decisions about application placement, taking into account requirements, previsions as well as all the available information on the footprint generated by applications on resources during their past executions. It is also in charge of making major adaptive offloading decisions at runtime that were not resolved at provider level.

The Decision Maker (DM) is organized as a centralized module. The majority of the data exploited by the DM is retrieved from other sources that are the ones in charge of maintaining and managing such information, such the Application Repository (AR).

The DM takes the application submitted by the cloud user, from the AR, and then in cooperation with other modules prepares the BEAM structure. The BEAM is realized by coordinating and consuming the input from the Knowledge Extractor and the Resource Broker. Once the deployment plans have been derived, the *Decision Maker* communicates with the *Application Controller* for enacting the actual deployment plans. The interaction between the decision maker and these components of BASMATI can be summarized in the following Table.

Component	**Description**	**Data**	**Comm. Technology**

Application & Qos Representation	The Application & QoS Representation module provides the application according to the Basmati Enhanced Application Model (B.E.A.M).	TOSCA	**REST**
Knowledge Extractor	The Knowledge Extractor feeds the Decision maker with previsions about users' mobility and application footprints.	**JSON**	**REST**
Resource Broker	The Decision Maker connects with the Broker to retrieve information about the potential resources that are available for hosting the application. The Broker returns a ranked list of resources able to satisfy the Decision Maker request, indicating all the information characterizing the resource, such as price, resource availability, reliability.	**JSON**	**REST**
Application Controller	The Application Controller receives deployment plans and application description from the Decision Maker to take care of all the actions required for the actual deployment of the application.	**JSON**	**REST**

Table 4: interaction between the decision maker and components of BASMATI

As aforementioned, the amount of data effectively stored and managed by the Decision Maker is very limited. Basically, it consists of the information associated with the application it manages and is currently in execution on the cloud federation, possibly annotated with user preferences and requirements. Keeping such information is fundamental to be able to properly react to performance issues and violation, either for providing alternative deployment scenarios or for restructuring the application.

The Decision maker will use a multi-objective optimization algorithm, in which more than one objective function will be minimized or maximized at the same time. An economic-based approach with various cost functions (Time, Energy and Monetary Cost) will be used to provide a set of optimal trade-off solutions. Also, the machine learning approach can be used for the learning feature of the Decision maker from the Past offloading decision.

4.1.2 Knowledge Extractor

Description. The Knowledge Extractor (KE) provides two key features in the BASMATI approach. The first one is the prediction of resource demands based on the user mobility modelling and the application analysis. The second one is the evaluation of a proposed deployment document. Each feature constitutes two separate subcomponents in the KE as depicted in the figure 3. In addition, there are three auxiliary subcomponents for synchronization and support of the main

subcomponents, providing the supervised machine learning models, the compatible knowledge base and the techniques that can represent the incoming data using cohesive and coherent structures.

The predictive and evaluation models built by KE are triggered and exploited by the decision maker in order to drive the mapping of the sessions between users and applications onto resources and evaluate the deployment documents in an efficient way. In order to provide such analysis, it is expected to process a large amount of data and perform scalable data processing functions, both for training (i.e. for machine learning) and prediction purposes.

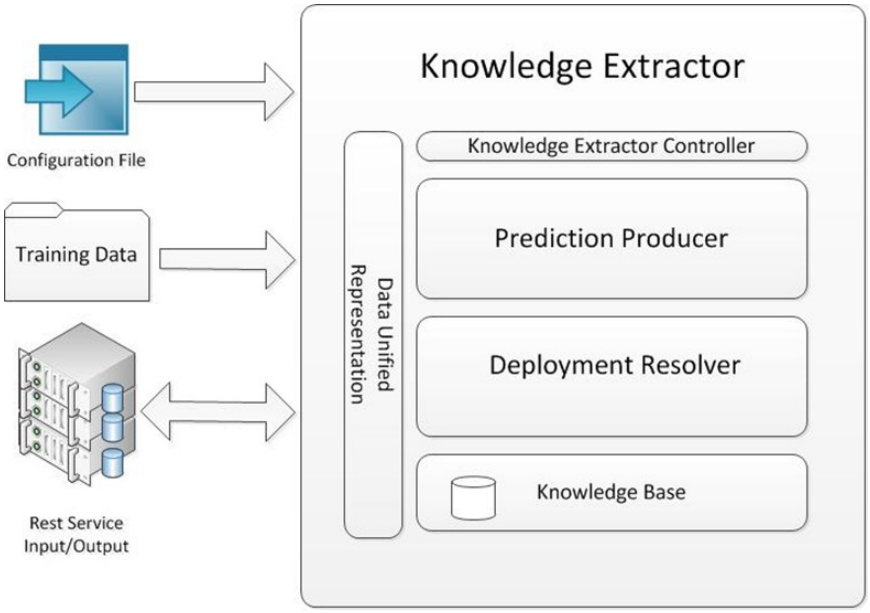


Figure 3: Knowledge Extractor component and its subcomponents

Interaction. For easier access and inter-operability in the Basmati architecture, the intercommunication of the KE with the Decision Maker and the Application Monitoring will be deployed as a RESTful service. A global addressing space is provided by a URI to retrieve and push the corresponding information via JSON files using the PUT, GET operations. The table X describes how the KE communicates with the other Basmati components and the data that are exchanged while the following paragraphs describe the intercommunication of the KE as depicted in the figure 3.

Configuration File

The configuration file is passed to the KE Controller as a header of the input data or training data and specifies:

- The mode in which the KE will work namely Prediction Producer, Deployment Resolver, or Knowledge Base Update
- Input data
- Parameters of interests
- Prediction techniques

Some of these fields can have predefined values. For instance, the prediction producer may use a predefined classification technique that will be used instead of an explicitly specified alternatively available prediction technique.

Training Data

The predictor Producer and Deployment Resolver may use supervised machine learning techniques in which case a training dataset should be provided in order to build their internal knowledge base representation.

Input/Output

The input/output communication, in JSON format, the data to be processed and the output values resulting from predictions. Specifically, this data include the following:

Input data:

- Context_info
- deployment_document

Output data:

- user&app_behaviour_prediction
- plan_acceptance

The REST interface will offer the standard CRUD (Create, Retrieve, Update, Delete) functionality. Data streams can then be sent via PUSH requests through the REST interface. The data will be

analyzed in near-real-time using e.g. Clustering methods. The REST interface of the analyzer will offer adequate methods for the mobile applications, and other services within the BASMATI framework, to pull the results of the analysis via HTTP GET requests. A dedicated interface can be provided for general validation of the data, e.g. consistency and coherence checking.

Component	Description	Data	Comm.Technology
Application Monitoring	An interface of KE with the Application Monitoring in order to take as input the datasets that will be represented and stored in the knowledge base.	Training Data to update the Knowledge Base	REST
Application Monitoring	An interface of KE with the Application Monitoring in order to take as input the observations of users, applications, infrastructures, and environment.	Context Info: user, application, environment, infrastructure data	REST
Decision Maker	Request resource predictions for a user-application-session	The user-application-session Identifier	REST
Decision Maker	An interface of KE with the Decision Maker in order to provide the resource predictions in terms of CPU resource usage, Memory resource needs and Bandwidth.	Resources Predictions Response	REST
Decision Maker	An interface of KE with the Decision Maker in order to take as input a deployment plan.	Deployment Document	REST
Decision Maker	An interface of KE with the Decision Maker in order to	Plan Acceptance	REST

	provide the evaluation of the deployment plan.		

Table 5: Knowledge Extractor *interaction with other BASMATI components*

Deployment. The knowledge extractor component will be implemented in the java language and it will run as a hosted service in a common computer server. The subcomponents can be hosted in the same Server or in different servers using RESTful services. In any case the prediction results will be produced selecting dynamically a different predictive model each time based on the configuration file in a seamless plug and play way.

Data. The Knowledge Extractor component is capable to process any kind of textual, graph and vector data using the unified representation subcomponent that is described in the deliverable D3.1. The input data of Knowledge Extractor will be provided by the Application Monitoring, while the prediction requests and the deployment documents by the Decision Maker. The output predictions will be provided to the Decision Maker. Some of the prediction methods are based on supervised machine learning techniques so a training dataset is used to construct an internal knowledge representation. The training datasets will also be provided by the Application monitoring and stored in compatible files in the same server that hosts the knowledge extractor component.

Technologies. The Knowledge Extractor will use the Weka library[1] and the Scikit-learn library[2], each of which provides a set of feature engineering, classifying, clustering and regression methods. The Weka library is developed in Java language and the Scikit-learn in python. The Python scripts will be embedded in the Java code using a wrapper such as p2j[3] .

Prediction Producer subcomponent

Description. The general task of the subcomponent "Prediction Producer" that is part of the knowledge extractor is to perform analysis of the contextual info that contains data about the users, the applications, the environment and the infrastructure in order to produce predictions that concern the CPU resource usage, the memory resource needs and the bandwidth. This subcomponent, as depicted in Figure 4, will be implemented as a set of several predictors, which use machine learning techniques such as SVM, Bayes Classifiers and decision trees onto vectors that represent the observations of users, applications, environmental and infrastructure info. Based on the input data and the prediction needs, each time the corresponding predictor will be activated.

1 http://www.cs.waikato.ac.nz/ml/weka/
2 http://scikit-learn.org/stable/
3 https://github.com/chrishumphreys/p2j

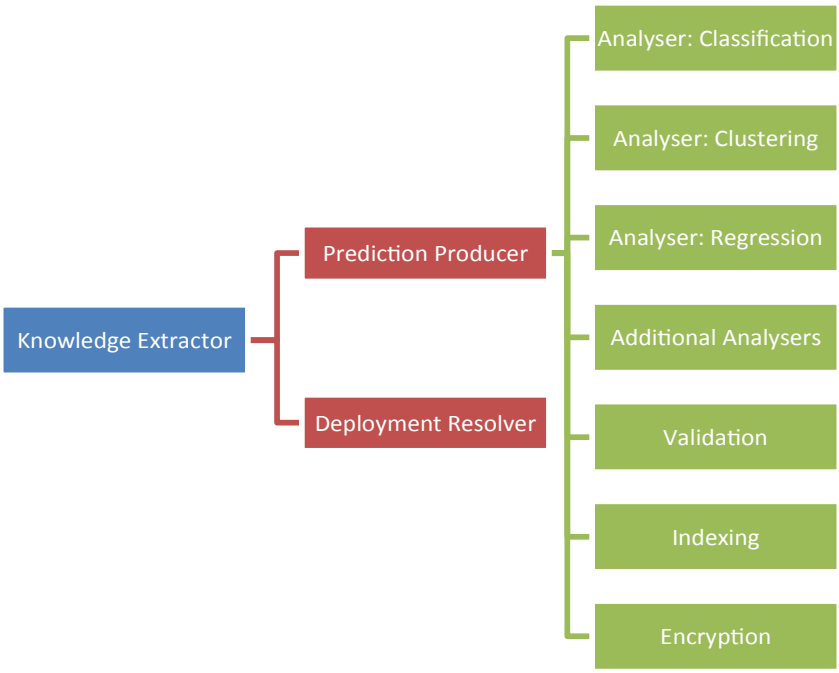


Figure 4: Subcomponents of Prediction Producer

Interface. As described, the subcomponent "Prediction Producer" will be composed by several predictors that will be orchestrated by the KE controller in a unified way, based on the configuration file. The predictors can be hosted in the same or different servers using the Rest interface.

Data. In order to perform the analysis task in an efficient and stable way, the data that is collected needs partly to be stored and buffered on the mobile device because periods with no internet connection or bad connections are assumed – especially in the "Das Fest" use case where several thousands of people will use the same infrastructure. Mobility data will be fed into the "Mobility Understanding and Modelling" component in form of a discrete data stream. For example, GPS coordinates might be tracked and collected every second and then send to the server every 5 seconds – the actual time intervals can be configured and adapted according to the specific needs.

Additionally, statistical data from previous years can be used for predicting user behavior and submitting proposals. Historical data will be pre-structured and pre-analyzed and processed in a slightly different way. The result of the analysis and the data itself will be stored in a NoSQL database like Elasticsearch[4] or MongoDB[5] to ensure stable processing in near-real-time and can be used as training data.

4

5

In order to enhance the accuracy of the geo locations, it is envisaged to additionally use Beacons[6] in some of the BASMATI use cases. Based on a pre-analysis of the accuracy of the geo positions and data, it will be decided if the use of Beacons is useful to enhance the quality and accuracy of the mobility data.

Depending on the use case, additional data can be used within the analysis. In the Das Fest use case for example, geo meta information about POI (points of interest) can be used to optimize the analysis and to set concrete analysis targets. This information is available in json analogue to the format specified by geojson[7].

Deployment Resolver Subcomponent

Description. The Decision Maker will provide a deployment plan that describes how each session, between users and applications, is to be allocated. The KE takes as input the deployment plan and responds if it is found to be feasible. So KE acts as a feedback to the Decision Maker about the execution plans that has been produced. The figure 5 depicts the parts of the Deployment Resolver.

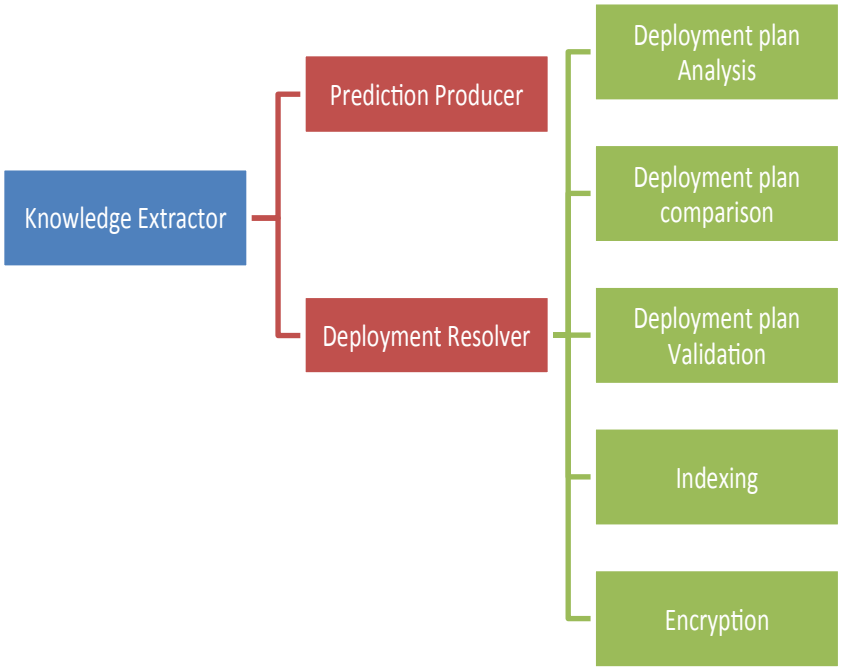


Figure 5: Subcomponents of Deployment Resolver

Data. The Deployment Resolver takes the deployment document through the Knowledge Extractor Controller and the compatible data structures from the Knowledge Base for the needs of the supervised techniques. The deployment document is provided in a JSON file.

6

7

Interface. The Deployment Resolver does not have a separate interface. It is activated directly by the Knowledge Extractor Controller (KEC) based on the provided configuration file.

Auxiliary Subcomponents

Prediction Producer and Deployment Resolver, developed as separate subcomponents inside in the KE architecture as depicted in the figure 3. KE, also include some auxiliary subcomponents that make auxiliary functionalities in order to orchestrate the input data, the parameters of interest, the training data, and the knowledge base using a unified representation according to the configuration file. In the following paragraphs the intercommunication of KE with the other Basmati components and the KE auxiliary subcomponents will be described.

Knowledge Extractor Controller

The Knowledge Extractor Controller (KEC) is the subcomponent of KE that receives the Configuration file and orchestrates all the subcomponents that will be enabled and the actions that will be carried out. Specifically, the KEC is the first subcomponent that receives the input or the training data and determines the data flow and processing through the other KE subcomponents.

Data Unified Representation

The Data Unified Representation (DUR) subcomponent transforms the training data in a form that is compatible with and readable from the predictive models contained in the Prediction Producer and Deployment Resolver. In a similar way the DUR processes the training data in order to be stored in the Knowledge Base. The DUR contains the actions of Data Fusion, Feature Engineering, and Data Normalization producing the Unified Data Structures that represent the data observations in a flexible and unified way.

Knowledge Base

The internal Knowledge Base of KE is a subcomponent that is responsible for the storage and loading of the knowledge representation data that will be used by the Prediction Producer and the Deployment Resolver.

4.1.3 Application Controller

Description. The Application Controller takes care of all the actions required for the coordination of the deployment of the application. The Application controller receives deployment plans from the Decision Maker and processes actions to deploy the applications. To do that, at first the Application Controller generates Action Plans from deployment plans. After that, it communicates with the Cloud Providers Management to deliver the action plans. Then, it notifies the federation SLA Manager for the monitoring of the SLA status of allocated resources and services of the applications. Finally, it stores deployment status.

Component	Description	Data	Comm.

			Technology
Decision Maker	To deploy applications, Decision Maker sends deployment plans to Application Controller	Application description (B.E.A.M.), deployment plan (created by Decision Maker) and initial scripts (supplied by the ASP)	RESTful
Cloud Providers Management	Application Controller converts deployment plans to Action Plans which can be accepted by the Cloud Provider Management. Then Application Controller delivers the action plans to the Cloud Provider Management.	One or more Action Plans (Target Providers and resource types list, SLA information of target resources)	RESTful
Federation SLA Manager	Application Controller notifies information about actual allocated resources and services of the applications to the Federation SLA Manager. The federation SLA manager will use the information to monitor SLA violation.	Target resources list. SLA information of target resources.	RESTful
Application Repository	Application Controller stores deployment status of the applications after deployment of it.	Deployment status of application.	TBD

Table 6: Application Controller interaction with other BASMATI components

Deployment. In general, there is a single instance of the Application Controller for each BASMATI platform, running on the federated cloud infrastructure. The Application Controller acts in a stateless manner. It means that the Application Controller doesn't keep state data. All information about applications, including the deployment status of applications, are stored in the Application Repository.

4.1.4 Application Repository

The Application Repository (AR) stores the high-level information relative to applications, and provides the interfaces to access them (write, read, delete). By its role, the design of the AR is

strongly interfaces with the concepts expressed by the BEAM representation of the application (described in Section 1.1.2).

From a functional point of view, the AR is a passive component, in the sense that little intelligence is required for its operation. Indeed, most of the operations in which the AR is involved concern the access to data of applications by other components. BASMATI applications are indeed represented by means of the TOSCA standard, a dialect (thanks to TOSCA extensibility), defined during the BASMATI project and called BASMATI Enhanced Application Model (BEAM).

In particular, the BEAM is defined as a set of heterogeneous documents that participate towards a comprehensive description of an application, such that it can be operated by the BASMATI platform for deployment and lifecycle management. Some of these documents are provided by the cloud users (e.g. the application topology), others are constructed by the components of the platform is an additive fashion (e.g. Knowledge Extractor, Decision Maker). A summary of these documents is described in the following Table.

Name	Format	Purpose
ID	128-bit UUID	Identifies the instance of an application within the BASMATI environment
Application Topology	TOSCA	Describes the application in terms of components and connection among them
Functional Requirements	TOSCA	Describes the functional requirements of each component of the application
Template Agreement	WS-agreement	Describes the definition of the QoS terms
Decomposition Document	XML/TOSCA	Defines the partitions of the applications and the degree of replication of each partition
Selection Document	XML/TOSCA	Provides a ranking of cloud resources for each partition identified in the decomposition document
Deployment Document	XML/TOSCA	Identifies the mapping between application partition and resources of the federated environment

Table 7: Documents comprising BEAM

A direct consequence of its central role, the AR has connections with many components of the BASMATI platform. A summary of AR interactions can be found in the following table. Finally, the deployment of this component is centralized and will be containerized using Docker.

Component	Description	Data	Comm. Technology
Knowledge Extractor (i.e., Application Profile)	The knowledge extractor takes the application description and updates it with empirical performance data.	Graph representation with TOSCA	REST
Knowledge Extractor (i.e., Application Profile)	The knowledge extractor provides the updated application description with the performance data to the decision maker.	Graph representation with TOSCA	REST
Application Vendor	The cloud user submits the application description (including the SLA templates, application topology and functional requirements) to the AR	Graph representation with TOSCA	REST
Resource Broker	The resource broker stores the selection document and retrieves application information.	TOSCA/XML	REST
Decision Maker	The DM stores the decomposition and the deployment documents and retrieves application information.	TOSCA/XML	REST

Table 8: Application repository interaction with other BASMATI components

4.2 Federation Management

4.2.1 Federation Business Logic

Description. This module embodies the logic and mechanisms required to perform three key tasks: (i) keeping track of the interoperability level between resources of the federation (ii) orchestrating and managing the SLAs of applications and of the members of the federation by driving and configuring the setup of the different SLA managers located inside the providers belonging to the federation. (iii) Enforcing a proper revenue sharing scheme, defined within the federation and implemented by conditioning of the resource selection process.

Component	Description	Data	Comm. Technology
Resource Broker (i.e., Resource Repository)	The resource broker (i.e., resource repository) gets input from the federation logic about the available resources at the participating members of the federation.	Resource description, performance, availability, price	REST
Resource Broker (i.e., Optimal Resource Selection)	The resource broker (i.e., optimal resource selection) gets input about the interoperability and portability of the resources of the cloud providers. This could be realized as a repository as well.	Network of resources, where the nodes are resources and the links represent the parameters that are similar.	Not specified yet
Resource Broker (i.e., Brokering Logic)	The resource broker (i.e., brokering logic) gets input about the revenue sharing plan that applied to different federations.	The data is provided by the SLA repository in JSON/XML format	REST
Federation Monitoring	The federation monitoring provides input to the federation SLA manager, which analyzes this data by comparing it with the input received from the federation agreement.	JSON / XML	REST
Application Controller	The application controller provides information on the state of applications to the cloud provider accounting and invoicing.		REST
User Accounting & Invoicing	User accounting & invoicing data is provided to the cloud provider accounting and invoicing.		
SLA Manager	The SLA managers of each member of the cloud federation reports on the available resources in their clouds.	JSON / XML	REST

Table 9: FBL interaction with other BASMATI components

Deployment. The deployment of the federation logic can be centralized for a single federation. It creates and maintains a database of federation level SLA, including the revenue sharing scheme implemented by the federation members.

Data. The federation logic needs write access to the SLA repository.

4.2.2 Resource Broker

Description. The resource broker role in the BASMATI architecture is twofold. On one side the broker provides tools and mechanisms for the decision about the placement of (part of) application on federated resources. To this end, it exploits optimization mechanisms to perform resource classification that considers feedback on past allocations, cost model objectives and legal aspects. On the other side, the brokering logic organizes the indexing and retrieval of the list of cloud resources available for placement and offloading. These list of cloud resources has been generated by considering the different limitations (such as capacity constraints, availability of resources, and federation agreements between providers) affecting federated heterogeneous resources (including public cloud, cloudlet and edge resources).

Component	***Description***	***Data***	***Comm. Technology***
Decision Maker	The Decision maker sends the request for the application, and the broker returns a possible ranking for allocation	The application topology and requirements contained in the BEAM and taken from Application Repository. The ranking list of resources (Selection Document)	REST
Federation Monitoring	Federation Monitoring sends the state of the available resources along with their metrics	The data can be given in an XML or JSON format.	REST
SLA Manager	The SLA manager feeds the broker with feedback about allocations (e.g. violations)	Violation data: date, time, duration, affected resources, affected applications, etc.	REST
Federation Business Logic	Federation Business Logic sends an input about pricing scheme to the Resource Broker so that it can give more precise ranking lists to Decision Maker.	Pricing scheme for each CSP/resources in format of XML or JSON.	REST

Table 10: Resource Broker interaction with other BASMATI components

Deployment. Ideally, there is an instance of this component in execution on each cloud federation member. Each broker works as an access point toward the brokering interfaces, so the brokers communicate to each other keeping data structures consistent.

Data. The broker component stores different kind of data structure: (i) resources exposed by the provider. This information is relatively dynamic and small, can be kept in a distributed index (such as Distributed Hash T); (ii) feedback on past allocations. This data is used to computed proper prediction models about the quality of resources for a given application. It is expected to accumulate over time and be of considerable size. A centralized federation-level storage is used to keep this data. (iii) pricing scheme for each CSP. This data is used by the Resource Broker, so that it can give better recommendations of resource lists.

Technologies. We plan to evaluate the exploitation of machine learning mechanisms and algorithm to periodically refresh the prediction models.

4.2.3 Federation Data Management

4.2.3.1 Description of General Components

Description. The Federation Data Management component is aiming to unify the data flow inside the BASMATI multi-cloud. Each application based on the BASMATI platform will be using a set of data, which is clearly separated from the application's business logic and processing tasks. That separation provides us with a scalable architecture, providing the data requested by an application when it needs them.

Naturally, in a multi-cloud architecture the problem of polyglot data stores arises. Each application, and the system in which it is deployed, be it a cloud or a single machine, uses a different data store. One application may be using an SQL-based database, another MongoDB and a third HBase. These data stores that are inherently different need to communicate and exchange data with the same data management platform in order to coordinate the applications that are using them or even create new applications that use data from some of them.

The BASMATI Unified Data Management Framework (BUDaMaF) is a service facilitating this process. It unifies the local data stores that each application is using and provide a way for more general application or system administrators to perform some data related tasks that affect the entirety of the BASMATI platform. This unification will also enable us to apply analytics on the data created and used by the applications with no regard to the specific data store they are using. It creates a common database that BASMATI components will use to store commonly used data.

To perform these tasks, BUDaMaF comprises five basic components; the common data store, the core service, the wrappers, the security add-on and the analytics service. The work

contributed by the analytics service is related to the analytical tools that will be applied to the data passing through this unified data framework. The wrappers will handle the communication with each specific data store in the BASMATI environment, translating the queries (and their results) to and from the native languages to the common language. Wrappers are particularly important for application data in order to perform data replication and migration tasks in their local data store structure.

The core service will coordinate all the components, ensuring that they are working in unison. The common data store will keep useful data created and used inside the framework, such as analytics results. Finally, the security add-on will handle data security and privacy protection throughout the BUDaMaF.

Interactions. The BUDaMaF will employ a set of different connection methods with other components of the BASMATI framework. This variance in connection methods is created due to the inherently general purpose and great reach of a data management framework. Also, this variance necessitates the classification of relevant data into three categories:

1. Application Data: The data used and produced by applications during the normal operations of their services. For example, in the TripBuilder use case, this would be the user location or their selected points of interest.
2. Monitoring Data: The data collected by monitoring processes. This data concerns the function of the applications, but they do not use them. For example, they may contain information about CPU usage, response time, Disk usage and others.
3. Federation Data: The data created and used by the BASMATI framework. This data contains information extracted by the components of the framework and used by other internal components, such as the knowledge extractor output.

These categories will ease the process of handling the vast amount of data, by providing greater freedom in two aspects; separating the data into different data stores and creating separate encryption or anonymization tasks for each category.

Component	**Description**	**Data Categories**	**Comm. Technology**
Offloading APIs	An interface that each Wrapper module can implement for each data store supported by the platform.	1 and 2	Restful Web Services
Wrapper	Mediator between the framework	1 and 2	Data store APIs

Module	and each specific data store used by the applications		and Restful Web Services
Core	The core component of the framework	1, 2 and 3	Java method calls and Restful Web Services
Analytics Engine	A component facilitating the communication between the analytics modules and the BUDaMaF core	3	Data store APIs and Restful Web Services
Analytics Module	The basic platform for analytical plugins, that will perform data analysis using the framework.	3	Restful Web Services
Security and Privacy	Encryption and anonymization engine that will care for the protection of all sensitive data moving through the framework	1	Java API, Restful Web Services

Table 11: BUDaMaF interaction with other BASMATI components

Offloading APIs: This component serves as an interface that each wrapper module can implement in order to facilitate the connections to and from each supported data store. The role of this component is to enforce a degree of uniformity in the implementation of each wrapper module, which can be created by different developers over a large time span.

Wrapper Modules: The wrapper modules, which are in the lower levels of the BUDaMaF's architecture, will connect to the data stores remotely, as dedicated users in order to access the data. This connection will be facilitated through java libraries or online APIs, depending on the type of each data store. They will be connected with the core platform using java methods since they will be running on the same machine or cluster of machines.

Core: The core component, in turn, will be exposing an API in the form of a set of restful web services, giving access to all the applications running on the BASMATI platform.

Analytics Engine: This component will facilitate the uniform communication between the BUDaMaF and each analytics module attached to it.

Analytics Module: The analytics module will be interfacing with the analytics engine component in order to perform analysis tasks on the monitoring data gathered in the BASMATI environment. The modules will be loosely coupled to the engine (using Restful web services), easing the addition of modules created using different technologies.

Common Data Store: Finally, the common data store will connect to the core component through a wrapper, just like any other data store, allowing the other components to access it through the API provided by the core component. Its function will be storage of federation data and facilitation of inter-component communication.

The main bulk of the component (core, wrappers, analytics and common data store) will be deployed locally in ICCS machines or in ICCS controlled cloud infrastructure. This will foster the minimization of communication overhead and delays between them, creating a more efficient service. These three basic sub-components will be written in java, using the Weka open source library for data analysis and machine learning. The common data store will be deployed using a scalable DBMS like MongoDB or HDFS.

4.2.4 Federation Monitoring

Description. The Federation Monitoring is a component for integrated monitoring, which collects the monitoring data collected from the Cloud Provider Layer and the monitoring elements generated at the Federation Layer. The BASMATI platform integrates multiple cloud infrastructures to provide cloud broker services or to provide cloud offloading. Therefore, the Federation Monitoring Component collects data from several Application Service Provider (ASP). In order to determine cloud broker or cloud off-loading through Decision Maker or Resource Broker, various analysis' are needed to select SLA or other providers for various Cloud Service Provider (CSP). The Provider Resource Monitoring component collects the SLA data used by the CSP.

Data and Deployment. Monitoring data is collecting by default from the Cloud Service Provider (CSP). The key data provided by the CSP is about the usage of VM in cloud. VM performance metrics collect information such as CPU, memory, and disk usage of VM Instance. Monitoring data consists of the VM performance metrics where the application service is running, and the service process state defined by the ASP.

The type of monitoring data and the collection method are different for each CSP. Therefore, Federation Monitoring defines common items and collects them. The collection method collects data through a separate monitoring agent in the VM Instance used in the CSP. Monitoring agent is distributed together with the resource deployment and resource brokering by the provider deployment in the Federation Layer. Monitoring data collected by the monitoring agent distributed in VM Instance is collected separately from the Provider Resource Monitoring item. Data collected through the Provider Resource Monitoring Component and the Federation Monitoring Component passes the resource state information to the Federation SLA Manager.

Data collected through the monitoring agent provides a platform-integrated monitoring environment for the BASMATI platform administrator through the integrated dashboard. ASP providing mobile service using BASMATI platform provides monitoring information of VM installs used by ASP.

The BASMATI platform administrator manages ASPs that are users. Resource usage of the VM used by the user can be classified according to the user, and service can be classified according to the service provided by ASP. CSP are managed by the BASMATI platform administrator. Important information such as SLA information and SLA compliance provided by CSP are needed. Therefore, administrator should be able to monitor not only the resource usage of the VM but also the information about the CSP.

ASP are users of the BASMATI platform and are not interested in which CSP VMs are provided. Therefore, VM resource usage and service status information are the main concern, monitored, and do not provide information on the CSP.

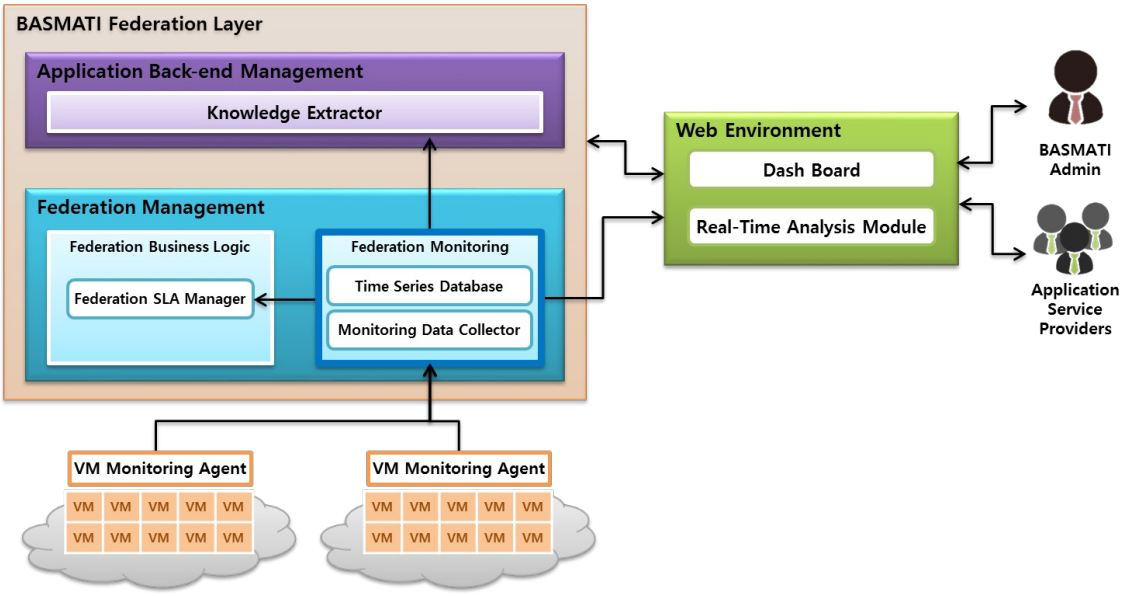


Figure 6 Monitoring Data Collection Process

Interactions. The VM Monitoring Agent collects performance metrics information of the VMs in real time to deliver monitoring data to Federation Monitoring. Federation Monitoring has Monitoring Data Collector that is responsible for each CSP and collects VM information for each CSP and stores it in Time Series Database, in real time. Data stored in the Time Series Database is transferred to Federation SLA Manager, Knowledge Extractor, and Web Environment.

The Web Environment is a web environment for BASMATI platform managers and ASPs that provide services using BASMATI. The Real-Time Analysis Module analyzes real-time data received from Federation Monitoring and provides it to the Dash Board. The Dash Board provides visualization of data analyzed by Real-Time Analysis Module according to the needs of BASMATI manager and ASP.

Component	Description	Data	Comm.

			Technology
Knowledge Extractor		Application and user usage data	REST
Cloud Provider Management	Monitoring components on provider send data to the federation monitoring		*To be decided*

Table 12: Federation Monitoring interaction with other BASMATI components

4.3 Application Frontend Management

4.3.1 Edge Execution

Description. This is the module driving application offloading from smart-phones and tablets to the devices located at the edge of federation, and to the other federation resources. Operatively, it is aimed at determining when and what part of the mobile application to offload.

Component	**Description**	**Data**	**Comm. Technology**
Device Selector	Responsible for finding a nearby computing device at the edge of the federation as well as other federation resources		*To be decided*
Offloading Manager	Responsible for handling the offloading the computation between the mobile device and the edge computing device	JSON	REST
Edge SLA Manger	This sub-component runs on mobile device to determine and define the service level agreement with edge devices to ensure that offloading would not worsen the Quality of Experience (QoE)	JSON	REST

Table 13: Edge Execution interaction with other BASMATI components

Deployment. In terms of deployment, we can expect a single Edge Execution for mobile devices, which can serve for all the instances of BASMATI applications on the same device.

Technologies. In order to determine which part to run locally on the mobile device and which parts to be offloaded from the mobile device to the edge computing device, this component

aims to perform application partitioning. Based on the application modelling, there three possible application partitioning approaches: graph-based, LP-based, or hybrid.

4.3.2 Edge SLA Manager

Description. The Edge SLA Manager is the component, running on mobile devices, aimed at determining and defining service level agreements with offloading targets to ensure that offloading would not worsen the QoE. The Edge SLA manger orchestrates specific QoS aspects associated with the mobile application (e.g. a nano-service for the processing of an image executed within 1 second) which is defined statically by the mobile application developer.

The core of the SLA Manager is flexible enough to work with different monitoring approaches such as simple monitoring systems that must be polled to retrieve the metrics, monitoring systems that are able to push the metrics once available or smart monitoring systems that are able to evaluate the constraints and raise the appropriate violations.

Component	**Description**	**Data**	**Comm. Technology**
Edge SLA Manger	This sub-component runs on mobile device to determine and define the service level agreement with edge devices to ensure that offloading would not worsen the Quality of Experience (QoE)	JSON	REST

Table 14: Edge SLA Manager interaction with other BASMATI components

Technologies. The SLA core is required to be compliant. WS-Agreement specifies an XML structure to define agreements and templates, and two layers interface of web services for operation. This implementation is focused on the xml structure and defines simpler REST interfaces for operations abstracting the complexity of interacting with the xml specification of the WS-Agreement. The afore mentioned API accepts requests in both XML and JSON formats.

4.4 Adapters

4.4.1 Cloud Providers Management

The Cloud Provider Management includes all those mechanisms and components aimed at easing the exploitation of distinct Cloud providers by the federation. Adapters, stubs or partial reimplementation of common services will depend on the API provided by the underlying Cloud providers.

Identity and Accounting. Each provider member of the federation exposes the identity API, which allows other members of the Federation to make use of their offered services so long as

the terms of the agreement governing their offer are respected. This is an essential part of the federation, as it is the backbone for the resource and service pricing schemes which will be used for the identification of the financial transactions required for the invoicing of customers and federation members and for revenue and cost sharing across the Federation.

Deployment. In order to join a federation, cloud providers expose a federated deployment API for use by other members of the Federation, a federation service level agreement SLA as required by the selected topology of the federation and an API for monitoring deployed services. From an operative point of view, the federated deployment provides an encapsulation for the transport and integration of disparate cloud service components irrespective of their source or type of provisioning. The internal sub-components of Provider Deployment are the Generic Deployment Interface, Generic Deployment Description, Deployment Logic, and Deployment Specialization.

The Generic Deployment Interface is the interface through which requests from Federation Logic will be forwarded to Provider Deployment. A RESTful OCCI web service interface is provided to this effect for use in through the Generic Deployment Interface. Authorization of federation member requests is required using the authentication credentials provided in the corresponding service level agreements. Generic Deployment Description provides suitable input from Federation Logic. is based on terms described by the specification of WS-Agreement. Deployment Logic processes elements of the TOSCA/BEAM service template provided on the input. The template defines the topology, node types, relationships between nodes, first-boot scripts, and workflow of the deployments. The Deployment Logic coordinates requests for the resources described in the service template through the Specialized Deployment interfaces which requests provisioning through to corresponding Cloud Service Provider (CSP). This may result in sending multiple Cloud Provider specific requests being sent to the Cloud Provider to ensure the correct deployment and execution of the resources described in service template. All interacts with the different CSPs will be performed through the Specialized Deployment interfaces subsequently protecting the Caller for the different levels of granularity and functionality that they expose.

Monitoring. Basic monitoring items provided by the CSP are limited, and the Monitoring Items provided by each CSP are different. In addition, the type of the Monitoring Component provided by each CSP is also different. For additional monitoring, it provides extensibility by adding ad-on of 3rd-party program (OpenStack), or provides detailed monitoring information at an additional charge in addition to items provided for free by default (Amazon Web Service). Provider Resource Monitoring Component defines the monitoring elements that are shared by CSP. Monitoring data is collected using the Monitoring API provided by the CSP, or the data collected by a separate monitoring agent is transmitted to the Monitoring Collector to collect and store the data. Monitoring elements collected from CSP are collected through Federation Monitoring and used for integrated monitoring. Monitoring items are distributed to the CSP and collect only the resources for the resources of the VM instance in use. In the case of Private Cloud

Solution, the monitoring item and data collection can also be used for the usage of physical infra resources. However, BASMATI platform is not only a Private Cloud Solution but also a Public Cloud Service (ie. AWS ...). Detailed monitoring of the CSP is limited to the level provided by the CSP, and the BASMATI platform does not consider monitoring the CSP infrastructure. Therefore, the BASMATI platform's monitoring focuses on the resources of the VM instance, not the physical assets. In addition, SLA items of virtual resources provided by CSP are solved through Provider SLA Manager.

SLA Manager. When the resources required to satisfy requests for Service, received from customers of a Federation member, are found to be insufficient, unsatisfactory or otherwise unavailable, delegation of provisioning, where permitted in the SLA, will be performed by other suitable members of the BASMATI federation.

Component	**Description**	**Data**	**Comm. Technology**
Federation Deployment	Federation Deployment sends a deployment plan to deploy a service	XML data in the specification of WS-Agreement	REST
Cloud Service Provider/Cloud Management Platform	CSP or CMP receives a service template to deploy resources for a service	XML data int the specification of TOSCA/BEAM	REST

Table 15: SLA Manager interaction with other BASMATI components

4.4.2 Edge providers management

Description. The sparsity of edge resources, as well as the constraints of reduced memory footprint, code complexity and overhead, mandate a different, possibly decentralized implementation of the essential Identity and Accounting services. The Deployment service will support FE-offloading, which is a restricted case of computation offloading, requiring different protocols from those of CPM Deployment.

Component	**Description**	**Data**	**Comm. Technology**

Resource Selector	The Edge Providers Management provides the Resource Selector available edge resources supporting the user's mobility.	Resource description	*N/A*
Edge SLA Manager	The Edge SLA manager running on the user's mobile device defines and determines the SLA agreements with the edge devices to ensure that offloading would not worsen the QoE	QoS requirements/ Edge SLA	*N/A*

Table 16: Edge Provider interaction with other BASMATI components

4.5 Security and Privacy

A full and detailed description of the security mechanism put in place by the BASMATI consortium is presented in Deliverable 5.5 "Service, security and privacy quality enforcement: Design and specification". Here we provide a summary of such mechanisms in relation to security and privacy of data and communication within the components of the architecture.

Application and user data: Applications using the Basmati framework and components for analysis task and cloud resource management will need to ensure by themselves that data that is send to Basmati framework in accordance to general data protection regulations. This means e.g. that data sets are referenced by arbitrary IDs instead of concrete IP addresses (associated to the device that sends the data) and that the data that is to be analyzed is not mixed up with additional data like login information or other user specific data.

For the use cases within Basmati, the client applications like the "Das Fest" app will gather an explicit agreement from the user to ensure that he commits to the transport of his data in a pseudonymized form which means in this concrete use case that all unique identifiers like IP address or names are replaced by arbitrary IDs. The explicit confirmation by the user is necessary since geo trajectories are so diverse that nearly each data set will at the end be unique which in principal makes it possible to infer the actual sender - at least given some additional information.

Data storage: As mentioned in chapter 4.2.3.1, the actual data store used by the application that consumes the Basmati framework, can differ. Here also, adequate data protection strategies and mechanisms need to be implemented by the application developer and provider himself. From Basmati perspective, the main aim is to ensure that the same data protection is still valid from the moment where Basmati gets access to the data.

Communication: Transport of data from an application client to the server should be done with HTTP over TLS[8]. For the Basmati server we will create a CSR and the according certificate will be installed in the web server for secured communication with the basmati server.

8 https://en.wikipedia.org/wiki/HTTPS

Data access: In order to access data for analysis task, the data needs to be accessed. Basmati will offer simple but secure access right and permission system to

(a) Define roles and permissions for data stores. It should be possible to define that a dedicated data stored can be accessed by a user or application with role X
(b) Let either a user or an application be assigned with role X
(c) Define read/write/execute(delete) permissions on data stores
(d) Delete roles/users/permissions.

5 Sequence Diagrams

5.1 Application deployment

This section illustrates the interactions among the components when an application is submitted to the BASMATI federation. In the next figure, we assume the cloud user already provided the application description (topology and functional requirements), as well as service level requirements. This information has been stored in the Application repository.

Once the Decision Maker (DM) is informed about a new application, the DM contacts the application repository to gather the information submitted by the cloud user. It then integrates the application with the input from the Knowledge Extractor with information about user, application modelling and situational awareness to derive a way to decompose the application in partitions. The result of this activity, the decomposition document, is stored in the Application Repository (AR).

Subsequently, the DM triggers the Resource Broker to retrieve, for each module composing the application, a ranked list of potential candidates. The Resource Broker considers actual information about cloud provider resources and feedback on past allocations, which are provided by the SLA manager. The result of this activity is the selection document, which is also stored in the AR.

Once the selection document is ready, the control goes back to the DM that prepares a series of deployment plans. These plans are then inserted into the AR, upon a verification of their correctness by the Knowledge Extractor. Once the plans are ready and verified, the application partitions are deployed into the selected resources by means of the Application Controller, which contacts the ACE engine to proceed with the actual deployment that involves the monitoring and the SLA components.

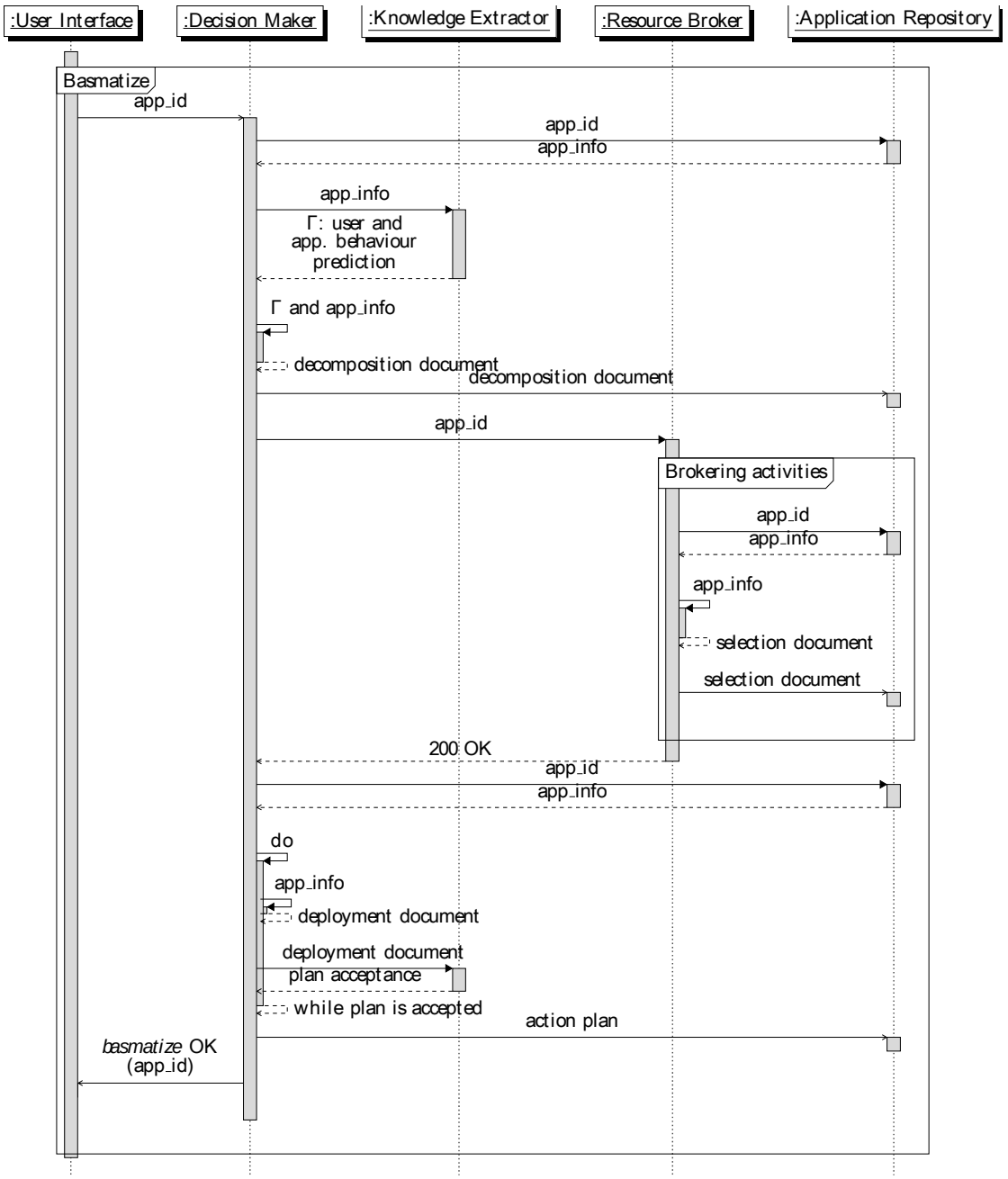


Figure 7. Application Deployment information flow

5.2 Application reconfiguration

Violations occur when an SLA Managers (either the Federation SLA Manager or one of the SLA Managers within the providers) notice a behavior of (a part of) the application that does not comply with the service level agreed with the cloud user. A sequence diagram of the interactions among the BASMATI components in case of violations is given in Figure 8. In terms of scope, we consider two types of violations, local and global.

Local violation. This violation is discovered and managed locally by an SLA Manager within a cloud provider (PSLAM). In this case, when the PSLAM detects a violation that can be addressed locally it contacts the provider's deployment module to perform the necessary adjustments.

Global violation. This violation is discovered and managed by the Federation SLA Manager, either via communication from the PSLAM or by analyzing data from the Federation Monitoring. The FSLAM raises up the violation to the Application Controller, which checks whether one of the alternative plans can fit to restore the application behavior in accordance with the SLA. If no backup plan is acceptable, the Application Controller contacts the Decision Maker and the Resource Broker asking for a new set of plans (interaction 4). Subsequently, the Application Controller sends the plan to the ACE Manager, which realizes it by contacting the necessary cloud providers.

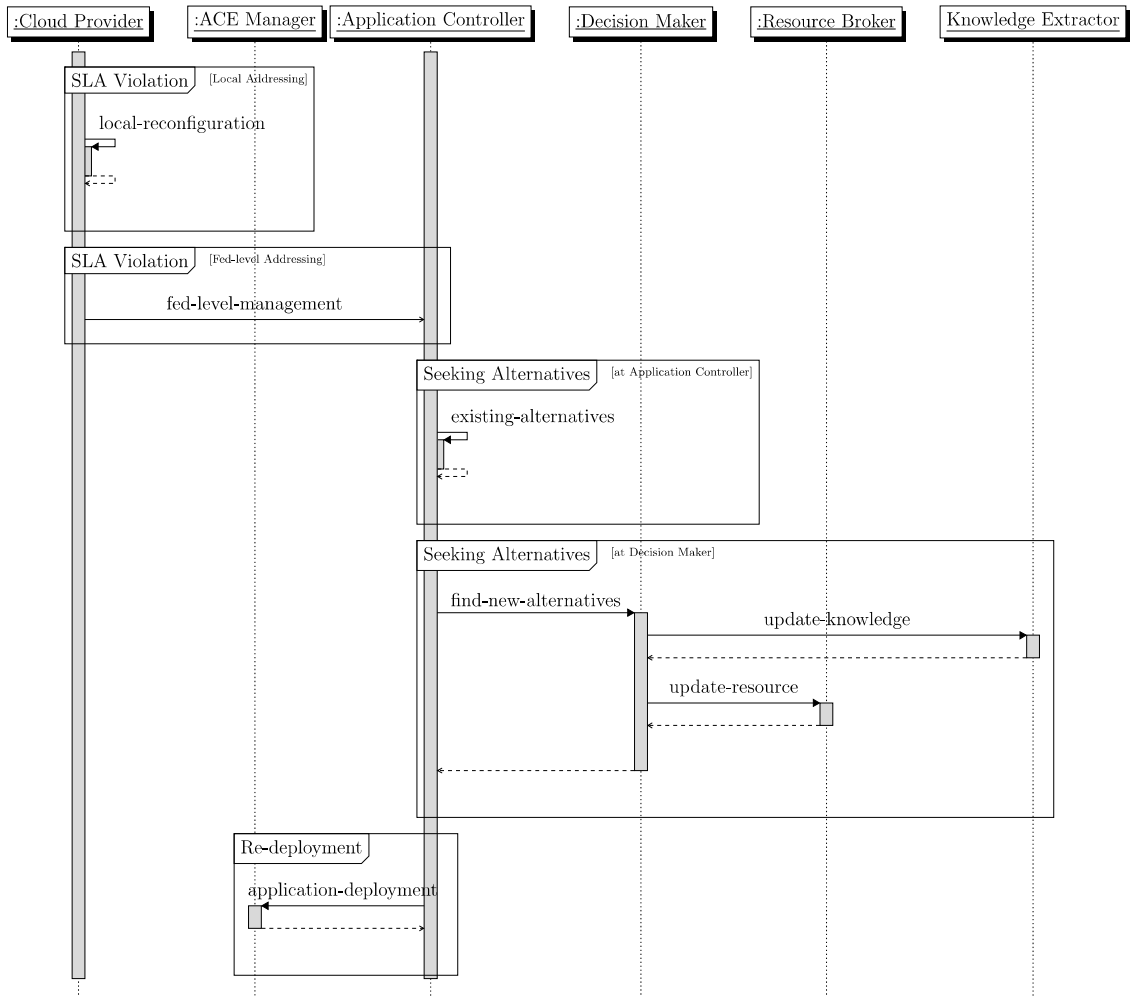


Figure 8. Violation management information flow

6 Conclusions

This deliverable presents the architecture of the BASMATI platform. In addition to a description of each of the components of the architecture, the deliverable provides sequence diagrams that highlight the interactions between the components in the context of application placement and runtime configuration. The deliverable also introduces the main concepts behind the BASMATI platforms as well as a mapping that connects the requirements to the mechanisms of the architecture that satisfy them. In terms of design, the presented architecture functions as a "container" that has paved the way for the implementation specific solutions and technologies towards applications. Several of these solutions are already presented in the current version of the deliverable. In particular, the sequence of actions for application deployment and the runtime reconfiguration of applications have already been defined.

This deliverable outlines the state of work of the various components at the moment of its planned release in November 2017 (M18). The deliverable has been subsequently updated one year later, in November 2018. During this time, although the general schema of the architecture remained substantially the same, the interaction between components and the application workflows had been subject to improvements and fine-tuning. Specifically, a proper integration of edge resources has been performed by bringing part of the ACE platform to the edge devices, such as to make them part of the federation. This aspect has been tested in the DAS FEST event in July 2018, where around 25 edge devices were working as part of a heterogeneous pool of resources. Further, the flows for the application submission and runtime control has been finalized. To this regard, different proofs of concept have been created for each of the use cases.